

1. GWDG iRODS/Handle Interface Client-Server Application.**2. Exception Types (excptntp.web).**

Log

[LDF 2012.07.13.] Added this file.

3. Include files.

```
<Include files 3> ≡
#ifndef _XOPEN_SOURCE
#define _XOPEN_SOURCE
#endif
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <sys/types.h>
#include <dirent.h>
#include <string.h>
#include <pwd.h>
#include <errno.h>
#include <pthread.h>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <iostream>
#include <string>
#include <sstream>
#include <deque>
#include <map>
#include <vector>
#if HAVE_CONFIG_H
#include <config.h>
#endif
#include "glblcnst.h++"
#include "glblvrbl.h++"
```

See also sections 8, 31, 308, 323, 547, 563, 701, 751, 886, 995, 1037, 1261, 1337, 1369, 1412, 1426, 1444, 1460, 1479, 1497, 1515, 1547, 1579, 1627, 1660, 1675, 1715, 1789, and 1801.

This code is cited in section 1034.

This code is used in sections 5, 27, 305, 320, 544, 560, 698, 748, 883, 992, 1034, 1258, 1333, 1366, 1409, 1423, 1441, 1457, 1476, 1494, 1512, 1544, 1576, 1624, 1657, 1672, 1712, 1786, 1798, and 1835.

4. struct Initialize_Exception_Type.

Log

[LDF 2012.07.13.] Added this **struct** declaration.

```
<Declare struct Initialize_Exception_Type 4> ≡
struct Initialize_Exception_Type { };
```

This code is used in sections 5 and 6.

5. Putting utility functions together. [LDF 2008.12.05.]

```
<Include files 3>
using namespace std;
using namespace gwrdifpk;
<Declare struct Initialize_Exception_Type 4>
```

6. This is what's written to the header file `excptntp.h`. [LDF 2008.12.05.]

```
<excptntp.h 6>≡
#ifndef EXCPTNTP_H
#define EXCPTNTP_H 1
using namespace std;
using namespace gwrdifpk;
< Declare struct Initialize_Exception_Type 4>
#endif
```

7. class Response_Type.

8. Include files.

```
<Include files 3>+≡
#include <stdlib.h> /* Standard Library for C */
#include <stdio.h>
#if 0
#include <errno.h>
#endif
#include <fstream> /* Standard Template Library (STL) for C++ */
#include <iomanip>
#include <ios>
#include <iostream>
#include <map>
#include <set>
#include <deque>
#include <string>
#include <vector>
#if 0
#include <time.h>
#include <math.h>
#endif
#include <sstream>
#include <pthread.h> /* POSIX threads */
#include <mysql.h>
#if HAVE_CONFIG_H
#include <config.h>
#endif
#include "gblcnst.h++"
#include "gblvrb1.h++"
#include "hndlvltp.h++"
#include "rspnstp.h++"
#include "irdsavtp.h++"
#include "irdsobtp.h++"
#include "hndltype.h++"
```

9. class Response_Type. [LDF 2012.07.30.]

Log

[LDF 2012.07.30.] Added this type declaration.

[LDF 2012.10.16.] Added `static const unsigned int SEND_HANDLE_TYPE` and `Handle_Value_Type *handle`.

[LDF 2012.11.22.] Added `unsigned int pid_options` and the `string` data members `pid_str`, `pid_prefix_str`, `pid_suffix_str`, `pid_institute_str` and `temporary_filename`.

[LDF 2012.11.27.] Added `static const unsigned int LS_TYPE`.

[LDF 2012.12.13.] Added `static const unsigned int SEND_METADATA_TYPE`.

[LDF 2012.12.31.] Added `unsigned int metadata_options`.

[LDF 2013.03.06.] Added `friend` declaration for `client_sending_file_rule_func`.

[LDF 2013.04.03.] Added data members `static const unsigned int END_SERVER_TYPE`, `static const unsigned int PWD_TYPE`, `static const unsigned int CD_TYPE` and `string dirname`.

[LDF 2013.04.04.] Added `static const unsigned int GET_TYPE`.

[LDF 2013.04.19.] Added `static const unsigned int SLEEP_TYPE` and `int int_val`.

[LDF 2013.04.25.] Added `vector<string> string_vector`.

[LDF 2013.04.25.] Added `static const unsigned int MKDIR_TYPE`.

[LDF 2013.05.03.] Added `static const unsigned int SHOW_CERTIFICATE_TYPE`.

[LDF 2013.05.22.] Added `friend` declarations for `distinguished_name_rule_func` and `get_user_info_func`.

[LDF 2013.05.23.] Added the following `static const unsigned int` data members: `RM_TYPE`, `GET_METADATA_TYPE`, `GET_HANDLE_TYPE`, `SEND_TAN_LIST_TYPE`, `PROCESS_PENDING_TYPE`.

[LDF 2013.05.24.] Made `unsigned int pid_options` and `unsigned int metadata_options` separate variables. Formerly, they were in a union.

[LDF 2013.05.24.] Added `static const unsigned int GET_USER_INFO_TYPE`, `static const unsigned int CREATE_HANDLE_TYPE`, `string string_val` and `bool no_delay`.

[LDF 2013.05.27.] Added `static const unsigned int MAX_RESPONSE_TYPE`.

[LDF 2013.06.15.] Added `static const unsigned int ADD_HANDLE_VALUE_TYPE`.

[LDF 2013.06.16.] Added private data member `Handle_Value_Triple hvt`.

[LDF 2013.07.04.] Added `static const unsigned int DELETE_HANDLE_TYPE`.

[LDF 2013.08.07.] Added `unsigned int options` and `unsigned long int delay_val`.

[LDF 2013.08.15.] Replaced `static const unsigned int RM_TYPE` with `UNDELETE_FILE_TYPE`. The former was no longer being used.

[LDF 2013.08.21.] Added `static const unsigned int UNDELETE_HANDLE_TYPE`.

[LDF 2013.08.26.] Renamed `unsigned long int delay_val` to `delay_value` to match `Scan_Parse_Parameter_Type::delay_value`.

[LDF 2013.08.30.] Added `static const unsigned int DELETE_HANDLE_VALUE_TYPE` and `static const unsigned int UNDELETE_HANDLE_VALUE_TYPE`.

[LDF 2013.09.11.] Added `friend` declaration for class `Handle_Value_Type`.

```
< class Response_Type declaration 9 > ≡
class Response_Type {
    friend int yyparse(yyscan_t parameter);
    friend class Scan_Parse_Parameter_Type;
    friend class Irods_Object_Type;
    friend class Handle_Value_Type;
    friend int exchange_data_with_server(Scan_Parse_Parameter_Type &);
    friend int exchange_data_with_client(Scan_Parse_Parameter_Type &);
    friend int client_sending_file_rule_func(Scan_Parse_Parameter_Type *param, string filename, int reference);
    friend int distinguished_name_rule_func(Scan_Parse_Parameter_Type *, const char *);
    friend int get_user_info_func(Scan_Parse_Parameter_Type *, const char *);
private: unsigned int type;
```

```
string local_filename;
string remote_filename;
string temporary_filename;
string dirname;
string command;
string flags;
unsigned int options; /* For general use. [LDF 2013.08.07.] */
unsigned int metadata_options;
unsigned int pid_options;
string pid_str;
string pid_prefix_str;
string pid_suffix_str;
string pid_institute_str;
Handle_Type * handle;
Handle_Value_Triple hvt;
int int_val;
string string_val;
bool no_delay;
unsigned long int delay_value;
vector<string> string_vector;
public: static map<unsigned int, string> typename_map;
static const unsigned int NULL_RESPONSE_TYPE;
static const unsigned int COMMAND_ONLY_TYPE;
static const unsigned int SEND_FILE_TYPE;
static const unsigned int RECEIVE_PUT_FILE_TYPE;
static const unsigned int RECEIVE_METADATA_FILE_TYPE;
static const unsigned int SEND_HANDLE_TYPE;
static const unsigned int LS_TYPE;
static const unsigned int PWD_TYPE;
static const unsigned int CD_TYPE;
static const unsigned int MKDIR_TYPE;
static const unsigned int UNDELETE_FILE_TYPE;
static const unsigned int MARK_IRODS_OBJECTS_FOR_DELETION_TYPE;
static const unsigned int GET_TYPE;
static const unsigned int SEND_METADATA_TYPE;
static const unsigned int END_SERVER_TYPE;
static const unsigned int SLEEP_TYPE;
static const unsigned int SHOW_CERTIFICATE_TYPE;
static const unsigned int GET_METADATA_TYPE;
static const unsigned int GET_HANDLE_TYPE;
static const unsigned int SEND_TAN_LIST_TYPE;
static const unsigned int PROCESS_PENDING_TYPE;
static const unsigned int GET_USER_INFO_TYPE;
static const unsigned int CREATE_HANDLE_TYPE;
static const unsigned int ADD_HANDLE_VALUE_TYPE;
static const unsigned int DELETE_HANDLE_TYPE;
static const unsigned int UNDELETE_HANDLE_TYPE;
static const unsigned int DELETE_HANDLE_VALUE_TYPE;
static const unsigned int UNDELETE_HANDLE_VALUE_TYPE;
static const unsigned int MAX_RESPONSE_TYPE;
⟨ Response_Type function declarations 12 ⟩
```

```
};
```

This code is used in section 29.

10. Initialize static Response_Type member variables. [LDF 2012.07.30.]

```
< Initialize static constant Response_Type member variables 10 > ≡
const unsigned int Response_Type::NULL_RESPONSE_TYPE = 0;
const unsigned int Response_Type::COMMAND_ONLY_TYPE = 1;
const unsigned int Response_Type::SEND_FILE_TYPE = 2;
const unsigned int Response_Type::RECEIVE_PUT_FILE_TYPE = 3;
const unsigned int Response_Type::RECEIVE_METADATA_FILE_TYPE = 4;
const unsigned int Response_Type::SEND_HANDLE_TYPE = 5;
const unsigned int Response_Type::LS_TYPE = 6;
const unsigned int Response_Type::PWD_TYPE = 7;
const unsigned int Response_Type::CD_TYPE = 8;
const unsigned int Response_Type::MKDIR_TYPE = 9;
const unsigned int Response_Type::UNDELETE_FILE_TYPE = 10;
const unsigned int Response_Type::MARK_IRODS_OBJECTS_FOR_DELETION_TYPE = 11;
const unsigned int Response_Type::GET_TYPE = 12;
const unsigned int Response_Type::SEND_METADATA_TYPE = 13;
const unsigned int Response_Type::END_SERVER_TYPE = 14;
const unsigned int Response_Type::SLEEP_TYPE = 15;
const unsigned int Response_Type::SHOW_CERTIFICATE_TYPE = 16;
const unsigned int Response_Type::GET_METADATA_TYPE = 17;
const unsigned int Response_Type::GET_HANDLE_TYPE = 18;
const unsigned int Response_Type::SEND_TAN_LIST_TYPE = 19;
const unsigned int Response_Type::PROCESS_PENDING_TYPE = 20;
const unsigned int Response_Type::GET_USER_INFO_TYPE = 21;
const unsigned int Response_Type::CREATE_HANDLE_TYPE = 22;
const unsigned int Response_Type::ADD_HANDLE_VALUE_TYPE = 23;
const unsigned int Response_Type::DELETE_HANDLE_TYPE = 24;
const unsigned int Response_Type::UNDELETE_HANDLE_TYPE = 25;
const unsigned int Response_Type::DELETE_HANDLE_VALUE_TYPE = 26;
const unsigned int Response_Type::UNDELETE_HANDLE_VALUE_TYPE = 27;
const unsigned int Response_Type::MAX_RESPONSE_TYPE = 27; /* MAX_RESPONSE_TYPE should
always have the highest value assigned to another Response_Type constant. [LDF 2013.05.27.] */
map<unsigned int, string> Response_Type::typename_map;
```

This code is used in section 27.

11. Response_Type_Functions. [LDF 2012.07.30.]

12. Default constructor. [LDF 2012.07.30.]

```
< Response_Type function declarations 12 > ≡
Response_Type(void);
```

See also sections 14, 16, 18, 20, 22, and 24.

This code is used in section 9.

13.

```
< Response_Type constructor definitions 13 > ≡
Response_Type::Response_Type(void)
{
    type = NULL_RESPONSE_TYPE;
    options = metadata_options = pid_options = 0U;
    handle = 0;
    int_val = 0;
    delay_value = 0UL;
    no_delay = false;
    return;
} /* End of Response_Type default constructor definition */
```

See also section 15.

This code is used in section 27.

14. Copy constructor. [LDF 2013.03.01.]

Log

[LDF 2013.05.24.] Now calling assignment operator.

```
< Response_Type function declarations 12 > +≡
Response_Type(const Response_Type &r);
```

15.

```
< Response_Type constructor definitions 13 > +≡
Response_Type::Response_Type(const Response_Type &r)
{
    operator=(r);
    return;
} /* End of Response_Type copy constructor definition */
```

16. Destructor. [LDF 2012.09.06.]

Log

[LDF 2012.09.06.] Added this function.

```
< Response_Type function declarations 12 > +≡
~Response_Type(void);
```

17.

```
< Response_Type destructor definition 17 > ≡
Response_Type::~Response_Type(void)
{
    string_vector.clear();
    if (handle) delete handle;
    return;
} /* End of Response_Type destructor definition */
```

This code is used in section 27.

18. Assignment operator. [LDF 2013.05.24.]

`<Response_Type function declarations 12> +≡
const Response_Type &operator=(const Response_Type &r);`

19.

`<Response_Type ::operator= definition 19> ≡
const Response_Type &Response_Type ::operator=(const Response_Type &r)
{
 type = r.type;
 local_filename = r.local_filename;
 remote_filename = r.remote_filename;
 temporary_filename = r.temporary_filename;
 dirname = r.dirname;
 command = r.command;
 flags = r.flags;
 int_val = r.int_val;
 delay_value = r.delay_value;
 string_val = r.string_val;
 no_delay = r.no_delay;
 pid_str = r.pid_str;
 pid_prefix_str = r.pid_prefix_str;
 pid_suffix_str = r.pid_suffix_str;
 pid_institute_str = r.pid_institute_str;
 hvt = r.hvt;
 options = r.options;
 metadata_options = r.metadata_options;
 pid_options = r.pid_options;
 if (r.handle) {
 handle = new Handle_Type;
 *handle = *(r.handle);
 }
 else handle = 0;
 string_vector.clear();
 string_vector = r.string_vector;
 return r;
} /* End of Response_Type ::operator= definition */`

This code is used in section 27.

20. Initialize maps. [LDF 2012.10.02.]

Log

[LDF 2012.10.02.] Added this function.

`<Response_Type function declarations 12> +≡
static int initialize_maps(void);`

21.

```
< Response_Type function definitions 21 > ≡
int Response_Type::initialize_maps(void)
{
    typename_map[0] = "NULL_RESPONSE_TYPE";
    typename_map[1] = "COMMAND_ONLY_TYPE";
    typename_map[2] = "SEND_FILE_TYPE";
    typename_map[3] = "RECEIVE_PUT_FILE_TYPE";
    typename_map[4] = "RECEIVE_METADATA_FILE_TYPE";
    typename_map[5] = "SEND_HANDLE_TYPE";
    typename_map[6] = "LS_TYPE";
    typename_map[7] = "PWD_TYPE";
    typename_map[8] = "CD_TYPE";
    typename_map[9] = "MKDIR_TYPE";
    typename_map[10] = "UNDELETE_FILE_TYPE";
    typename_map[11] = "MARK_IRODS_OBJECTS_FOR_DELETION_TYPE";
    typename_map[12] = "GET_TYPE";
    typename_map[13] = "SEND_METADATA_TYPE";
    typename_map[14] = "END_SERVER_TYPE";
    typename_map[15] = "SLEEP_TYPE";
    typename_map[16] = "SHOW_CERTIFICATE_TYPE";
    typename_map[17] = "GET_METADATA_TYPE";
    typename_map[18] = "GET_HANDLE_TYPE";
    typename_map[19] = "SEND_TAN_LIST_TYPE";
    typename_map[20] = "PROCESS_PENDING_TYPE";
    typename_map[21] = "GET_USER_INFO_TYPE";
    typename_map[22] = "CREATE_HANDLE_TYPE";
    typename_map[23] = "ADD_HANDLE_VALUE_TYPE";
    typename_map[24] = "DELETE_HANDLE_TYPE";
    typename_map[25] = "UNDELETE_HANDLE_TYPE";
    typename_map[26] = "DELETE_HANDLE_VALUE_TYPE";
    typename_map[27] = "UNDELETE_HANDLE_VALUE_TYPE";
    return 0;
}
```

See also sections 23 and 25.

This code is used in section 27.

22. Clear. [LDF 2012.11.22.]

Log

[LDF 2012.11.22.] Added this function.

```
< Response_Type function declarations 12 > +≡
void clear(void);
```

23.

```
⟨ Response_Type function definitions 21 ⟩ +≡
void Response_Type::clear(void)
{
    type = NULL_RESPONSE_TYPE;
    ;
    options = metadata_options = pid_options = 0U;
    dirname = local_filename = remote_filename = temporary_filename = command = flags = pid_str =
        pid_prefix_str = pid_suffix_str = pid_institute_str = "";
    handle = 0;      /* Not deleted! It may be referenced by another object. [LDF 2012.11.22.] */
    int_val = 0;
    delay_value = 0UL;
    string_val = "";
    no_delay = false;
    string_vector.clear();
    return;
} /* Response_Type::clear */
```

24. Show. [LDF 2012.07.30.]

Log

[LDF 2012.11.22.] Made this function **const**.

```
⟨ Response_Type function declarations 12 ⟩ +≡
int show(string s = "") const;
```

25.

```

⟨ Response_Type function definitions 21 ⟩ +≡
int Response_Type::show(string s) const
{
    if (s.size() == 0) s = "Response_Type:";  

    cerr << s << endl;  

    cerr << "'type'" << typename_map[type] << "," << endl <<
        "'local_filename'" << local_filename << endl << "'remote_filename'" <<
        remote_filename << endl << "'temporary_filename'" << temporary_filename << endl <<
        "'dirname'" << dirname << endl << "'command'" << command << endl <<
        command << endl << "'flags'" << flags << endl <<
        oct << "options" << endl << options << "(octal)" << endl <<
        "pid_options" << endl << pid_options << "(octal)" << endl <<
        "metadata_options" << metadata_options << "(octal)" << endl << dec <<
        "pid_str" << endl << pid_str << endl << "pid_prefix_str" << endl <<
        pid_prefix_str << endl << "pid_suffix_str" << endl << pid_suffix_str << endl <<
        "pid_institute_str" << endl << pid_institute_str << endl << "int_val" << endl <<
        int_val << endl << "delay_value" << endl << delay_value << endl <<
        "string_val" << endl << string_val << endl << "no_delay" << endl <<
        no_delay << endl;  

    if (handle) handle->show("handle:");  

    else cerr << "handle=NULL" << endl;  

    cerr << "string_vector.size()" << endl << string_vector.size() << endl;  

    if (string_vector.size() > 0) cerr << "Strings:" << endl;  

    for (vector<string>::const_iterator iter = string_vector.begin(); iter != string_vector.end(); ++iter)  

        cerr << " " << *iter << endl;  

    return 0;  

} /* Response_Type::show */

```

26. Putting `rspnstp.web` together.

27. This is what's compiled.

```

using namespace std;  

⟨ Include files 3 ⟩  

using namespace gwrdifpk;  

⟨ Initialize static constant Response_Type member variables 10 ⟩  

⟨ Response_Type constructor definitions 13 ⟩  

⟨ Response_Type destructor definition 17 ⟩  

⟨ Response_Type::operator= definition 19 ⟩  

⟨ Response_Type function definitions 21 ⟩

```

28. This is what's written to the header file `rspnstp.h`.

29.

```
<rspnstp.h 29> ≡  
#ifndef RSPNSTP_H  
#define RSPNSTP_H 1  
    class Scan_Parse_Parameter_Type;  
    class Handle_Type;  
    < class Response_Type declaration 9>  
#endif
```

30. Scan_Parse_Parameter_Type.

format *Scan_Parse_Parameter_Type* *int*

31. Include files.

```
<Include files 3> +≡
#ifndef _GNU_SOURCE
#define _GNU_SOURCE
#endif
#include <stdlib.h>      /* Standard Library for C */
#include <stdio.h>
#include <sys/mman.h>
#include <errno.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <limits.h>
#include <pwd.h>
#include <string.h>
#include <algorithm>      /* Standard Template Library (STL) for C++ */
#include <bitset>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <iostream>
#include <map>
#include <string>
#include <time.h>
#include <math.h>
#include <sstream>
#include <vector>
#include <deque>
#include <stack>
#include <set>
#include <pthread.h>      /* POSIX threads */
#include <gcrypt.h>        /* for gcry_control */
#include <gnutls/gnutls.h>
#include <gnutls/x509.h>
#include <mysql.h>
#include <errmsg.h>
#include <expat.h>
#if HAVE_CONFIG_H
#include <config.h>
#endif
#undef NAME_LEN
#undef LOCAL_HOST
#include "rspercds.h++"
#include "glblcnst.h++"
#include "glblvrbl.h++"
#include "excptntp.h++"
#include "utilfncts.h++"
#include "grouptp.h++"
#include "hndlvltp.h++"
#include "irdsavtp.h++"
#include "helper.h++"
#include "tanfncts.h++"
```

```
#include "pidfncts.h++"
#include "parser.h++"
#include "scanner.h++"
#include "rspnstp.h++"
#include "irdsobtp.h++"
#include "hndltype.h++"
#include "dcmtddtp.h++"
#include "x509cert.h++"
#include "dstngnmt.h++"
#include "usrinftp.h++"
```

32.

⟨ External function declarations 32 ⟩ ≡

```
int yyparse(yyscan_t parameter);
```

See also sections 324, 548, 1541, and 1573.

This code is used in sections 305, 544, 560, 1544, and 1576.

33. Scan_Parse_Parameter_Type. [LDF 2012.06.27.]

Log

- [LDF 2012.07.04.] Added **friend** declaration for *yyparse*. Added **bool** PARSER_DEBUG.
- [LDF 2012.07.05.] Added *fstream response_strm*.
- [LDF 2012.07.11.] Added **string** Distinguished_Name and **string** username. Added **int** user_id. It's initialized to -1 in the constructor and set in **Scan_Parse_Parameter_Type::get_user**. Added **string** irods_password_encrypted, **string** homedir, **string** zone and **string** default_resource. They're set in **Scan_Parse_Parameter_Type::set_user**.
- [LDF 2012.07.13.] Added **int** thread_ctr. It's set to -1 in the constructor.
- [LDF 2012.07.27.] Added the data members **string** data_filename and **char** data_buffer[BUFFER_SIZE].
- [LDF 2012.07.30.] Added the private data member deque < **Response_Type** > response_deque.
- [LDF 2012.07.30.] Removed *fstream response_strm*.
- [LDF 2012.09.12.] Added data member **string** irods_password_encrypted_timestamp.
- [LDF 2012.09.20.] Added data members *gnutls_session_t session* and **bool** remote_connection.
- [LDF 2012.09.21.] Added data member **bool** anonymous. Added **friend** declarations for *listen_local*, *listen_remote_anon* and *listen_remote_X_509*.
- [LDF 2012.09.27.] Added data member **string** icommands_flag_str.
- [LDF 2012.09.27.] Added the private data member **vector<string>** filename_vector.
- [LDF 2012.09.28.] Added the private data members **string** pid_str, **string** pid_suffix and **string** pid_institute_str.
- [LDF 2012.10.08.] Added the private data members **string** pid_prefix_str, **unsigned int** default_handle_prefix_id and **string** default_handle_prefix.
- [LDF 2012.10.08.] Added the private data members **unsigned int** default_institute_id and **string** default_institute_name.
- [LDF 2012.10.16.] Added **Handle_Value_Type** handle. It's used in *zzparse*.
- [LDF 2012.10.17.] Changed the names of the **string** variables homedir, zone, default_resource and current_dir to irods_homedir, irods_zone, irods_default_resource and irods_current_dir, respectively.
- [LDF 2012.11.22.] Added **map<unsigned int, Response_Type>** response_map.
- [LDF 2012.11.22.] Added **pthread_mutex_t** response_map_mutex.
- [LDF 2013.03.22.] Added **string** irods_auth_filename and **string** irods_env_filename.
- [LDF 2013.04.03.] Added the private data members deque < **Response_Type** > pending_response_deque, **bool** pending_operations_flag and deque < **Response_Type** > ::iterator pending_operations_iter.
- [LDF 2013.04.25.] Removed data member **string** icommands_flag_str.
- [LDF 2013.05.10.] Removed data member **string** Distinguished_Name.
- [LDF 2013.05.17.] Added the data members *User_Info_Type * user_info_ptr*.
- [LDF 2013.05.17.] Added data member *Distinguished_Name_Type distinguished_name*.
- [LDF 2013.05.19.] Added member function *set_user_info*.
- [LDF 2013.05.22.] Added **friend** declarations for *distinguished_name_rule_func* and *get_user_info_func*.
- [LDF 2013.05.22.] Added data members **map<string, int>** user_id_map and **map<int, User_Info_Type>** user_info_map.
- [LDF 2013.05.24.] Renamed deque < **Response_Type** > pending_response_deque to delayed_response_deque.
- [LDF 2013.06.16.] Added **Handle_Value_Triple hvt**. It's used in *yyparse*.
- [LDF 2013.07.11.] Added **string** input_commands. It's used by **gwirdcli** only and set by the command-line option --commands.
- [LDF 2013.08.07.] Added **friend** declaration for *yylex*.
- [LDF 2013.08.09.] Added **unsigned long** delay_value. It's used in *yyparse*.
- [LDF 2013.09.04.] Added **friend** declaration for *create_databases*.
- [LDF 2013.09.20.] Added **string** public_key_id.

```
< class Scan_Parse_Parameter_Type declaration 33 > ≡
class Scan_Parse_Parameter_Type { friend int create_databases(Scan_Parse_Parameter_Type
    &param);
friend int Irods_Object_Type::get_avus_from_irods_system(string command, string
    filename, Scan_Parse_Parameter_Type &param, int *ctr);
```

```

friend int main(int, char **);
friend void *listen_local(void *v);
friend void *listen_remote_anon(void *v);
friend void *listen_remote_X_509(void *v);
friend void *connect_func(void *v);
friend int yyparse(yyscan_t parameter); friend int yylex ( YYSTYPE * , yyscan_t ) ;
friend int client_func(Scan_Parse_Parameter_Type &param);
friend int parse_post_data(Scan_Parse_Parameter_Type &param);
friend int exchange_data_with_server(Scan_Parse_Parameter_Type &param);
friend int exchange_data_with_client(Scan_Parse_Parameter_Type &param);
friend int client_sending_file_rule_func(Scan_Parse_Parameter_Type *param, string filename, int
reference);
friend int distinguished_name_rule_func(Scan_Parse_Parameter_Type *, const char *);
friend int get_user_info_func(Scan_Parse_Parameter_Type *, const char *);
int sock;
gnutls_session_t session;
bool remote_connection;
bool anonymous;
unsigned int connection_type;
bool PARSER_DEBUG;
MYSQL *mysql_ptr;
Distinguished_Name_Type distinguished_name;
int user_id;
string username;
vector<Group_Type> group_vector;
unsigned int privileges;
/* On pcfinston.gwdg.de, this allows the definition of 32 privileges. [LDF 2013.05.17.] */
string irods_auth_filename;
string irods_env_filename;
string irods_password_encrypted;
string irods_password_encrypted_timestamp;
string irods_homedir;
string irods_current_dir;
string irods_zone;
string irods_default_resource;
int thread_ctr;
time_t expires;
string data_filename;
char data_buffer[BUFFER_SIZE];
string input_commands; deque < Response_Type > response_deque; deque <
Response_Type > delayed_response_deque;
bool pending_operations_flag; deque < Response_Type > ::iterator pending_operations_iter;
map<unsigned int, Response_Type> response_map;
pthread_mutex_t response_map_mutex;
bool client_finished;
bool server_finished;
vector<string> filename_vector; /* These are set by Scan_Parse_Parameter_Type::get_user
(server-side only). [LDF 2012.10.08.] */
unsigned int default_handle_prefix_id;
string default_handle_prefix;
unsigned int default_institute_id;

```

```
string default_institute_name; /* These are used in the parser. [LDF 2012.10.08.] */
string pid_str;
string pid_prefix_str;
string pid_suffix_str;
string pid_institute_str;
unsigned long delay_value;
Handle_Value_Triple hvt;
map<string, int> user_id_map;
map<int, User_Info_Type> user_info_map;
Handle_Value_Type handle_value;
User_Info_Type * user_info_ptr; /* Used in zzparse [LDF 2013.05.17.] */
```

See also sections 34 and 35.

This code is used in sections 305 and 306.

34. Irods_Object_Type **irods_object* is used in the client-side parser *zzparse* for collecting data sent by the server. Currently, these are just the user-defined metadata (a.k.a. AVUs) and the filename. So far, I've just been using simple filenames, i.e., not the complete path. The object is created (with **new**) in the rule “⟨avu list⟩ → Empty” and deleted and set to 0 in the rule “⟨statement⟩ → GET_ZZ METADATA_ZZ RESPONSE_ZZ STRING_ZZ INTEGER_ZZ INTEGER_ZZ ⟨avu list⟩”.

The objects pointed to by *irods_object* are pushed onto **map**⟨**Irods_Object_Type**⟩ *irods_object_vector*, so the latter contains a collection of **Irods_Object_Type** objects that have been retrieved during a run of the program.

!! TODO: It would be possible to store them in a database. In this case, I should use an expiration time, as the Handle System does for handles.

!! TODO: Maybe use a **map** with the PID as the key. [LDF 2012.10.12.]

Log

[LDF 2012.11.16.] Added **vector**⟨**string**⟩ *temp_file_vector*. It replaces the global variable of the same type and name. The temporary files will be deleted in the destructor for **Scan_Parse_Parameter_Type**. This ensures that temporary files that are still needed in a running thread won't be deleted.

[LDF 2012.11.19.] Added **unsigned int** *errors_occurred* and **unsigned int** *warnings_occurred*.

[LDF 2013.01.07.] Working on **Irods_Object_Type**. Added data members. Added corresponding database table *gwirdsif.Irods_Objects*. See *irdsobtp.web* and *archobjs.sql*.

[LDF 2013.04.25.] Added **int** *thread_cancel_state*. It is initialized to PTHREAD_CANCEL_ENABLE in the default constructor (as of this date, the only constructor).

[LDF 2013.04.25.] Added **vector**⟨**string**⟩ *string_vector*. It's used in *yyparse*.

[LDF 2013.05.08.] Added *X509_Cert_Type* *user_cert* and *X509_Cert_Type* *ca_cert*.

[LDF 2013.05.27.]

[LDF 2013.05.27.] Added the **static** maps *server_action_map*, *client_action_map*, *server_action_name_map* and *client_action_name_map*.

```
( class Scan_Parse_Parameter_Type declaration 33 ) +≡
  Irods_Object_Type *irods_object;
  vector<Irods_Object_Type> irods_object_vector;
  vector<string> temp_file_vector;
  vector<string> string_vector;
  unsigned int errors_occurred;
  unsigned int warnings_occurred;
  typedef int(Scan_Parse_Parameter_Type ::*func_ptr)(Response_Type &);
  static map<unsigned int,func_ptr> server_action_map;
  static map<unsigned int,func_ptr> client_action_map;
  static map<unsigned int,string> server_action_name_map;
  static map<unsigned int,string> client_action_name_map;
  int thread_cancel_state;
  X509_Cert_Type user_cert;      /* Used by server */
  X509_Cert_Type server_cert;    /* Used by client */
  X509_Cert_Type ca_cert;       /* Currently not used. [LDF 2013.05.10.] */
  X509_Cert_Type *cert_ptr;     /* Used in zzparse. [LDF 2013.05.15.] */
  string public_key_id;
```

35.

Log

[LDF 2013.05.08.] Added definition of `get_thread_ctr`.

[LDF 2013.05.10.] Added the following **static** constants: `NULL_AUTH_TYPE`, `LOCAL_NULL_AUTH_TYPE`, `X_509_AUTH_TYPE` and `ANON_AUTH_TYPE`.

[LDF 2013.05.16.] Added the following **static** constants: `SUPERUSER_PRIVILEGE`, `DELEGATE_PRIVILEGE`, `SHOW_CERTIFICATES_PRIVILEGE`, `SHOW_DISTINGUISHED_NAMES_PRIVILEGE` and `SHOW_PRIVILEGES_PRIVILEGE`. ■

[LDF 2013.05.17.] Added **static unsigned int** `SHOW_USER_INFO_PRIVILEGE`.

[LDF 2013.06.05.] Added **static unsigned int** `SHOW_GROUPS_PRIVILEGE`.

[LDF 2013.07.04.] Added **static unsigned int** `DELETE_HANDLES_PRIVILEGE`.

[LDF 2013.09.12.] Added the **static unsigned int** constants `DELETE_HANDLE_VALUES_PRIVILEGE`, `DELETE_HS_ADMIN_HANDLE_VALUES_PRIVILEGE` and `UNDELETE_HANDLE_VALUES_PRIVILEGE`. Made the other **static unsigned int** variables for privileges and authorization type constants, too.

```

< class Scan_Parse_Parameter_Type declaration 33 > +≡
public: < Scan_Parse_Parameter_Type function declarations 38 >
    int get_pids(string s, int *ctr = 0)
    {
        return get_metadata(s, 1, ctr);
    }
    int get_thread_ctr(void)
    {
        return thread_ctr;
    }
    static const unsigned int NULL_AUTH_TYPE;
    static const unsigned int LOCAL_NULL_AUTH_TYPE;
    static const unsigned int X_509_AUTH_TYPE;
    static const unsigned int ANON_AUTH_TYPE; /* On pcfinston.gwdg.de, 32 privileges are allowed
        (sizeof(unsigned int) * 8). [LDF 2013.05.17.] */
    static const unsigned int SUPERUSER_PRIVILEGE;
    static const unsigned int DELEGATE_PRIVILEGE;
    static const unsigned int DELETE_HANDLES_PRIVILEGE;
    static const unsigned int DELETE_HANDLE_VALUES_PRIVILEGE;
    static const unsigned int DELETE_HS_ADMIN_HANDLE_VALUES_PRIVILEGE;
    static const unsigned int DELETE_LAST_HS_ADMIN_HANDLE_VALUE_PRIVILEGE;
    static const unsigned int UNDELETE_HANDLE_VALUES_PRIVILEGE;
    static const unsigned int SHOW_USER_INFO_PRIVILEGE;
    static const unsigned int SHOW_GROUPS_PRIVILEGE;
    static const unsigned int SHOW_CERTIFICATES_PRIVILEGE;
    static const unsigned int SHOW_DISTINGUISHED_NAMES_PRIVILEGE;
    static const unsigned int SHOW_PRIVILEGES_PRIVILEGE; } ;

```

36. Initialize **Scan_Parse_Parameter_Type** static data members. [LDF 2013.05.10.]

Log

[LDF 2013.05.10.] Added this section.

```
( Initialize Scan_Parse_Parameter_Type static data members 36 ) ≡
const unsigned int Scan_Parse_Parameter_Type::NULL_AUTH_TYPE = 0;
const unsigned int Scan_Parse_Parameter_Type::LOCAL_NULL_AUTH_TYPE = 1;
const unsigned int Scan_Parse_Parameter_Type::X_509_AUTH_TYPE = 2;
const unsigned int Scan_Parse_Parameter_Type::ANON_AUTH_TYPE = 3;
    /* On pcfinston.gwdg.de, 32 privileges are allowed (sizeof(unsigned int)*8). [LDF 2013.05.17.] */
const unsigned int Scan_Parse_Parameter_Type::SUPERUSER_PRIVILEGE = 1U;
const unsigned int Scan_Parse_Parameter_Type::DELEGATE_PRIVILEGE = 2U;
const unsigned int Scan_Parse_Parameter_Type::DELETE_HANDLES_PRIVILEGE = 4U;
const unsigned int Scan_Parse_Parameter_Type::DELETE_HANDLE_VALUES_PRIVILEGE = 8U;
const unsigned int Scan_Parse_Parameter_Type::DELETE_HS_ADMIN_HANDLE_VALUES_PRIVILEGE =
    16U;
const unsigned int
    Scan_Parse_Parameter_Type::DELETE_LAST_HS_ADMIN_HANDLE_VALUE_PRIVILEGE = 32U;
const unsigned int Scan_Parse_Parameter_Type::UNDELETE_HANDLE_VALUES_PRIVILEGE = 64U;
const unsigned int Scan_Parse_Parameter_Type::SHOW_USER_INFO_PRIVILEGE = 128U;
const unsigned int Scan_Parse_Parameter_Type::SHOW_GROUPS_PRIVILEGE = 256U;
const unsigned int Scan_Parse_Parameter_Type::SHOW_CERTIFICATES_PRIVILEGE = 512U;
const unsigned int Scan_Parse_Parameter_Type::SHOW_DISTINGUISHED_NAMES_PRIVILEGE = 1024U;
const unsigned int Scan_Parse_Parameter_Type::SHOW_PRIVILEGES_PRIVILEGE = 2048U;
typedef int(Scan_Parse_Parameter_Type::*func_ptr)(Response_Type &);
map<unsigned int, func_ptr> Scan_Parse_Parameter_Type::server_action_map;
map<unsigned int, func_ptr> Scan_Parse_Parameter_Type::client_action_map;
map<unsigned int, string> Scan_Parse_Parameter_Type::server_action_name_map;
map<unsigned int, string> Scan_Parse_Parameter_Type::client_action_name_map;
```

This code is used in section 305.

37. Scanner_Type functions. [LDF 2012.06.27.]

38. Default constructor. [LDF 2012.06.27.]

Log

[LDF 2012.06.27.] Added this function.

[LDF 2012.07.10.] Added code for instantiating an *IrodsAccess* object.

[LDF 2012.11.19.] Now initializing **unsigned int errors_occurred** and **unsigned int warnings_occurred** to 0.

[LDF 2013.04.25.] Now initializing *thread_cancel_state* to PTHREAD_CANCEL_ENABLE.

[LDF 2013.08.09.] BUG FIX: Added code for calling *mysql_real_connect* and *mysql_select_db* in loops: If either call fails, it's repeated up to 5 times. If the sixth attempt fails, the function calls *pthread_exit*.

(**Scan_Parse_Parameter_Type** function declarations 38) ≡

Scan_Parse_Parameter_Type(void);

See also sections 50, 56, 61, 69, 103, 104, 105, 107, 109, 114, 130, 132, 135, 144, 150, 200, 217, 232, 234, 253, 254, 255, 273, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, and 302.

This code is used in section 35.

39.

```
⟨ Scan_Parse_Parameter_Type constructor definitions 39 ⟩ ≡
Scan_Parse_Parameter_Type::Scan_Parse_Parameter_Type(void){ bool DEBUG = false;
    /* true */
    set_debug_level(DEBUG, 0, 0);
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Entering 'Scan_Parse_Parameter_Type' default constructor." << endl;
        unlock_cerr_mutex();
    }      /* if (DEBUG) */
    int status;
    thread_ctr = 0;
    user_id = -1;
    privileges = 0U;
    expires = 0;
    sock = 0;
    PARSER_DEBUG = false;
    client_finished = false;
    server_finished = false;
    remote_connection = false;
    anonymous = false;
    connection_type = 0U;
    default_handle_prefix_id = 0;
    default_institute_id = 0;
    delay_value = 0UL;
    cert_ptr = 0;
    user_info_ptr = 0;
    memset(data_buffer, 0, BUFFER_SIZE);
    irods_object = 0;
    errors_occurred = 0;
    warnings_occurred = 0;
    thread_cancel_state = PTHREAD_CANCEL_ENABLE;
    bool failed = false;
```

See also sections 40, 41, 42, 43, 44, 45, 46, 47, 48, and 49.

This code is used in section 305.

40.

Log

[LDF 2012.07.10.] Added this section.

[LDF 2012.09.18.] Now only calling *setup-java-vm* and the MySQL functions if *is_gwirdsif* \equiv true. Neither the Java VM nor the MySQL database are needed at this time for *gwirdcli* or *gurdwbp*. However, it's not possible to compile the conditionally, because the same object file is used for all three programs. I prefer it this way and don't want to change it. I don't see any reason to supply a "client-only" version of the package, especially considering that the necessary libraries are easily available.

[LDF 2013.04.26.] Now connecting to the database if \neg *is_gwirdsif*, i.e., also if this function is called on the client-side. Now using the *gwirdcli* database on the client-side. In this case, *getpwuid_r* is called to retrieve the user name, which is used as the database user name, too.

```
( Scan_Parse_Parameter_Type constructor definitions 39 ) +≡
mysql_ptr = mysql_init(0);
if (mysql_ptr) {
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'Scan_Parse_Parameter_Type' constructor:" << "'mysql_init' succeeded." <<
            endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
} /* if (mysql_ptr) */
else {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Scan_Parse_Parameter_Type' constructor:" << endl <<
        "'mysql_init' failed. Exiting thread unsuccessfully with" << "return value 0." <<
        endl;
    unlock_cerr_mutex();
    pthread_exit(0);
}
```

41. Set *mysql_socket_filename*. [LDF 2012.09.07.]

Log

[LDF 2012.09.07.] Added this section. It's needed, because the name of the socket file for the MySQL server is platform-dependent.

```

⟨ Scan_Parse_Parameter_Type constructor definitions 39 ⟩ +≡
    string mysql_socket_filename;
    errno = 0;
    status = access("/var/run/mysql/mysql.sock", F_OK);
    if (status == -1) {
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "In 'Scan_Parse_Parameter_Type'::constructor:" << endl <<
                "MySQL_server_socket_file '/var/run/mysql/mysql.sock' doesn't exist:" <<
                endl << "'access' error:" << strerror(errno) << endl <<
                "Trying '/var/run/mysqld/mysqld.sock'." << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
        errno = 0;
        status = access("/var/run/mysqld/mysqld.sock", F_OK);
        if (status == -1) {
            lock_cerr_mutex();
            cerr << "ERROR! In 'Scan_Parse_Parameter_Type'::constructor:" << endl <<
                "MySQL_server_socket_file '/var/run/mysqld/mysqld.sock' doesn't exist:" <<
                endl << "'access' error:" << strerror(errno) << endl <<
                "Tested for '/var/run/mysql/mysql.sock' previously." << endl <<
                "Exiting thread unsuccessfully with " << "return_value 0." << endl;
            unlock_cerr_mutex();
            mysql_close(mysql_ptr);
            mysql_ptr = 0;
            pthread_exit(0);
        } /* if (status == -1) */
        else {
#ifndef DEBUG_COMPILE
            if (DEBUG) {
                lock_cerr_mutex();
                cerr << "In 'Scan_Parse_Parameter_Type'::constructor:" << endl <<
                    "MySQL_server_socket_file '/var/run/mysqld/mysqld.sock' exists." << endl;
                unlock_cerr_mutex();
            } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
            mysql_socket_filename = "/var/run/mysqld/mysqld.sock";
        } /* else */
    } /* if (status == -1) */
    else {
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();

```

```

    cerr << "In 'Scan_Parse_Parameter_Type' constructor:" << endl <<
        "MySQL_server_socket_file '/var/run/mysql/mysql.sock' exists." << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
mysql_socket_filename = "/var/run/mysql/mysql.sock";
} /* else */

```

42.

(Scan_Parse_Parameter_Type constructor definitions 39) +≡
string temp_str;
stringstream temp_strm;

43.

(Scan_Parse_Parameter_Type constructor definitions 39) +≡

44.

(Scan_Parse_Parameter_Type constructor definitions 39) +≡
my_bool reconnect = 1;
unsigned int timeout = 120;
mysql_options(mysql_ptr, MYSQL_OPT_RECONNECT, &reconnect);
mysql_options(mysql_ptr, MYSQL_OPT_CONNECT_TIMEOUT, &timeout);
#if 0
mysql_options(mysql_ptr, MYSQL_INIT_COMMAND, "set time_zone='+0:00');
#endif
failed = false;

45.

```
<Scan_Parse_Parameter_Type constructor definitions 39> +≡
  if (is_gwirdsif) {
    if (mysql_username.empty()) {
      mysql_username = "root";
    }
    if (mysql_password == 0) {
#if DEBUG_COMPILE
      if (DEBUG) {
        lock_cerr_mutex();
        cerr << "'mysql_password' == 0." << endl << "Allocating memory and setting it here." <<
          endl;
        unlock_cerr_mutex();
      } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
      mysql_password = new char[MYSQL_PASSWORD_LENGTH];
      memset(mysql_password, 0, MYSQL_PASSWORD_LENGTH);
      strcpy(mysql_password, "root");
    } /* if (mysql_password == 0) */
#endif /* DEBUG_COMPILE */
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "'mysql_password' != 0. 'mysql_password' == " << mysql_password << endl <<
        "Not allocating memory and setting it here (already set)." << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (is_gwirdsif) */
```

46.

```

⟨ Scan_Parse_Parameter_Type constructor definitions 39 ⟩ +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'Scan_Parse_Parameter_Type'::default_constructor:" << endl <<
            "mysql_username==" << mysql_username << endl;
        if (mysql_password != 0) cerr << "mysql_password==" << mysql_password << endl;
        else cerr << "mysql_password==NULL" << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    for (int i = 0; i < 6; ++i) {
        if (!mysql_real_connect(mysql_ptr, 0, mysql_username.c_str(), mysql_password, 0, 0,
                               mysql_socket_filename.c_str(), 0)) {
            failed = true;
            lock_cerr_mutex();
            cerr << "ERROR! In 'Scan_Parse_Parameter_Type'::constructor:" << endl <<
                "'mysql_real_connect' failed." << "Error:" << mysql_error(mysql_ptr) << endl <<
                "Error_number:" << mysql_errno(mysql_ptr) << endl;
            unlock_cerr_mutex();
        if (i == 5) {
            lock_cerr_mutex();
            cerr << "Exiting thread unsuccessfully with " << "return_value 0." << endl;
            unlock_cerr_mutex();
            mysql_close(mysql_ptr);
            mysql_ptr = 0;
            pthread_exit(0);
        } /* if */
        else {
            lock_cerr_mutex();
            cerr << "Trying again, i.e., calling 'mysql_real_connect' again." << endl;
            unlock_cerr_mutex();
            sleep(1);
            continue;
        }
    } /* if */
else {
    if (failed || DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'Scan_Parse_Parameter_Type'::constructor:" << endl <<
            "'mysql_real_connect' succeeded. Database user name==" << mysql_username <<
            "." << endl;
        unlock_cerr_mutex();
    }
    failed = false;
    break;
} /* else */
} /* for */

```

47. Select database. [LDF 2012.07.10.]

```
<Scan_Parse_Parameter_Type constructor definitions 39> +≡
if (is_gwirdsif) temp_str = "gwirdsif";
else temp_str = "gwirdcli";
failed = false;
for (int i = 0; i < 6; ++i) {
    status = mysql_select_db(mysql_ptr, temp_str.c_str());
    if (status ≡ 0) {
        if (failed ∨ DEBUG) {
            lock_cerr_mutex();
            cerr ≪ "In 'Scan_Parse_Parameter_Type' constructor:" ≪
                "'mysql_select_db_succeeded'." ≪ endl ≪ "Connected to "
                temp_str ≪ "' database successfully." ≪ endl;
            unlock_cerr_mutex();
        }
        failed = false;
        break;
    } /* if (status ≡ 0) */
else /* status ≠ 0 */
{
    failed = true;
    lock_cerr_mutex();
    cerr ≪ "In 'Scan_Parse_Parameter_Type' constructor:" ≪
        "'mysql_select_db' failed, returning " ≪ status ≪ "."
        ≪ endl ≪ "Failed to connect to "
        temp_str ≪ "' database." ≪ endl;
    unlock_cerr_mutex();
    if (i ≡ 5) {
        lock_cerr_mutex();
        cerr ≪ "Exiting thread unsuccessfully with exit status 0." ≪ endl;
        unlock_cerr_mutex();
        mysql_close(mysql_ptr);
        mysql_ptr = 0;
        pthread_exit(0);
    }
    else {
        lock_cerr_mutex();
        cerr ≪ "Trying again, i.e., calling 'mysql_select_db' again." ≪ endl;
        unlock_cerr_mutex();
        sleep(1);
        continue;
    }
} /* else (status ≠ 0) */
} /* for */
```

48.

<Scan_Parse_Parameter_Type constructor definitions 39> +≡

49.

```
⟨ Scan_Parse_Parameter_Type constructor definitions 39 ⟩ +≡
    pthread_mutex_init(&response_map_mutex, 0);
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Exiting 'Scan_Parse_Parameter_Type' default constructor successfully"
            << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
return; }

/* End of Scan_Parse_Parameter_Type::Scan_Parse_Parameter_Type(void) definition. */
```

50. Destructor. [LDF 2012.06.27.]

Log

[LDF 2012.06.27.] Added this function.
 [LDF 2012.09.18.] Now only calling `mysql_close(mysql_ptr)`, `destroy_java_vm` and `rcDisconnect(conn)` if `is_gwirdsif` ≡ true.
 [LDF 2012.11.16.] Added code for deleting the temporary files whose names are stored on `vector<string> temp_file_vector`.
 [LDF 2012.11.27.] Added code for deleting files named `.irodsEnv` with numerical suffixes created in `$HOME/.irods` by the icommand `icd`.

```
⟨ Scan_Parse_Parameter_Type function declarations 38 ⟩ +≡
~Scan_Parse_Parameter_Type(void);
```

51.

```

⟨ Scan_Parse_Parameter_Type destructor definition 51 ⟩ ≡
Scan_Parse_Parameter_Type::~Scan_Parse_Parameter_Type(void){ bool DEBUG = false;
    /* true */
    set_debug_level(DEBUG, 0, 0);
    int status = 0;
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "]Entering 'Scan_Parse_Parameter_Type' \\" \
            "destructor." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
    response_deque.clear();
    delayed_response_deque.clear();
    group_vector.clear();
    if (remote_connection == true) {
        gnutls_bye(session, GNUTLS_SHUT_RDWR);
        tcp_close(sock);
        gnutls_deinit(session);
    }
    else if (sock > 0) {
        close(sock);
    }
    if (is_gwirdsif) {
        if (mysql_ptr) mysql_close(mysql_ptr);
        mysql_ptr = 0;
    } /* if (is_gwirdsif) */
    if (irods_object) {
        delete irods_object;
        irods_object = 0;
    }
    if (cert_ptr) {
        delete cert_ptr;
        cert_ptr = 0;
    }
    if (user_info_ptr) {
        delete user_info_ptr;
        user_info_ptr = 0;
    }
}

```

See also sections 52, 53, 54, and 55.

This code is used in section 305.

52.

Log

[LDF 2012.11.16.] Added this section.

```

⟨ Scan_Parse_Parameter_Type destructor definition 51 ⟩ +=

#ifndef DEBUG_COMPILE
    if (temp_file_vector.size() == 0 & DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] In 'Scan_Parse_Parameter_Type' destructor:" << endl <<
            "'temp_file_vector.size()=='0'. No temporary files to delete." << endl;
        unlock_cerr_mutex();
    } /* if (temp_file_vector.size() == 0 & DEBUG) */
#endif /* DEBUG_COMPILE */
    else
        if (temp_file_vector.size() > 0 & save_temp_files == false) {
#ifndef DEBUG_COMPILE
            if (DEBUG) {
                lock_cerr_mutex();
                cerr << "[Thread" << thread_ctr << "] In 'Scan_Parse_Parameter_Type' destructor:" <<
                    endl << "'temp_file_vector.size()' > 0 and 'save_temp_files' == 'false'." <<
                    endl << "Deleting temporary files." << endl;
                unlock_cerr_mutex();
            } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
            for (vector<string>::const_iterator iter = temp_file_vector.begin(); iter != temp_file_vector.end();
                ++iter) {
                errno = 0;
                status = unlink(iter->c_str());
                if (status == -1) {
                    lock_cerr_mutex();
                    cerr << "[Thread" << thread_ctr << "] ERROR!" <<
                        " In 'Scan_Parse_Parameter_Type' destructor:" << endl <<
                        "'unlink' failed, returning -1:" << endl << strerror(errno) <<
                        endl << "Failed to delete file " << *iter << "." << endl <<
                        "Will try to continue." << endl;
                    unlock_cerr_mutex();
                } /* if (status == -1) */
#endif /* DEBUG_COMPILE */
                else
                    if (DEBUG) {
                        lock_cerr_mutex();
                        cerr << "[Thread" << thread_ctr << "] "
                            " In 'Scan_Parse_Parameter_Type' destructor:" << endl <<
                            "'unlink' succeeded, returning 0." << endl << "Deleted file " << *iter <<
                            "' successfully." << endl;
                        unlock_cerr_mutex();
                    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
                } /* for */
                temp_file_vector.clear();
} /* else if (temp_file_vector.size() > 0 & save_temp_files == false) */

```

```
#if DEBUG_COMPILE
else
    if (save_temp_files == true & DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread]" << thread_ctr << "] In 'Scan_Parse_Parameter_Type' destructor:" <<
            endl << "'save_temp_files'" == 'true' . Not deleting temporary files." << endl;
        unlock_cerr_mutex();
    } /* else if (save_temp_files == true & DEBUG) */
#endif /* DEBUG_COMPILE */
```

53.

```
< Scan_Parse_Parameter_Type destructor definition 51 > +=  
filename_vector.clear();  
string_vector.clear();  
pthread_mutex_destroy(&response_map_mutex);
```

54. Delete iRODS environment files created by `icd` command. [LDF 2012.11.27.]

Log

[LDF 2012.11.27.] Added this section.

```
<Scan_Parse_Parameter_Type destructor definition 51> +≡
if (is_gwirdsif) {
    stringstream temp_strm;
    temp_strm.str("");
    temp_strm << "find $HOME/.irods/_depth -regex $HOME/.irodsEnv\\.[0-9]+_delete";
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "temp_strm.str() == " << temp_strm.str() << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
status = system(temp_strm.str().c_str());
if (status != 0) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] In 'Scan_Parse_Parameter_Type' \n"
        << "destructor:" << "'system' failed, returning" << status << ":" << endl;
    if (WIFEXITED(status)) {
        cerr << "WEXITSTATUS(" << status << ") == " << WEXITSTATUS(status) << endl;
    }
    else {
        cerr << "Process didn't exit." << endl;
    }
    unlock_cerr_mutex();
} /* if (status != 0) */
else if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] In 'Scan_Parse_Parameter_Type' destructor:" <<
        "'system' succeeded." << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
/* if (is_gwirdsif) */
```

55.

```
<Scan_Parse_Parameter_Type destructor definition 51> +≡
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] Exiting 'Scan_Parse_Parameter_Type' \n"
        << "destructor successfully" << "with void return value." << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
return; } /* End of Scan_Parse_Parameter_Type::~Scan_Parse_Parameter_Type(void) definition. */
```

56. Initialize maps. [LDF 2013.05.27.]

Log

[LDF 2013.05.27.] Added this function.

{ **Scan_Parse_Parameter_Type** function declarations 38 } +≡

static int *initialize_maps(void);*

57.

```

⟨ Initialize Scan_Parse_Parameter_Type maps 57 ⟩ ≡
int Scan_Parse_Parameter_Type::initialize_maps(void){
    server_action_map[Response_Type::COMMAND_ONLY_TYPE] =
        &Scan_Parse_Parameter_Type::server_action_command_only;
    server_action_map[Response_Type::SEND_FILE_TYPE] =
        &Scan_Parse_Parameter_Type::server_action_send_file;
    server_action_map[Response_Type::RECEIVE_PUT_FILE_TYPE] =
        &Scan_Parse_Parameter_Type::server_action_receive_put_file;
    server_action_map[Response_Type::RECEIVE_METADATA_FILE_TYPE] =
        &Scan_Parse_Parameter_Type::server_action_receive_metadata_file;
    server_action_map[Response_Type::SEND_HANDLE_TYPE] =
        &Scan_Parse_Parameter_Type::server_action_send_handle;
    server_action_map[Response_Type::LS_TYPE] =
        &Scan_Parse_Parameter_Type::server_action_ls;
    server_action_map[Response_Type::PWD_TYPE] =
        &Scan_Parse_Parameter_Type::server_action_pwd;
    server_action_map[Response_Type::CD_TYPE] =
        &Scan_Parse_Parameter_Type::server_action_cd;
    server_action_map[Response_Type::MKDIR_TYPE] =
        &Scan_Parse_Parameter_Type::server_action_mkdir;
    server_action_map[Response_Type::MARK_IRODS_OBJECTS_FOR_DELETION_TYPE] =
        &Scan_Parse_Parameter_Type::server_action_mark_irods_objects_for_deletion;
    server_action_map[Response_Type::GET_TYPE] =
        &Scan_Parse_Parameter_Type::server_action_get;
    server_action_map[Response_Type::SEND_METADATA_TYPE] =
        &Scan_Parse_Parameter_Type::server_action_send_metadata;
    server_action_map[Response_Type::END_SERVER_TYPE] =
        &Scan_Parse_Parameter_Type::server_action_end_server;
    server_action_map[Response_Type::SLEEP_TYPE] =
        &Scan_Parse_Parameter_Type::server_action_sleep;
    server_action_map[Response_Type::SHOW_CERTIFICATE_TYPE] =
        &Scan_Parse_Parameter_Type::server_action_show_certificate;
    server_action_map[Response_Type::GET_METADATA_TYPE] =
        &Scan_Parse_Parameter_Type::server_action_get_metadata;
    server_action_map[Response_Type::GET_HANDLE_TYPE] =
        &Scan_Parse_Parameter_Type::server_action_get_handle;
    server_action_map[Response_Type::SEND_TAN_LIST_TYPE] =
        &Scan_Parse_Parameter_Type::server_action_send_tan_list;
    server_action_map[Response_Type::PROCESS_PENDING_TYPE] =
        &Scan_Parse_Parameter_Type::server_action_process_pending;
    server_action_map[Response_Type::GET_USER_INFO_TYPE] =
        &Scan_Parse_Parameter_Type::server_action_get_user_info;
    server_action_map[Response_Type::CREATE_HANDLE_TYPE] =
        &Scan_Parse_Parameter_Type::server_action_create_handle;
    server_action_map[Response_Type::ADD_HANDLE_VALUE_TYPE] =
        &Scan_Parse_Parameter_Type::server_action_add_handle_value;
    server_action_map[Response_Type::DELETE_HANDLE_TYPE] =
        &Scan_Parse_Parameter_Type::server_action_delete_handle;
    server_action_map[Response_Type::UNDELETE_HANDLE_TYPE] =
        &Scan_Parse_Parameter_Type::server_action_undelete_handle;
}

```

```
server_action_map[Response_Type::DELETE_HANDLE_VALUE_TYPE] =
    &Scan_Parse_Parameter_Type::server_action_delete_handle_value;
server_action_map[Response_Type::UNDELETE_HANDLE_VALUE_TYPE] =
    &Scan_Parse_Parameter_Type::server_action_undelete_handle_value;
server_action_map[Response_Type::UNDELETE_FILE_TYPE] =
    &Scan_Parse_Parameter_Type::server_action_undelete_file;
server_action_map[Response_Type::MAX_RESPONSE_TYPE + 1] =
    &Scan_Parse_Parameter_Type::server_action_unknown;
```

See also sections 58, 59, and 60.

This code is used in section 305.

58.

```

⟨ Initialize Scan_Parse_Parameter_Type maps 57 ⟩ +≡
server_action_name_map[Response_Type::COMMAND_ONLY_TYPE] =
    "Scan_Parse_Parameter_Type::server_action_command_only";
server_action_name_map[Response_Type::SEND_FILE_TYPE] =
    "Scan_Parse_Parameter_Type::server_action_send_file";
server_action_name_map[Response_Type::RECEIVE_PUT_FILE_TYPE] =
    "Scan_Parse_Parameter_Type::server_action_receive_put_file";
server_action_name_map[Response_Type::RECEIVE_METADATA_FILE_TYPE] =
    "Scan_Parse_Parameter_Type::server_action_receive_metadata_file";
server_action_name_map[Response_Type::SEND_HANDLE_TYPE] =
    "Scan_Parse_Parameter_Type::server_action_send_handle";
server_action_name_map[Response_Type::LS_TYPE] = "Scan_Parse_Parameter_Type::server_actio\
n_ls";
server_action_name_map[Response_Type::PWD_TYPE] =
    "Scan_Parse_Parameter_Type::server_action_pwd";
server_action_name_map[Response_Type::CD_TYPE] = "Scan_Parse_Parameter_Type::server_actio\
n_cd";
server_action_name_map[Response_Type::MKDIR_TYPE] =
    "Scan_Parse_Parameter_Type::server_action_mkdir";
server_action_name_map[Response_Type::MARK_IRODS_OBJECTS_FOR_DELETION_TYPE] =
    "Scan_Parse_Parameter_Type::server_action_mark_irods_objects_for_deletion";
server_action_name_map[Response_Type::GET_TYPE] =
    "Scan_Parse_Parameter_Type::server_action_get";
server_action_name_map[Response_Type::SEND_METADATA_TYPE] =
    "Scan_Parse_Parameter_Type::server_action_send_metadata";
server_action_name_map[Response_Type::END_SERVER_TYPE] =
    "Scan_Parse_Parameter_Type::server_action_end_server";
server_action_name_map[Response_Type::SLEEP_TYPE] =
    "Scan_Parse_Parameter_Type::server_action_sleep";
server_action_name_map[Response_Type::SHOW_CERTIFICATE_TYPE] =
    "Scan_Parse_Parameter_Type::server_action_show_certificate";
server_action_name_map[Response_Type::GET_METADATA_TYPE] =
    "Scan_Parse_Parameter_Type::server_action_get_metadata";
server_action_name_map[Response_Type::GET_HANDLE_TYPE] =
    "Scan_Parse_Parameter_Type::server_action_get_handle";
server_action_name_map[Response_Type::SEND_TAN_LIST_TYPE] =
    "Scan_Parse_Parameter_Type::server_action_send_tan_list";
server_action_name_map[Response_Type::PROCESS_PENDING_TYPE] =
    "Scan_Parse_Parameter_Type::server_action_process_pending";
server_action_name_map[Response_Type::GET_USER_INFO_TYPE] =
    "Scan_Parse_Parameter_Type::server_action_get_user_info";
server_action_name_map[Response_Type::CREATE_HANDLE_TYPE] =
    "Scan_Parse_Parameter_Type::server_action_create_handle";
server_action_name_map[Response_Type::ADD_HANDLE_VALUE_TYPE] =
    "Scan_Parse_Parameter_Type::server_action_add_handle_value";
server_action_name_map[Response_Type::DELETE_HANDLE_TYPE] =
    "Scan_Parse_Parameter_Type::server_action_delete_handle";
server_action_name_map[Response_Type::UNDELETE_HANDLE_TYPE] =
    "Scan_Parse_Parameter_Type::server_action_undelete_handle";
server_action_name_map[Response_Type::UNDELETE_FILE_TYPE] =
    "Scan_Parse_Parameter_Type::server_action_undelete_file";

```

```

server_action_name_map[Response_Type::DELETE_HANDLE_VALUE_TYPE] =
    "Scan_Parse_Parameter_Type::server_action_delete_handle_value";
server_action_name_map[Response_Type::UNDELETE_HANDLE_VALUE_TYPE] =
    "Scan_Parse_Parameter_Type::server_action_undelete_handle_value";
server_action_name_map[Response_Type::UNDELETE_FILE_TYPE] =
    "Scan_Parse_Parameter_Type::server_action_undelete_file";
server_action_name_map[Response_Type::MAX_RESPONSE_TYPE + 1] =
    "Scan_Parse_Parameter_Type::server_action_unknown";

```

59.

```

⟨ Initialize Scan_Parse_Parameter_Type maps 57 ⟩ +≡
client_action_map[Response_Type::COMMAND_ONLY_TYPE] =
    &Scan_Parse_Parameter_Type::client_action_command_only;
client_action_map[Response_Type::SEND_FILE_TYPE] =
    &Scan_Parse_Parameter_Type::client_action_send_file;
client_action_map[Response_Type::MAX_RESPONSE_TYPE + 1] =
    &Scan_Parse_Parameter_Type::client_action_unknown;

```

60.

```

⟨ Initialize Scan_Parse_Parameter_Type maps 57 ⟩ +≡
client_action_name_map[Response_Type::COMMAND_ONLY_TYPE] =
    "Scan_Parse_Parameter_Type::client_action_command_only";
client_action_name_map[Response_Type::SEND_FILE_TYPE] =
    "Scan_Parse_Parameter_Type::client_action_send_file";
client_action_name_map[Response_Type::MAX_RESPONSE_TYPE + 1] =
    "Scan_Parse_Parameter_Type::client_action_unknown";
return 0; } /* End of Scan_Parse_Parameter_Type::initialize_maps definition */

```

61. Get Database Username. [LDF 2013.05.10.]

Log

[LDF 2013.05.10.] Added this function.

```

⟨ Scan_Parse_Parameter_Type function declarations 38 ⟩ +≡
int get_database_username(void);

```

62.

```
< Scan_Parse_Parameter_Type ::get_database_username definition 62 > ≡
int Scan_Parse_Parameter_Type ::get_database_username(void){ bool DEBUG = false;      /* true */
    set_debug_level(DEBUG);
    int status = 0;
    string thread_ctr_str = "";
{
    stringstream s;
    s << "[Thread_" << thread_ctr << "]:" ;
    thread_ctr_str = s.str();
}
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "Entering " 'Scan_Parse_Parameter_Type::get_database_username' \
        . " " << endl;
    unlock_cerr_mutex();
}      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */
See also sections 63, 64, 65, 66, 67, and 68.
This code is used in section 305.
```

63.

```

⟨ Scan_Parse_Parameter_Type ::get_database_username definition 62 ⟩ +≡
  Distinguished_Name_Type distinguished_name;
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::get_database_username':" << endl <<
      "Calling Distinguished_Name_Type::operator=" << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  distinguished_name = user_cert;
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::get_database_username':" << endl <<
      "unlock_cerr_mutex()";
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  MYSQL_RES *result = 0;
  MYSQL_ROW curr_row;
  unsigned int row_ctr = 0;
  unsigned int field_ctr = 0;
  stringstream query_strm;
  int certificate_id;
  query_strm << "select u.user_id, u.username" << "from gwiridsif.Users as u, gwiridsif.Cert\"
    ificates as c" << "where u.user_id = c.user_id and u.user_id > 0" <<
    "and ((c.serialNumber is not NULL and c.serialNumber <> 0 and c.serialNumber ="
      user_cert.serialNumber << ") or " << "(c.commonName ='" << distinguished_name.commonName <<
      "' and c.organization ='" << distinguished_name.organization <<
      "' and c.organizationalUnitName ='" << distinguished_name.organizationalUnitName <<
      "' and c.countryName ='" << distinguished_name.countryName << "') )" <<
      "order by u.user_id";
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::get_database_username':" << endl <<
      "'query_strm.str()' =='" << endl << query_strm.str() << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

64.

```

⟨ Scan_Parse_Parameter_Type :: get_database_username definition 62 ⟩ +≡
  status = submit_mysql_query(query_strm.str(), result, &row_ctr, &field_ctr);
  if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ get_timestamp() ≪ thread_ctr_str ≪ "ERROR! In 'Scan_Parse_Parameter_Type::"
      get_database_username':" ≪ endl ≪ "'submit_mysql_query' failed, returning" ≪
      status ≪ ":" ≪ endl ≪ "Error:" ≪ mysql_error(mysql_ptr) ≪
      endl ≪ "Error_number:" ≪ mysql_errno(mysql_ptr) ≪ endl ≪
      "Exiting function unsuccessfully with return value 1." ≪ endl;
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    username = "";
    ++errors_occurred;
    return 1;
  } /* if (status ≠ 0) */
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ thread_ctr_str ≪ "In 'Scan_Parse_Parameter_Type::get_database_username':" ≪ endl ≪
      "'mysql_query' succeeded, returning 0." ≪ endl ≪ "row_ctr== " ≪ row_ctr ≪ endl ≪
      "field_ctr==" ≪ field_ctr ≪ endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

65. $row_ctr \equiv 0$. [LDF 2013.05.10.]

```

⟨ Scan_Parse_Parameter_Type :: get_database_username definition 62 ⟩ +≡
  if (row_ctr ≡ 0) {
    lock_cerr_mutex();
    cerr ≪ get_timestamp() ≪ thread_ctr_str ≪ "ERROR! In 'Scan_Parse_Parameter_Type::"
      get_database_username':" ≪ endl ≪ "No rows returned." ≪ endl ≪
      "Exiting function unsuccessfully with return value 1." ≪ endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    username = "";
    ++errors_occurred;
    return 1;
  } /* if (row_ctr ≡ 0) */

```

66. $row_ctr > 1$. This shouldn't ever happen. [LDF 2013.05.10.]

```
< Scan_Parse_Parameter_Type ::get_database_username definition 62 > +≡
else
  if (row_ctr > 1) {
    lock_cerr_mutex();
    cerr << get_timestamp() << thread_ctr_str << "ERROR! In 'Scan_Pa\
rse_Parameter_Type::get_database_username':"
    << endl << row_ctr <<
    "rows returned (>1). Only one row should have been returned."
    << endl <<
    "Exiting function unsuccessfully with return value 1."
    << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    ++errors_occurred;
    return 1;
} /* else if (row_ctr ≡ 0) */
```

67. Found entries in database tables. [LDF 2013.05.10.]

```

⟨ Scan_Parse_Parameter_Type::get_database_username definition 62 ⟩ +≡
else      /* row_ctr ≡ 1 */
{
    curr_row = mysql_fetch_row(result);
    if (curr_row ≡ 0) {
        lock_cerr_mutex();
        cerr ≪ get_timestamp() ≪ thread_ctr_str ≪ "ERROR! In 'Scan_Pa\
rse_Parameter_Type::get_database_username':"
        cerr ≪ endl ≪
        "mysql_fetch_row' failed:" ≪ endl ≪ mysql_error(mysql_ptr) ≪ endl ≪
        "Exiting function unsuccessfully with return value 1." ≪ endl;
        unlock_cerr_mutex();
        if (result) mysql_free_result(result);
        ++errors_occurred;
        return 1;
    }      /* if (curr_row ≡ 0) */
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ thread_ctr_str ≪ "In 'Scan_Parse_Parameter_Type::get_database_username':"
    if (curr_row[0]) cerr ≪ "'curr_row[0]' == " ≪ curr_row[0] ≪ endl;
    else cerr ≪ "'curr_row[0]' is NULL." ≪ endl;
    unlock_cerr_mutex();
}      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */
if (curr_row[0]) {
    sscanf(curr_row[0], "%d", &user_id);
    distinguished_name.user_id = user_cert.user_id = user_id;
}
else distinguished_name.user_id = user_cert.user_id = user_id = 0;
if (curr_row[1]) {
    distinguished_name.user_name = username = user_cert.user_name = curr_row[1];
}
else {
    distinguished_name.user_name = username = user_cert.user_name = "";
    lock_cerr_mutex();
    cerr ≪ get_timestamp() ≪ thread_ctr_str ≪ "WARNING! In 'Scan_Parse_Parameter_Type\
::get_database_username':"
    cerr ≪ endl ≪ "Failed to find username." ≪ endl ≪
    "Exiting function unsuccessfully with return value 1." ≪ endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    ++errors_occurred;
    return 1;
}      /* else (¬curr_row[1]) */
}      /* else (row_ctr ≡ 1) */

```

68.

```
<Scan_Parse_Parameter_Type::get_database_username definition 62> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "Exiting `Scan_Parse_Parameter_Type::get_database_username' \\" 
            successfully" << "with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; } /* End of Scan_Parse_Parameter_Type::get_database_username definition */
```

69. Get user. (*get_user*). [LDF 2012.07.11.]

Log

[LDF 2012.07.11.] Added this function.
 [LDF 2012.09.18.] Added code for exiting unsuccessfully with return value 1 if *is_gwirdsif* ≡ *false*.
 [LDF 2012.10.08.] Added code for setting the data members *default_prefix_id*, *default_prefix*, *default_institute_id* and *default_institute_name*.
 [LDF 2013.05.10.] Added optional argument **int** *user_id* = 0. Made **const char *dn** argument optional with default 0.
 [LDF 2013.05.14.] Added optional arguments *User_Info_Type * user_info* = 0 and **bool** *set_user* = *false*.
 !! PLEASE NOTE: The default of *set_user* is “really” *true*, unless *user_info* is non-null. That is, this if *user_info* ≡ 0, *set_user* is reset to *true*. The combination of *user_info* ≡ 0 and *set_user* ≡ *false* doesn’t make any sense, because the function would then have no effect. Normally, it should either set the user, in which case *user_info* might as well be NULL, or it should store user info in a *User_Info_Type* object, in which case the current user should already have been set. However, one can do both by using *user_info* ≠ Λ and *set_user* ≡ *true*.

```
<Scan_Parse_Parameter_Type function declarations 38> +≡
int get_user(int curr_user_id = 0, const char *dn = 0, string curr_username = "", 
    User_Info_Type * user_info = 0, bool set_user = false);
```

70.

```
<Scan_Parse_Parameter_Type::get_user definition 70> ≡
int Scan_Parse_Parameter_Type::get_user(int curr_user_id, const char *dn, string
                                         curr_username, User_Info_Type *user_info, bool set_user){ bool DEBUG = false; /* true */
set_debug_level(DEBUG, 0, 0);
int status;
stringstream temp_strm;
temp_strm << "[Thread]" << thread_ctr << "]";
string thread_ctr_str = temp_strm.str();
temp_strm.str("");
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "Entering 'Scan_Parse_Parameter_Type::get_user'." << endl;
    unlock_cerr_mutex();
}
/* if (DEBUG) */
#endif /* DEBUG_COMPILE */
if (!is_gwirdsif) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "WARNING! In 'Scan_Parse_Parameter_Type::get_user':"
        endl << "'is_gwirdsif' == 'false'. This function is only meant to be"
        "used by the server" << "program 'gwirdsif'." << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    return 1;
} /* if (!is_gwirdsif) */

```

See also sections 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, and 102.

This code is used in section 305.

71.

```
<Scan_Parse_Parameter_Type::get_user definition 70> +≡
if (user_info == 0) /* See explanation above. [LDF 2013.05.14.] */
    set_user = true;
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::get_user'." << endl;
    if (user_info) cerr << "'user_info' is non-NULL." << endl;
    else cerr << "'user_info' is NULL." << endl;
    cerr << "'set_user' == " << set_user << endl;
    unlock_cerr_mutex();
}
/* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

72.

```
<Scan_Parse_Parameter_Type::get_user definition 70> +≡
  if (curr_user_id ≡ 0 ∧ dn ≡ 0 ∧ curr_username.empty()) {
    /* First, try setting curr_user_id to user_id. [LDF 2013.05.19.] */
    curr_user_id = user_id;
  }
  if (curr_user_id ≤ 0 ∧ dn ≡ 0 ∧ curr_username.empty()) {
    /* Now, if curr_user_id is still <= 0, we can't query the database. [LDF 2013.05.19.] */
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'Scan_Parse_Parameter_Type::get_user':"
    << endl << "Invalid arguments: 'curr_user_id' == "
    << curr_user_id << " and "
    << "(<= 0) and "
    << "dn" << endl << "0 and "
    << "'curr_username' is empty." << endl << "Can't query database." << endl <<
    "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
  }
  return 1;
} /* if (curr_user_id ≡ 0 ∧ dn ≡ 0 ∧ curr_username.empty()) */
```

73. !! PLEASE NOTE: *curr_user_id* has priority over *dn* and *dn* has priority over *curr_username*. Only one of the three items is used in the query! [LDF 2013.05.19.]

```

⟨ Scan_Parse_Parameter_Type::get_user definition 70 ⟩ +≡
  stringstream sql_strm;
  sql_strm << "select_udistinct_u.u.user_id,u.username,\\" 
    u.Distinguished_Name," << "u.irods_password_encrypted," <<
    "u.irods_password_encrypted_timestamp,u.irods_homedir," <<
    "u.irods_zone,u.irods_default_resource,u.default_prefix_id,up.prefix," <<
    "u.default_institute_id,ui.name,u.public_key_id" << "from_Users_as_u," <<
    "Prefixes_as_up,Users_Prefixes_as_up,Institutes_as_i"; /* Removed &user_info ≡ 0
      from the following conditional. I don't think it's needed. [LDF 2013.07.17.] */
  if (curr_user_id > 0) sql_strm << "where_u.user_id=u" << curr_user_id << "u";
  else if (dn ≠ 0) sql_strm << "where_u.Distinguished_Name=u'" << dn << ",u";
  else if (!curr_username.empty()) sql_strm << "where_u.username=u" << curr_username << ",u";
  sql_strm << "and_u.default_prefix_id=up.prefix_id_and_up.user_id=u.u.user_id" <<
    "and_up.prefix_id=up.prefix_id" << "and_u.default_institute_id=ui.institute_id";
  sql_strm << "order_by_u.user_id";
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr.str << "sql_strm.str() == " << sql_strm.str() << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  MYSQL_RES *result = 0;
  unsigned int row_ctr;
  unsigned int field_ctr;
  MYSQL_ROW curr_row;
  status = submit_mysql_query(sql_strm.str(), result, &row_ctr, &field_ctr);
  if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << thread_ctr.str << "ERROR! In 'Scan_Parse_Parameter_Type::get_user':"
    << endl << "'Scan_Parse_Parameter_Type::submit_mysql_query' failed, returning" << status <<
    "." << endl << mysql_error(mysql_ptr) << endl << "Can't set user info." << endl <<
    "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    if (result) {
      mysql_free_result(result);
    }
    ++errors_occurred;
    return 1;
  } /* if (status ≠ 0) */
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << thread_ctr.str << "In 'Scan_Parse_Parameter_Type::get_user':"
      << endl << "'Scan_Parse_Parameter_Type::submit_mysql_query' succeeded."
      << endl << "'row_ctr' == " << row_ctr << endl << "'field_ctr' == "
      << field_ctr << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */

```

```
#endif /* DEBUG_COMPILE */
```

74.

```
< Scan_Parse_Parameter_Type::get_user definition 70 > +≡
if (row_ctr ≡ 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "WARNING! In 'Scan_Parse_Parameter_Type::get_user':" << endl <<
        "'Scan_Parse_Parameter_Type::submit_mysql_query' returned 0 rows." <<
        endl << "Can't set user info." << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
unlock_cerr_mutex();
mysql_free_result(result);
++errors_occurred;
return 1;
} /* if (row_ctr ≡ 0) */
```

75.

```
< Scan_Parse_Parameter_Type::get_user definition 70 > +≡
else
if (row_ctr > 1) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "WARNING! In 'Scan_Parse_Parameter_Type::get_user':" <<
        endl << "'Scan_Parse_Parameter_Type::submit_mysql_query' returned " << row_ctr <<
        " rows (> 1)." << endl << "Will use the data from the first row. Continuing." <<
        endl;
unlock_cerr_mutex();
} /* if (row_ctr > 1) */
```

76.

```
< Scan_Parse_Parameter_Type::get_user definition 70 > +≡
if ((curr_row = mysql_fetch_row(result)) ≡ 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'Scan_Parse_Parameter_Type::get_user':" <<
        endl << "'mysql_fetch_row' failed:" << endl << mysql_error(mysql_ptr) << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
unlock_cerr_mutex();
mysql_free_result(result);
++errors_occurred;
return 1;
} /* if (curr_row = mysql_fetch_row(result) ≡ 0) */
#ifndef DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::get_user':" << endl <<
        "'mysql_fetch_row' succeeded." << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

77.

Log

[LDF 2013.05.10.] Now only setting *user_id* if it hasn't already been set, e.g., by **Scan_Parse_Parameter_Type::get_d**

```

⟨ Scan_Parse_Parameter_Type::get_user definition 70 ⟩ +≡
if (user_id ≤ 0) {
    if (curr_row[0] ≡ 0 ∨ strlen(curr_row[0]) ≡ 0) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "ERROR! " << "In 'Scan_Parse_Parameter_Type::get_user': "
        endl << "'curr_row[0]' is NULL or empty. Can't set 'user_id'." << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        mysql_free_result(result);
        ++errors_occurred;
        return 1;
    } /* if (curr_row[0] ≡ 0 ∨ strlen(curr_row[0]) ≡ 0) */
    if (set_user) user_id = atoi(curr_row[0]);
} /* if (user_id ≤ 0) */
if (user_info) {
    curr_user_id = user_info->user_id = atoi(curr_row[0]);
}
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::get_user': ";
    if (set_user) cerr << "'user_id' == " << user_id << endl;
    else if (user_info) cerr << "'user_info->user_id' == " << user_info->user_id << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

78.

Log

[LDF 2013.05.10.] Now only setting *username* if it hasn't already been set, e.g., by **Scan_Parse_Parameter_Type::get_user**

```

⟨ Scan_Parse_Parameter_Type :: get_user definition 70 ⟩ +≡
if (username.empty()) {
    if (curr_row[1] ≡ 0 ∨ strlen(curr_row[1]) ≡ 0) {
        lock_cerr_mutex();
        cerr ≪ thread_ctr_str ≪ "ERROR! " ≪ In_‘Scan_Parse_Parameter_Type::get_user’: " ≪
            endl ≪ “curr_row[1]” _is_NULL_or_empty. Can’t_set_‘username’. ” ≪ endl ≪
            “Exiting_function_unsuccessfully_with_return_value_1.” ≪ endl;
        unlock_cerr_mutex();
        mysql_free_result(result);
        ++errors_occurred;
        return 1;
    } /* if (curr_row[1] ≡ 0 ∨ strlen(curr_row[1]) ≡ 0) */
    if (set_user) username = curr_row[1];
} /* if (username.empty()) */
if (user_info) user_info->username = curr_row[1];
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ thread_ctr_str ≪ "In_‘Scan_Parse_Parameter_Type::get_user’: ";
    if (set_user) cerr ≪ “username” == “ ” ≪ username ≪ endl;
    else if (user_info) cerr ≪ “user_info->username” == “ ” ≪ user_info->username ≪ endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

79.

Log

[LDF 2013.05.17.] Added this section.

```

⟨ Scan_Parse_Parameter_Type::get_user definition 70 ⟩ +=≡
if (curr_row[2] ≡ 0 ∨ strlen(curr_row[2]) ≡ 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! " << "In 'Scan_Parse_Parameter_Type::get_user':"
    endl << "'curr_row[2]' is NULL or empty. Can't set 'distinguished_name'." << endl <<
    "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    ++errors_occurred;
    return 1;
} /* if (curr_row[2] ≡ 0 ∨ strlen(curr_row[2]) ≡ 0) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::get_user': " << endl <<
            "'curr_row[2]' == \"DistinguishedName\" " << curr_row[2] << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
string temp_username = curr_row[1];
if (set_user) {
    distinguished_name.set(curr_row[2], user_id, &temp_username);
}
if (user_info) {
    user_info->distinguished_name.set(curr_row[2], user_info->user_id, &temp_username);
}
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::get_user': " << "set_user == "
        set_user << endl;
    if (set_user) {
        distinguished_name.show("distinguished_name:");
    }
    else if (user_info) {
        user_info->distinguished_name.show("user_info->distinguished_name:");
        distinguished_name.show("distinguished_name:");
    }
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

80.

```

⟨ Scan_Parse_Parameter_Type :: get_user definition 70 ⟩ +≡
  if (curr_row[3] == 0 ∨ strlen(curr_row[3]) == 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'Scan_Parse_Parameter_Type::get_user':"
    << endl << "'curr_row[3]' is NULL or empty. Can't set 'irods_password_encrypted'."
    << endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    ++errors_occurred;
    return 1;
} /* if (curr_row[3] == 0 ∨ strlen(curr_row[3]) == 0) */
if (set_user) irods_password_encrypted = curr_row[3];
if (user_info) user_info->irods_password_encrypted = curr_row[3];
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::get_user':";
    if (set_user)
      cerr << "'irods_password_encrypted' == " << endl << irods_password_encrypted << endl;
    if (user_info) cerr << "'user_info->irods_password_encrypted' == "
      << endl << user_info->irods_password_encrypted << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

81.

```

⟨ Scan_Parse_Parameter_Type :: get_user definition 70 ⟩ +≡
  if (curr_row[4] == 0 ∨ strlen(curr_row[4]) == 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'Scan_Parse_Parameter_Type::get_user':"
    << endl << "'curr_row[4]' is NULL or empty. Can't set 'irods_password_encrypted_timestamp'\"
    . " << endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    ++errors_occurred;
    return 1;
} /* if (curr_row[4] == 0 ∨ strlen(curr_row[4]) == 0) */
if (set_user) irods_password_encrypted_timestamp = curr_row[4];
if (user_info) user_info->irods_password_encrypted_timestamp = curr_row[4];
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::get_user':";
    if (set_user) cerr << "'irods_password_encrypted_timestamp' == " << endl <<
      irods_password_encrypted_timestamp << endl;
    if (user_info) cerr << "'user_info->irods_password_encrypted_timestamp' == "
      << endl << user_info->irods_password_encrypted_timestamp << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

82.

Log

[LDF 2012.10.17.] Now setting *irods_current_dir* to *curr_row[5]* as well as *irods_homedir*.

```

⟨ Scan_Parse_Parameter_Type::get_user definition 70 ⟩ +≡
if (curr_row[5] ≡ 0 ∨ strlen(curr_row[5]) ≡ 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'Scan_Parse_Parameter_Type::get_user':"
    endl << "'curr_row[5]' is NULL or empty. Can't set 'irods_homedir'."
    endl << "Exiting function unsuccessfully with return value 1."
    endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    ++errors_occurred;
    return 1;
} /* if (curr_row[5] ≡ 0 ∨ strlen(curr_row[5]) ≡ 0) */
if (set_user) irods_current_dir = irods_homedir = curr_row[5];
if (user_info) user_info->irods_current_dir = user_info->irods_homedir = curr_row[5];
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::get_user':";
        if (set_user)
            cerr << "'irods_homedir' == 'irods_current_dir'" << endl << irods_homedir << endl;
        if (user_info)
            cerr << "'user_info->irods_homedir' == 'user_info->irods_current_dir'" << endl <<
                user_info->irods_homedir << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

83.

```

⟨ Scan_Parse_Parameter_Type :: get_user definition 70 ⟩ +≡
  if (curr_row[6] ≡ 0 ∨ strlen(curr_row[6]) ≡ 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'Scan_Parse_Parameter_Type::get_user':"
    endl << "'curr_row[6]' is NULL or empty. Can't set 'irods_zone'." << endl <<
    "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    ++errors_occurred;
    return 1;
} /* if (curr_row[6] ≡ 0 ∨ strlen(curr_row[6]) ≡ 0) */
if (set_user) irods_zone = curr_row[6];
if (user_info) user_info->irods_zone = curr_row[6];
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::get_user':";
    if (set_user) cerr << "'irods_zone'" << endl << irods_zone << endl;
    if (user_info) cerr << "'user_info->irods_zone'" << endl << user_info->irods_zone << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

84.

```

⟨ Scan_Parse_Parameter_Type :: get_user definition 70 ⟩ +≡
  if (curr_row[7] ≡ 0 ∨ strlen(curr_row[7]) ≡ 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'Scan_Parse_Parameter_Type::get_user':"
    endl << "'curr_row[7]' is NULL or empty. Can't set 'irods_default_resource'." << endl <<
    "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    ++errors_occurred;
    return 1;
} /* if (curr_row[7] ≡ 0 ∨ strlen(curr_row[7]) ≡ 0) */
if (set_user) irods_default_resource = curr_row[7];
if (user_info) user_info->irods_default_resource = curr_row[7];
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::get_user':"
    << "'irods_default_resource'" << endl << irods_default_resource << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

85.

Log

[LDF 2012.10.08.] Added this section.

```

⟨ Scan_Parse_Parameter_Type::get_user definition 70 ⟩ +≡
if (curr_row[8] ≡ 0 ∨ strlen(curr_row[8]) ≡ 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'Scan_Parse_Parameter_Type::get_user':"
        << endl <<
        "'curr_row[8]' is NULL or empty. Can't set 'default_prefix_id'." << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    ++errors_occurred;
    return 1;
} /* if (curr_row[8] ≡ 0 ∨ strlen(curr_row[8]) ≡ 0) */
if (set_user) default_handle_prefix_id = atoi(curr_row[8]);
if (user_info) user_info->default_handle_prefix_id = atoi(curr_row[8]);
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::get_user':";
        if (set_user) cerr << "'default_handle_prefix_id' == "
            << default_handle_prefix_id << endl;
        if (user_info)
            cerr << "'user_info->default_handle_prefix_id' == "
            << user_info->default_handle_prefix_id <<
            endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

86.**Log**

[LDF 2012.10.08.] Added this section.

```
<Scan_Parse_Parameter_Type::get_user definition 70> +≡
if (curr_row[9] ≡ 0 ∨ strlen(curr_row[9]) ≡ 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'Scan_Parse_Parameter_Type::get_user':"
        << endl <<
        "'curr_row[9]' is NULL or empty. Can't set 'default_handle_prefix'."
        << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    ++errors_occurred;
    return 1;
} /* if (curr_row[9] ≡ 0 ∨ strlen(curr_row[9]) ≡ 0) */
if (set_user) default_handle_prefix = curr_row[9];
if (user_info) user_info->default_handle_prefix = curr_row[9];
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::get_user':";
        if (set_user) cerr << "'default_handle_prefix' == "
            << default_handle_prefix << endl;
        if (user_info)
            cerr << "'user_info->default_handle_prefix' == "
            << user_info->default_handle_prefix << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

87.

Log

[LDF 2012.10.08.] Added this section.

```

⟨ Scan_Parse_Parameter_Type::get_user definition 70 ⟩ +≡
if (curr_row[10] ≡ 0 ∨ strlen(curr_row[10]) ≡ 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'Scan_Parse_Parameter_Type::get_user':"
    << endl << "'curr_row[10]' is NULL or empty. Can't set 'default_institute_id'."
    << endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    ++errors_occurred;
    return 1;
} /* if (curr_row[10] ≡ 0 ∨ strlen(curr_row[10]) ≡ 0) */
if (set_user) default_institute_id = atoi(curr_row[10]);
if (user_info) user_info->default_institute_id = atoi(curr_row[10]);
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::get_user':";
        if (set_user) cerr << "'default_institute_id' == "
            << default_institute_id << endl;
        if (user_info)
            cerr << "'user_info->default_institute_id' == "
            << user_info->default_institute_id << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

88.

Log

[LDF 2012.10.08.] Added this section.

```

⟨ Scan_Parse_Parameter_Type::get_user definition 70 ⟩ +≡
if (curr_row[11] ≡ 0 ∨ strlen(curr_row[11]) ≡ 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'Scan_Parse_Parameter_Type::get_user':"
    << endl <<
        "'curr_row[11]' is NULL or empty. Can't set 'default_institute_name'."
    << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    ++errors_occurred;
    return 1;
} /* if (curr_row[11] ≡ 0 ∨ strlen(curr_row[11]) ≡ 0) */
if (set_user) default_institute_name = curr_row[11];
if (user_info) user_info->default_institute_name = curr_row[11];
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::get_user':";
        if (set_user) cerr << "'default_institute_name' == "
            << default_institute_name << endl;
        if (user_info)
            cerr << "'user_info->default_institute_name' == "
            << user_info->default_institute_name << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

89. GPG Public key ID (*public_key_id*). [LDF 2013.09.20.]

Log

[LDF 2013.09.20.] Added this section.

```

⟨ Scan_Parse_Parameter_Type :: get_user definition 70 ⟩ +≡
if (curr_row[12] ≡ 0 ∨ strlen(curr_row[12]) ≡ 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'Scan_Parse_Parameter_Type::get_user':"
    << endl << "'curr_row[12]' is NULL or empty. Can't set 'public_key_id'."
    << endl << "Exiting function unsuccessfully with return value 1."
    << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    ++errors_occurred;
    return 1;
} /* if (curr_row[12] ≡ 0 ∨ strlen(curr_row[12]) ≡ 0) */
if (set_user) public_key_id = curr_row[12];
if (user_info) user_info->public_key_id = curr_row[12];
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::get_user':";
        if (set_user) cerr << "'public_key_id' == "
        << public_key_id << endl;
        if (user_info) cerr << "'user_info->public_key_id' == "
        << user_info->public_key_id << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

90.

```

⟨ Scan_Parse_Parameter_Type :: get_user definition 70 ⟩ +≡
mysql_free_result(result);
result = 0;

```

91. Get privileges. [LDF 2013.05.16.]

Log

[LDF 2013.05.16.] Added this section.

```

⟨ Scan_Parse_Parameter_Type :: get_user definition 70 ⟩ +≡
    unsigned int temp_privileges = 0U;
    status = get_privileges( curr_user_id, &temp_privileges );
    bitset<sizeof(unsigned int) * 8> temp_bitset( temp_privileges );
    if (status ≡ 2) {
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::get_user': "
                << "'Scan_Parse_Parameter_Type::get_privileges' succeeded, "
                << "returning 2." << endl
                << "No entry in database table 'gwirdsif.Privileges' for "
                << "user " << ((curr_user_id > 0) ? curr_user_id : user_id) << endl
                << "'temp_privileges' = " << temp_bitset
                /* Should be 0. [LDF 2013.05.16.] */
                << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    } /* if (status ≡ 2) */
    else if (status ≠ 0) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "ERROR! In 'Scan_Parse_Parameter_Type::get_user': "
            << "'Scan_Parse_Parameter_Type::get_privileges' failed, "
            << "returning " << status << "."
            << endl << "Failed to retrieve privileges for user "
            << curr_user_id << "."
            << endl << "Exiting function unsuccessfully with return value 1."
            << endl;
        unlock_cerr_mutex();
        ++errors_occurred;
        return 1;
    } /* if (status ≠ 0) */
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::get_user': "
                << "'Scan_Parse_Parameter_Type::get_privileges' succeeded, "
                << "returning 0." << endl
                << "'temp_privileges' = " << temp_bitset << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    if (set_user) privileges = temp_privileges;
    if (user_info) {
        user_info->privileges = temp_privileges;
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            bitset<sizeof(unsigned int) * 8> ui_priv(user_info->privileges);
            lock_cerr_mutex();

```

```

    cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::get_user':"
    << "user_info->privileges' == "
    << ui_priv << endl;
    unlock_cerr_mutex();
}
/* if (DEBUG) */
#endif /* DEBUG_COMPILE */
}

```

92. Retrieve X.509 certificate data. [LDF 2013.05.17.]

Log

[LDF 2013.05.17.] Added this section.

```

⟨ Scan_Parse_Parameter_Type :: get_user definition 70 ⟩ +≡
X509_Cert_Type temp_cert;
int temp_user_id = (set_user) ? user_id : user_info->user_id;
status = temp_cert.get_from_database(mysql_ptr, temp_user_id, thread_ctr_str);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'Scan_Parse_Parameter_Type::get_user':"
    << "x509_Cert_Type::get_from_database' failed, "
    << "returning "
    << status << ". "
    << endl << "Failed to retrieve X.509 certificate data for user "
    << temp_user_id << ". "
    << endl << "Exiting function unsuccessfully with return value 1. "
    << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    return 1;
} /* if (status ≠ 0) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::get_user':"
        << "x509_Cert_Type::get_from_database' succeeded, "
        << "returning 0. "
        << endl << "Retrieved X.509 certificate data for user "
        << temp_user_id << " successfully. "
        << endl;
        temp_cert.show("temp_cert:");
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
if (set_user) {
    if (user_cert.serialNumber ≡ 0) {
        /* This will be the case if user_cert has been set by verify_certificate and not read from the
           database. certificate_id is internal to gwrdifpk and not part of the actual X.509 certificate.
           [LDF 2013.05.22.] */
        user_cert = temp_cert;
    }
    else user_cert.certificate_id = temp_cert.certificate_id;
}
if (user_info) user_info->certificate = temp_cert;

```

93.

```
<Scan_Parse_Parameter_Type::get_user definition 70> +≡
  if (set_user) {
    Response_Type response;
    response.type = Response_Type::COMMAND_ONLY_TYPE;
    temp_strm.str("");
    temp_strm << "GET_USER_RESPONSE\0USER_ID\0" << user_id << "\0" << "USER_NAME\" \" <<
      username << "\0PRIVILEGES\0" << temp_privileges << "U";
    response.command = temp_strm.str();
#define DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "response.command==\" " << response.command << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    response_deque.push_front(response);
    temp_strm.str("");
  } /* if (set_user) */
```

94. If *set_user* ≡ *false*, return successfully. In this case, we don't retrieve the iRODS password from the database, decrypt it, and write it to a file. Instead, we just return successfully with return value 0. [LDF 2013.05.17.]

```
<Scan_Parse_Parameter_Type::get_user definition 70> +≡
  if (set_user ≡ false) {
#define DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::get_user':\n" << endl <<
      "'set_user' == 'false'. Not decrypting iRODS password and writing it to file.\n" <<
      endl << "Exiting function successfully with return value 0.\n" << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  return 0;
} /* if (set_user ≡ false) */
```

95. Lock *buffer* into into RAM. [LDF 2012.07.12.]

Log

[LDF 2012.07.12.] Added this section.

```

⟨ Scan_Parse_Parameter_Type::get_user definition 70 ⟩ +≡
char buffer[256];
memset(buffer, 0, 256);
errno = 0;
status = mlock(buffer, 256);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << thread_ctr.str << "ERROR! In 'Scan_Parse_Parameter_Type::get_user': 'mlock' \n"
        failed, " returning" << status << "." << endl << strerror(errno) << endl <<
        "Can't lock memory for decrypted iRODS password for user '" << username << "'." <<
        endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    return 1;
} /* if (status ≠ 0) */
#endif DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr.str << "In 'Scan_Parse_Parameter_Type::get_user': 'mlock' succeeded." << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

96.

Log

[LDF 2012.11.16.] Now pushing *temp_filename* onto **vector<string>** *temp_file_vector*, so it will be deleted at the end of the run (thread).

```

⟨ Scan_Parse_Parameter_Type::get_user definition 70 ⟩ +≡
char temp_filename[] = "/tmp/gwirdsif.XXXXXX";
int fd = mkstemp(temp_filename);
if (fd == -1) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'Scan_Parse_Parameter_Type::get_user': "
        "mkstemp' failed, returning -1." << endl << "mkstemp error: " << strerror(errno) <<
        endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    return 1;
} /* if (fd == -1) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::get_user': 'mkstemp' success\"
            ded. " << 'temp_filename' == " << " " << temp_filename << ":" << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
if (user_info == 0) temp_file_vector.push_back(temp_filename); /* irods_auth_filename */
close(fd);
fd = 0;

```

97. Decrypt iRODS password. [LDF 2012.07.11.]

Log

[LDF 2012.07.11.] Added this section.

[LDF 2012.09.12.] Now storing timestamp for the “scrambled” iRODS password in the database and retrieving it for the call to `touch` below.

[LDF 2013.04.17.] Now calling `touch` with the `-m` option for changing only the time of last modification. The time of last access is not changed. The modification time must be that of the original file created by `iinit`, otherwise the iRODS server considers the “scrambled” password to be invalid. The last access time is used in `purge_server_logs` to determine whether the temporary file can be deleted.

[LDF 2013.09.27.] Added code for passing a passphrase to `gpg`. This will work even if the secret key used for decryption doesn’t have a passphrase. In this case, `gpg` will succeed, even though a passphrase is supplied. It doesn’t matter what it is. However, I have now switched over to using iRODS passwords that have been encrypted with a keypair where the secret key does have a passphrase.

```

⟨ Scan_Parse_Parameter_Type::get_user definition 70 ⟩ +≡
temp_strm.str("");
if (gpg_passphrase ≠ 0 ∧ strlen(gpg_passphrase) > 0) {
    status = write_to_fifo(gpg_passphrase, gpg_passphrase_length, gpg_passphrase_fifo_fd);
    if (status ≠ 0) {
        lock_cerr_mutex();
        cerr ≪ thread_ctr_str ≪ "ERROR! In 'Scan_Parse_Parameter_Type::get_user': "
        ≪ "'write_to_fifo' failed, returning" ≪ status ≪ "."
        ≪ endl ≪ "Failed to write 'gpg_passphrase' to FIFO" ≪ ""
        ≪ gpg_passphrase_fifo_name ≪ "."
        ≪ endl ≪ "Exiting function unsuccessfully with return value 1."
        ≪ endl;
        unlock_cerr_mutex();
        ++errors_occurred;
    }
    return 1;
} /* if (status ≠ 0) */
#endif DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ thread_ctr_str ≪ "In 'Scan_Parse_Parameter_Type::get_user': "
    ≪ "'write_to_fifo' succeeded, returning 0."
    ≪ endl ≪ "Wrote 'gpg_passphrase' to FIFO"
    ≪ ""
    ≪ gpg_passphrase_fifo_name ≪ "."
    ≪ endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
/* if (gpg_passphrase ≠ 0 ∧ strlen(gpg_passphrase) > 0) */
temp_strm ≪ "env LANG=en_US.UTF-8 echo -e \""
            ≪ irods_password_encrypted ≪ "\n"
            ≪ "gpg --batch --decrypt";
if (gpg_passphrase ≠ 0 ∧ strlen(gpg_passphrase) > 0)
    temp_strm ≪ "--passphrase-file" ≪ gpg_passphrase_fifo_name ≪ ",";
temp_strm ≪ "2>/dev/null >" ≪ temp_filename ≪ " "
            ≪ "&& touch -m" ≪ temp_filename ≪ "
            ≪ "--date=\\\""
            ≪ irods_password_encrypted_timestamp ≪ "\\\""
            ≪ "echo $?";
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ thread_ctr_str ≪ "temp_strm.str() == "
            ≪ temp_strm.str() ≪ endl;
    unlock_cerr_mutex();
}

```

```
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
FILE *fp = popen(temp_strm.str().c_str(), "r");
if (fp == 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'Scan_Parse_Parameter_Type::get_user':" << endl <<
        "'popen' failed, returning NULL. Failed to decrypt iRODS password for "
        "user " << username << " or write it to " << temp_filename << " or to set units "
        "timestamp." << endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    return 1;
} /* if (fp == 0) */
#endif DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::get_user':" << endl <<
        "'popen' succeeded." << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

98. Read exit status GPG command from *fp*. This will be non-zero if signature verification failed.
[LDF 2012.07.12.]

```

⟨ Scan_Parse_Parameter_Type::get_user definition 70 ⟩ +≡
  char *temp_str = fgets(buffer, 16, fp);      /* Testing LDF 2012.09.10. */
  pclose(fp);
  fp = 0;
  int temp_ret_val = atoi(buffer);
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::get_user':"
    << endl << "'fgets' succeeded, reading"
    << strlen(buffer) << " characters."
    << endl << "'temp_ret_val' == "
    << temp_ret_val << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  if (temp_ret_val != 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'Scan_Parse_Parameter_Type::get_user':"
    << endl << "'temp_ret_val' == "
    << temp_ret_val << "(!=0)."
    << endl << "'gpg--decrypt' or 'touch' failed. Possibly the signature was invalid."
    << endl << "Authentication/authorization failed for user '"
    << username << "."
    << endl << "Exiting function unsuccessfully with return value 1."
    << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    return 1;
  } /* if (temp_ret_val != 0) */
#endif DEBUG_COMPILE
  else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::get_user':"
    << endl << "'temp_ret_val' == 0."
    << endl << "'gpg--decrypt' and 'touch' succeeded. Signature is valid."
    << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

99.

```
<Scan_Parse_Parameter_Type::get_user definition 70> +≡
  errno = 0;
  memset(buffer, 0, 256);
  status = munlock(buffer, 256);
  if (status ≡ -1) {
    lock_cerr_mutex();
    cerr ≪ thread_ctr_str ≪ "ERROR! In 'Scan_Parse_Parameter_Type::get_user': "
    ≪ "'munlock' failed, returning -1." ≪ endl ≪ "munlock error: " ≪ strerror(errno) ≪
    endl ≪ "Exiting function unsuccessfully with return value 1." ≪ endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    return 1;
  } /* if (status ≡ -1) */
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr ≪ thread_ctr_str ≪ "Unlocked memory successfully." ≪ endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  irods_auth_filename = temp_filename;
  if (user_info) user_info->irods_auth_filename = irods_auth_filename;
```

100. Create iRODS environment file. [LDF 2013.03.22.]

Log

[LDF 2013.03.22.] Added this section.

```
{ Scan_Parse_Parameter_Type::get_user definition 70 } +≡
    strcpy(temp_filename, "/tmp/gwirdsif.XXXXXX");
    fd = mkstemp(temp_filename);
    if (fd == -1) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "ERROR! In 'Scan_Parse_Parameter_Type::get_user': "
            "mkstemp failed, returning -1." << endl << "mkstemp error: " << strerror(errno) <<
            endl << "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        ++errors_occurred;
        return 1;
    } /* if (fd == -1) */
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::get_user': 'mkstemp' success\"
                ded." << "'temp_filename'==" << " " << temp_filename << ". " << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    if (user_info == 0) temp_file_vector.push_back(temp_filename); /* irods_env_filename */
    close(fd);
    fd = 0;
    irods_env_filename = temp_filename;
    if (user_info) user_info->irods_env_filename = irods_env_filename;
```

101. Write environment variables for iRODS to *env-strm*, i.e., *irods-env-filename*.

!! PLEASE NOTE: The file *.irods/.irodsEnv* must exist below the home directory of the user under whose account the server program ‘gwrdsif’ is running, even though a different environment file is used! This would appear to be a bug in iRODS. [LDF 2013.03.22.]

Log

[LDF 2013.03.22.] Added this section.

```

⟨ Scan_Parse_Parameter_Type::get_user definition 70 ⟩ +≡
ofstream env_strm;
env_strm.open(irods_env_filename.c_str());
if (!env_strm.is_open()) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Scan_Parse_Parameter_Type::get_user':"
        endl << "'ofstream::open' failed, returning 'false':"
        endl << "Failed to open "
        << irods_env_filename << "."
        endl << "Exiting function unsuccessfully with return value 1."
        endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    return 1;
} /* if (!env_strm.is_open()) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'Scan_Parse_Parameter_Type::get_user':"
            endl << "'ofstream::open' succeeded, returning 'true':"
            endl << "Opened 'irods_env_filename' == "
            << irods_env_filename << " successfully."
            endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
env_strm << "irodsHost" << gwrdsif_hostname.c_str() << ","
        endl << "irodsPort" << 1247 <<
        endl << "irodsDefResource" << irods_default_resource.c_str() << ","
        endl << "irodsHome" <<
        irods_homedir.c_str() << ","
        endl << "irodsCwd" << irods_homedir.c_str() << ","
        endl << "irodsUserName" << username.c_str() << ","
        endl << "irodsZone" << irods_zone.c_str() <<
        ","
        endl << "irodsAuthFileName" << irods_auth_filename << ","
        endl;
env_strm.close();

```

102.

```

⟨ Scan_Parse_Parameter_Type::get_user definition 70 ⟩ +≡
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "Exiting 'Scan_Parse_Parameter_Type::get_user' successfully\
        with return value 0."
        endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
if (result) mysql_free_result(result);
return 0; /* Scan_Parse_Parameter_Type::get_user */

```

103. Set user info. (*set_user_info*). [LDF 2013.05.19.]

Log

[LDF 2013.05.19.] Added this function declaration. The definition is in `spptfnc1.web`.

`(Scan_Parse_Parameter_Type function declarations 38) +≡`

```
int set_user_info(User_Info_Type & user_info)
    const;
```

104. get privileges. (*get_privileges*). [LDF 2013.05.16.]

Log

[LDF 2013.05.16.] Added this function declaration. The definition is in `spptfnc1.web`.

`(Scan_Parse_Parameter_Type function declarations 38) +≡`

```
int get_privileges(int curr_user_id = 0, unsigned int *privs = 0);
```

105. Show. [LDF 2012.06.27.]

Log

[LDF 2012.06.27.] Added this function.

[LDF 2012.07.27.] Added optional argument `stringstream *strm` with default 0.

[LDF 2012.11.16.] Added code for outputting the filenames on `temp_file_vector`, if present.

[LDF 2013.04.25.] Added code for outputting the value of `int thread_cancel_state`.

`(Scan_Parse_Parameter_Type function declarations 38) +≡`

```
void show(string s = "Scan_Parse_Parameter_Type:", stringstream *strm = 0);
```

106.

```

< Scan_Parse_Parameter_Type ::show definition 106 > ≡
void Scan_Parse_Parameter_Type ::show(string s, stringstream *strm)
{
    stringstream temp_strm;
    temp_strm ≪ s ≪ endl ≪ "user_id==oooooooooooooo" ≪ user_id ≪ endl ≪
        "username==ooooooooooooo" ≪ username ≪ endl;
    distinguished_name.show("distinguished_name:", &temp_strm);
    temp_strm ≪ "privileges:" ≪ endl ≪ "uuusuperuser:oooooooooooooo" ≪
        (((privileges & SUPERUSER_PRIVILEGE) > 0_U) ? 1 : 0) ≪ endl ≪
        "uuudelegate:oooooooooooooo" ≪ (((privileges & DELEGATE_PRIVILEGE) >
        0_U) ? 1 : 0) ≪ endl ≪ "uuudelete_handles:oooooooooooooo" ≪
        (((privileges & DELETE_HANDLES_PRIVILEGE) > 0_U) ? 1 : 0) ≪
        endl ≪ "uuudelete_handle_values:ooooooooooooo" ≪
        (((privileges & DELETE_HANDLE_VALUES_PRIVILEGE) > 0_U) ? 1 :
        0) ≪ endl ≪ "uuudelete_hs_admin_handle_values:oooooo" ≪
        (((privileges & DELETE_HS_ADMIN_HANDLE_VALUES_PRIVILEGE) > 0_U) ?
        1 : 0) ≪ endl ≪ "uuudelete_last_hs_admin_handle_value:uu" ≪
        (((privileges & DELETE_LAST_HS_ADMIN_HANDLE_VALUE_PRIVILEGE) > 0_U) ?
        1 : 0) ≪ endl ≪ "uuudelete_handle_values:ooooooooooooo" ≪
        (((privileges & UNDELETE_HANDLE_VALUES_PRIVILEGE) > 0_U) ? 1 :
        0) ≪ endl ≪ "uuushow_user_info:oooooooooooooo" ≪
        (((privileges & SHOW_USER_INFO_PRIVILEGE) > 0_U) ? 1 : 0) ≪ endl ≪
        "uuushow_groups:ooooooooooooooo" ≪ (((privileges & SHOW_GROUPS_PRIVILEGE) >
        0_U) ? 1 : 0) ≪ endl ≪ "uuushow_certificates:ooooooooooooooo" ≪
        (((privileges & SHOW_CERTIFICATES_PRIVILEGE) > 0_U) ? 1 :
        0) ≪ endl ≪ "uuushow_distinguished_names:oooooooooooo" ≪
        (((privileges & SHOW_DISTINGUISHED_NAMES_PRIVILEGE) > 0_U) ? 1 : 0) ≪ endl ≪
        "uuushow_privileges:ooooooooooooooo" ≪ (((privileges & SHOW_PRIVILEGES_PRIVILEGE) >
        0_U) ? 1 : 0) ≪ endl ≪ "sock==" ≪ sock ≪ endl ≪ "irods_auth_filename===" ≪
        irods_auth_filename ≪ endl ≪ "irods_env_filename===" ≪ irods_env_filename ≪ endl ≪
        "irods_zone==" ≪ irods_zone ≪ endl ≪ "irods_default_resource==" ≪
        irods_default_resource ≪ endl ≪ "irods_homedir==" ≪ irods_homedir ≪ endl ≪
        "irods_current_dir==" ≪ irods_current_dir ≪ endl ≪ "data_filename==" ≪
        data_filename ≪ endl ≪ "input_commands==" ≪ input_commands ≪ endl ≪
        "public_key_id==" ≪ public_key_id ≪ endl;
    if (group_vector.size() > 0) {
        temp_strm ≪ "group_vector:" ≪ endl;
        for (vector<Group_Type>::const_iterator iter = group_vector.begin(); iter != group_vector.end();
            ++iter) iter->show("", &temp_strm);
    }
    else temp_strm ≪ "group_vectorisempty" ≪ endl;
    if (strlen(data_buffer) == 0) temp_strm ≪ "strlen(data_buffer)==0" ≪ endl;
    else {
        temp_strm ≪ "strlen(data_buffer)" ≪ strlen(data_buffer) ≪ endl ≪ "data_buffer==" ≪
            endl ≪ data_buffer ≪ endl;
    }
    temp_strm ≪ "server_finished==" ≪ server_finished ≪ endl ≪ "client_finished==" ≪
        client_finished ≪ endl;
    temp_strm ≪ "default_handle_prefix_id==" ≪ default_handle_prefix_id ≪
        endl ≪ "default_handle_prefix==" ≪ default_handle_prefix ≪
        endl ≪ "default_institute_id==" ≪ default_institute_id ≪ endl ≪

```

```

"default_institute_name" << default_institute_name << endl << "delay_value" <<
delay_value << endl;
if (irods_object) temp_strm << "'irods_object' is non-null." << endl;
else temp_strm << "'irods_object' is NULL." << endl;
temp_strm << "'temp_file_vector.size()' " << temp_file_vector.size() << endl;
if (temp_file_vector.size() > 0) temp_strm << "Filenames:" << endl;
for (vector<string>::const_iterator iter = temp_file_vector.begin(); iter != temp_file_vector.end());
    ++iter) temp_strm << " " << *iter << endl;
temp_strm << "'string_vector.size()' " << string_vector.size() << endl;
if (string_vector.size() > 0) temp_strm << "Strings:" << endl;
for (vector<string>::const_iterator iter = string_vector.begin(); iter != string_vector.end(); ++iter)
    temp_strm << " " << *iter << endl;
temp_strm << "'errors_occurred'" << errors_occurred << endl <<
"'warnings_occurred'" << warnings_occurred << endl << "response_deque.size()" <<
response_deque.size() << endl << "delayed_response_deque.size()" << endl <<
delayed_response_deque.size() << endl << "pending_operations_flag" << endl <<
pending_operations_flag << endl << "thread_cancel_state" << endl;
if (thread_cancel_state == PTHREAD_CANCEL_ENABLE) temp_strm << "PTHREAD_CANCEL_ENABLE" << endl;
else if (thread_cancel_state == PTHREAD_CANCEL_DISABLE)
    temp_strm << "PTHREAD_CANCEL_DISABLE" << endl;
else temp_strm << thread_cancel_state << endl; /* This should never occur. [LDF 2013.04.25.] */
temp_strm << "user_id_map.size()" << user_id_map.size() << endl;
if (user_id_map.size() > 0) {
    temp_strm << "user_id_map:" << endl;
    for (map<string, int>::const_iterator iter = user_id_map.begin(); iter != user_id_map.end());
        ++iter) {
            temp_strm << iter->first << ", " << iter->second << endl;
        }
    temp_strm << endl;
}
temp_strm << "user_info_map.size()" << user_info_map.size() << endl;
if (user_info_map.size() > 0) {
    temp_strm << "user_info_map:" << endl;
    for (map<int,
        User_Info_Type>::const_iterator iter = user_info_map.begin(); iter != user_info_map.end());
        ++iter) {
            temp_strm << iter->first << ":" << endl;
            iter->second.show("User_Info_Type:", &temp_strm);
        }
    temp_strm << endl;
}
if (is_gwirdsif) {
    user_cert.show("user_cert:", &temp_strm, false);
} /* if (is_gwirdsif) */
if (cert_ptr) {
    temp_strm << "'cert_ptr' is non-NULL." << endl;
    cert_ptr->show("*cert_ptr:", &temp_strm, false);
}
else temp_strm << "'cert_ptr' is NULL." << endl;
if (user_info_ptr) {
    temp_strm << "'user_info_ptr' is non-NULL." << endl;
    user_info_ptr->show("*user_info_ptr:", &temp_strm);
}

```

```

    }
  else temp_strm << "‘user_info_ptr’ is NULL." << endl;
  if (strm) *strm << temp_strm.str();
  else cerr << temp_strm.str();
  return;
} /* End of Scan_Parse_Parameter_Type::show definition */

```

This code is used in section 305.

107. Submit SQL query. [LDF 2012.07.11.]

Log

[LDF 2012.07.11.] Added this function. I've copied it from the version I wrote for the OptiNum-Grid project and modified it slightly.

[LDF 2012.07.16.] Moved the original version of this function from this file (`scprpmtp.web`) to `utilfncts.web`. Changed it, so that it is no longer a member function of `Scan_Parse_Parameter_Type`. Added the required argument `MYSQL *mysql_ptr` and the optional argument `string thread_ctr_str` with default "".

The new version of this function below calls the non-class member version in `utilfncts.web`, passing `this` and a string for the “thread counter” as arguments.

[LDF 2012.09.18.] Added code for exiting unsuccessfully with return value 1 if `is_gwirdsif` \equiv false.

[LDF 2013.09.03.] Removed code that tests the value of `is_gwirdsif` and exits if it's false. Previously, this function was only meant to be used by `gwirdsif`, but that's not the case anymore.

```
⟨ Scan_Parse_Parameter_Type function declarations 38 ⟩ +≡
int submit_mysql_query(string query, MYSQL_RES *&result, unsigned int *row_ctr = 0, unsigned int
                       *field_ctr = 0, long *affected_rows = 0);
```

108.

```
⟨ Scan_Parse_Parameter_Type ::submit_mysql_query definition 108 ⟩ ≡
int Scan_Parse_Parameter_Type ::submit_mysql_query(string query, MYSQL_RES *&result, unsigned
                                                    int *row_ctr, unsigned int *field_ctr, long *affected_rows)
{
  stringstream temp_strm;
  temp_strm << "[Thread_" << thread_ctr << "]"; /* :: causes the global (i.e., non-class) version of
                                                    submit_mysql_query to be called. initialsLDF 2012.07.16. */
  return ::submit_mysql_query(query, result, mysql_ptr, row_ctr, field_ctr, affected_rows, temp_strm.str());
}
```

See also sections 110, 111, 112, and 113.

This code is used in section 305.

109. Submit multiple SQL queries (`Scan_Parse_Parameter_Type::submit_mysql_queries`). [LDF 2013.09.03.]

```
⟨ Scan_Parse_Parameter_Type function declarations 38 ⟩ +≡
int submit_mysql_queries(vector<string> &query_vector, MYSQL_RES **result_array, vector<unsigned
                           int *> &row_ctr_vector, vector<unsigned int *> &field_ctr_vector, vector<long int *>
                           &affected_rows_vector, bool continue_on_error = false);
```

110.

```
< Scan_Parse_Parameter_Type :: submit_mysql_query definition 108 > +≡
int Scan_Parse_Parameter_Type :: submit_mysql_queries( vector<string> &query_vector,
    MYSQL_RES **result_array, vector<unsigned int *> &row_ctr_vector, vector<unsigned int *>
    &field_ctr_vector, vector<long int *> &affected_rows_vector, bool continue_on_error){ bool
    DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    string thread_str;
    stringstream temp_strm;
    temp_strm << "[Thread]" << thread_ctr << "]";
    thread_str = temp_strm.str();
    temp_strm.str("");
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering 'Scan_Parse_Parameter_Type::submit_mysql_queries'." <<
            endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
```

111.

```
< Scan_Parse_Parameter_Type :: submit_mysql_query definition 108 > +≡
status = ::submit_mysql_queries(query_vector, result_array, mysql_ptr, row_ctr_vector, field_ctr_vector,
    affected_rows_vector, continue_on_error, thread_str);
if (status != 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::submit_mysql_queries':" <<
        endl << "Non-class function 'submit_mysql_queries' failed, returning" <<
        status << ":" << endl << mysql_error(mysql_ptr) << endl <<
        "Exiting function unsuccessfully with return value" << status << "." << endl;
    unlock_cerr_mutex();
    return 1;
} /* if (status != 0) */
```

112.

```
< Scan_Parse_Parameter_Type :: submit_mysql_query definition 108 > +≡
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Scan_Parse_Parameter_Type::submit_mysql_queries':" << endl <<
            "Non-class function 'submit_mysql_queries' succeeded, returning 0." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

113.

```
< Scan_Parse_Parameter_Type :: submit_mysql_query definition 108 > +≡
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "Exiting `Scan_Parse_Parameter_Type::submit_mysql_queries'" <<
        "successfully with return value 0." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
  return 0; } /* End of Scan_Parse_Parameter_Type :: submit_mysql_queries definition */
```

114. Send TAN list. [LDF 2012.07.19.]

Log

[LDF 2012.07.19.] Added this function.

[LDF 2012.09.18.] Added code for exiting unsuccessfully with return value 1 if *is_gwirdsif* ≡ false.

```
< Scan_Parse_Parameter_Type function declarations 38 > +≡
  int send_tan_list(void);
```

115.

```
< Scan_Parse_Parameter_Type :: send_tan_list definition 115 > ≡
  int Scan_Parse_Parameter_Type :: send_tan_list(void){ bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    if (!is_gwirdsif) {
      lock_cerr_mutex();
      cerr << "[Thread" << thread_ctr << "] WARNING!" <<
          " In `Scan_Parse_Parameter_Type::send_tan_list':"
          << endl <<
          "'is_gwirdsif' == 'false'. This function is only meant to be
          used by the server" << " program `gwirdsif'." << endl <<
          "Exiting function unsuccessfully with return value 1." << endl;
      unlock_cerr_mutex();
      ++errors_occurred;
      return 1;
    } /* if (!is_gwirdsif) */
    stringstream temp_strm;
    temp_strm << "[Thread" << thread_ctr << "]";
    string thread_ctr_str = temp_strm.str();
    temp_strm.str("");
    int status;
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "Entering `Scan_Parse_Parameter_Type::send_tan_list'." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

See also sections 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, and 129.

This code is used in section 305.

116.

```
< Scan_Parse_Parameter_Type :: send_tan_list definition 115 > +≡
  MYSQL_RES * result = 0;
  MYSQL_ROW curr_row;
  unsigned int row_ctr;
  unsigned int field_ctr;
  long int affected_rows;
  stringstream sql_strm;
  int TAN_ctr = 0;
```

117. Lock TANs table. [LDF 2012.07.19.]

```
< Scan_Parse_Parameter_Type :: send_tan_list definition 115 > +≡
  status = submit_mysql_query("lock_tables_TANs_write", result, 0, 0, 0);
  if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ thread_ctr_str ≪ "ERROR! In 'Scan_Parse_Parameter_Type::send_tan_list':"
    endl ≪ "'submit_mysql_query' failed, returning" ≪ status ≪
    "." ≪ endl ≪ "Failed to lock 'TANs' table." ≪ endl ≪
    "Exiting function unsuccessfully with return value 1." ≪ endl;
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    ++errors_occurred;
    return 1;
  } /* if (status ≠ 0) */
#endif /* DEBUG_COMPILE
else
if (DEBUG) {
  lock_cerr_mutex();
  cerr ≪ thread_ctr_str ≪ "In 'Scan_Parse_Parameter_Type::send_tan_list':"
  endl ≪ "'submit_mysql_query' succeeded, returning 0." ≪ endl ≪
  "Locked 'TANs' table successfully." ≪ endl;
  unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
mysql_free_result(result);
result = 0;
```

118.

```

⟨ Scan_Parse_Parameter_Type :: send_tan_list definition 115 ⟩ +≡
sql_strm << "select_count(TAN)from_TANswhere_user_id=0"; #
if DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "sql_strm.str() == " << sql_strm.str() << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
status = submit_mysql_query(sql_strm.str(), result, &row_ctr, &field_ctr, 0);
if (status != 0) {
  lock_cerr_mutex();
  cerr << thread_ctr_str << "ERROR! In 'Scan_Parse_Parameter_Type::send_tan_list':"
      << endl << "'submit_mysql_query' failed, returning " << status << "."
      << "Exiting function unsuccessfully with return value 1." << endl;
  unlock_cerr_mutex();
  if (result)
    mysql_free_result(result);
  result = 0;
}
submit_mysql_query("unlock_tables", result, 0, 0, 0);
if (result) mysql_free_result(result);
++errors_occurred;
return 1;
} /* if (status != 0) */
#endif DEBUG_COMPILE
else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::send_tan_list':"
        << endl << "'submit_mysql_query' succeeded, returning 0."
        << endl << "'row_ctr' == "
        << row_ctr << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

119.

```

⟨ Scan_Parse_Parameter_Type :: send_tan_list definition 115 ⟩ +≡
  if ((curr_row = mysql_fetch_row(result)) == 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'Scan_Parse_Parameter_Type::send_tan_list':"
    endl << "'mysql_fetch_row' failed:" << endl << mysql_error(mysql_ptr) << endl <<
    "Exiting 'Scan_Parse_Parameter_Type::send_tan_list' unsuccessfully" <<
    "with return value 1." << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    submit_mysql_query("unlock_tables", result, 0, 0, 0);
    if (result) mysql_free_result(result);
    ++errors_occurred;
    return 1;
} /* if (curr_row = mysql_fetch_row(result) == 0) */
#ifndef DEBUG_COMPILE
else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::send_tan_list':"
    << endl <<
    "'mysql_fetch_row' succeeded." << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

120.

```

⟨ Scan_Parse_Parameter_Type :: send_tan_list definition 115 ⟩ +≡
  if (curr_row[0] == 0 || strlen(curr_row[0]) == 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'Scan_Parse_Parameter_Type::send_tan_list':"
    endl << "'curr_row[0]' is NULL or empty. Can't set 'TAN_ctr'." << endl <<
    "Exiting 'Scan_Parse_Parameter_Type::send_tan_list' unsuccessfully" <<
    "with return value 1." << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    submit_mysql_query("unlock_tables", result, 0, 0, 0);
    if (result) mysql_free_result(result);
    ++errors_occurred;
    return 1;
} /* if (curr_row[0] == 0 || strlen(curr_row[0]) == 0) */
TAN_ctr = atoi(curr_row[0]);
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::send_tan_list':"
    << "'curr_row[0]' == " << curr_row[0] << endl << "'TAN_ctr' == " << TAN_ctr << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  mysql_free_result(result);
  result = 0;

```

121.

```

⟨ Scan_Parse_Parameter_Type :: send_tan_list definition 115 ⟩ +≡
  if (TAN_ctr < 100) {
#if DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::send_tan_list':" << endl <<
      "('TAN_ctr' == " << TAN_ctr << ") << 100" << endl << "Calling 'generate_tans'." <<
      endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  status = generate_tans(mysql_ptr, 200, 100, thread_ctr_str);
  if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'Scan_Parse_Parameter_Type::send_tan_list':" <<
      endl << "'generate_tans' failed, returning" << status << "." << endl <<
      "Exiting 'Scan_Parse_Parameter_Type::send_tan_list' unsuccessfully" <<
      "with return value 1." << endl;
    unlock_cerr_mutex();
    submit_mysql_query("unlock_tables", result, 0, 0, 0);
    if (result) mysql_free_result(result);
    ++errors_occurred;
    return 1;
  } /* if (status ≠ 0) */
#endif DEBUG_COMPILE
  else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::send_tan_list':" << endl <<
      "'generate_tans' succeeded, returning 0." << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (TAN_ctr < 100) */

```

122.

```

⟨ Scan_Parse_Parameter_Type :: send_tan_list definition 115 ⟩ +≡
sql_strm.str("");
sql_strm << "update_TANs_set_user_id=-1 where user_id=0 limit 50"; #
if DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "sql_strm.str()==" << sql_strm.str() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
status = submit_mysql_query(sql_strm.str(), result, 0, 0, &affected_rows);
if (status != 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'Scan_Parse_Parameter_Type::send_tan_list':"
        << endl << "'submit_mysql_query' failed, returning" << status << "."
        << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    if (result) {
        mysql_free_result(result);
        result = 0;
    }
    submit_mysql_query("unlock_tables", result, 0, 0, 0);
    if (result) mysql_free_result(result);
    ++errors_occurred;
    return 1;
} /* if (status != 0) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::send_tan_list':"
            << endl << "'submit_mysql_query' succeeded, returning 0."
            << endl << "'affected_rows'=="
            << affected_rows << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
if (affected_rows == 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'Scan_Parse_Parameter_Type::send_tan_list':"
        << endl << "'affected_rows'==0."
        << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    if (result) {
        mysql_free_result(result);
        result = 0;
    }
    submit_mysql_query("unlock_tables", result, 0, 0, 0);
    if (result) mysql_free_result(result);
    ++errors_occurred;
    return 1;
} /* if (status != 0) */

```

```
else if (affected_rows != 50) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "WARNING! In 'Scan_Parse_Parameter_Type::send_tan_list':"
        endl << "(" << affected_rows << ") != 50" << endl << "Will try to continue." << endl;
    unlock_cerr_mutex();
} /* else if (affected_rows != 50) */
#endif /* DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::send_tan_list':"
        endl <<
        "'affected_rows' == 50. (This is how it should be.)" << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
mysql_free_result(result);
result = 0;
```

123.

```

⟨ Scan_Parse_Parameter_Type :: send_tan_list definition 115 ⟩ +≡
  stringstream tan_strm;
  tan_strm << "echo-e\TANListfor" << username << "\n\n";
  /* !! TODO: LDF 2012.07.19. Add more information, datestamp, etc. */
  sql_strm.str("");
  sql_strm << "select TAN from TANs where user_id=-1"; #
  if DEBUG_COMPILE
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "sql_strm.str() == " << sql_strm.str() << endl;
      unlock_cerr_mutex();
    } /* if (DEBUG) */
  #endif /* DEBUG_COMPILE */
  status = submit_mysql_query(sql_strm.str(), result, &row_ctr, &field_ctr, 0);
  if (status != 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'Scan_Parse_Parameter_Type::send_tan_list':"
        << endl << "'submit_mysql_query' failed, returning " << status << "."
        << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    if (result) {
      mysql_free_result(result);
      result = 0;
    }
    submit_mysql_query("unlock_tables", result, 0, 0, 0);
    if (result) mysql_free_result(result);
    ++errors_occurred;
    return 1;
  } /* if (status != 0) */
#if DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::send_tan_list':"
          << endl << "'submit_mysql_query' succeeded, returning 0."
          << endl << "'row_ctr' == "
          << row_ctr << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

124.

```

⟨ Scan_Parse_Parameter_Type::send_tan_list definition 115 ⟩ +≡
  for (int i = 0; i < row_ctr; ++i) {
    if ((curr_row = mysql_fetch_row(result)) == 0) {
      lock_cerr_mutex();
      cerr << thread_ctr_str << "ERROR! In 'Scan_Parse_Parameter_Type::send_tan_list':"
      endl << "'mysql_fetch_row' failed:" << endl << mysql_error(mysql_ptr) << endl <<
      "Exiting 'Scan_Parse_Parameter_Type::send_tan_list' unsuccessfully"
      "with return value 1." << endl;
      unlock_cerr_mutex();
      mysql_free_result(result);
      submit_mysql_query("unlock_tables", result, 0, 0, 0);
      if (result) mysql_free_result(result);
      ++errors_occurred;
      return 1;
    } /* if (curr_row = mysql_fetch_row(result) == 0) */
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::send_tan_list':"
      << endl << "'mysql_fetch_row' succeeded."
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

125.

```

⟨ Scan_Parse_Parameter_Type::send_tan_list definition 115 ⟩ +≡
  if (curr_row[0] == 0 || strlen(curr_row[0]) == 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'Scan_Parse_Parameter_Type::send_tan_list':"
    endl << "'curr_row[0]' is NULL or empty." << endl <<
    "Exiting 'Scan_Parse_Parameter_Type::send_tan_list' unsuccessfully"
    "with return value 1." << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    submit_mysql_query("unlock_tables", result, 0, 0, 0);
    if (result) mysql_free_result(result);
    ++errors_occurred;
    return 1;
  } /* if (curr_row[0] == 0 || strlen(curr_row[0]) == 0) */
  tan_strm << curr_row[0] << "\n"; } /* for */

```

126.

Log

[LDF 2013.09.20.] Replaced “hard-wired” GPG key IDs with `gpg_key_id` for the signing key and `Scan_Parse_Parameters` for the key used for encryption.

```

< Scan_Parse_Parameter_Type :: send_tan_list definition 115 > +≡
    mysql_free_result(result);
    result = 0;
    tan_strm << "\"\u0020--gpgu--encrypt\u0020--armor\u0020--homedir\" << gpg_homedir << "\u0020-r\u0020" << public_key_id <<
        "\u0020--sign\u0020-u\u0020" << gpg_key_id;
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "tan_strm.str()\u003d" << tan_strm.str() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
FILE *fp = fopen(tan_strm.str().c_str(), "r");
if (fp == 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR!\u0020In\u0020`Scan_Parse_Parameter_Type::send_tan_list':\u0020" <<
        endl << "'popen'\u0020failed,\u0020returning\u00200.\u0020" << endl << "Failed\u0020to\u0020encrypt\u0020TAN\u0020list.\u0020" <<
        endl << "Exiting\u0020`Scan_Parse_Parameter_Type::send_tan_list'\u0020unsuccessfully\u0020" <<
        "with\u0020return\u0020value\u00201.\u0020" << endl;
    unlock_cerr_mutex();
    submit_mysql_query("unlock_tables", result, 0, 0, 0);
    if (result) mysql_free_result(result);
    ++errors_occurred;
    return 1;
} /* if (fp == 0) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "In\u0020`Scan_Parse_Parameter_Type::send_tan_list':\u0020" << endl <<
            "'popen'\u0020succeeded.\u0020Encrypted\u0020TAN\u0020list\u0020successfully.\u0020" << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
ofstream out_strm; /* !! TODO: LDF 2012.07.19. Create temporary file, perhaps. Add code for
                     sending using mailx or sendmail. */
unlink("/tmp/abc");
out_strm.open("/tmp/abc");
char buffer[2048];
status = 0;
do {
    memset(buffer, 0, 2048);
    status = fread(buffer, 1, 2047, fp);
    if (status > 0) out_strm << buffer;
} while (status == 2047);

```

```
out_strm.close();
pclose(fp);
fp = 0;
```

127.

```

⟨ Scan_Parse_Parameter_Type :: send_tan_list definition 115 ⟩ +≡
sql_strm.str("");
sql_strm << "update_TANs_set_user_id_=_" << user_id << "_where_user_id=_-1"; #
if DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "sql_strm.str() == " << sql_strm.str() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
status = submit_mysql_query(sql_strm.str(), result, 0, 0, &affected_rows);
if (status != 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'Scan_Parse_Parameter_Type::send_tan_list':"
        << endl << "'submit_mysql_query' failed, returning " << status << "."
        << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    if (result) {
        mysql_free_result(result);
        result = 0;
    }
    submit_mysql_query("unlock_tables", result, 0, 0, 0);
    if (result) mysql_free_result(result);
    ++errors_occurred;
    return 1;
} /* if (status != 0) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::send_tan_list':"
            << endl << "'submit_mysql_query' succeeded, returning 0."
            << endl << "'affected_rows' == "
            << affected_rows << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
if (affected_rows == 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'Scan_Parse_Parameter_Type::send_tan_list':"
        << endl << "'affected_rows' == 0."
        << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    if (result) {
        mysql_free_result(result);
        result = 0;
    }
    submit_mysql_query("unlock_tables", result, 0, 0, 0);
    if (result) mysql_free_result(result);
    ++errors_occurred;
    return 1;
} /* if (status != 0) */

```

```

else if (affected_rows != 50) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "WARNING! In 'Scan_Parse_Parameter_Type::send_tan_list':"
        endl << "(" <= " << affected_rows << ") != 50" << endl << "Will try to continue." << endl;
    unlock_cerr_mutex();
} /* else if (affected_rows != 50) */
#endif /* DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::send_tan_list':"
        endl << "'affected_rows' == 50. (This is how it should be.)" << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
mysql_free_result(result);
result = 0;

```

128. Unlock TANs table. [LDF 2012.07.19.]

```

<Scan_Parse_Parameter_Type::send_tan_list definition 115> +≡
status = submit_mysql_query("unlock_tables", result, 0, 0, 0);
if (status != 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'Scan_Parse_Parameter_Type::send_tan_list':"
        endl << "'submit_mysql_query' failed, returning" << status <<
        "." << endl << "Failed to unlock 'TANs' table." << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    ++errors_occurred;
    return 1;
} /* if (status != 0) */
#endif /* DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::send_tan_list':"
        endl << "'submit_mysql_query' succeeded, returning 0." << endl <<
        "Unlocked 'TANs' table successfully." << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
mysql_free_result(result);
result = 0;

```

129.

```
<Scan_Parse_Parameter_Type::send_tan_list definition 115> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "Exiting `Scan_Parse_Parameter_Type::send_tan_list' successfully"
            << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; } /* End of send_tan_list definition */
```

130. Set *expires*. [LDF 2012.07.23.]

Log

[LDF 2012.07.23.] Added this function.

```
<Scan_Parse_Parameter_Type function declarations 38> +≡
int set_expires(time_t seconds, stringstream *out_strm = 0);
```

131.

```
<Scan_Parse_Parameter_Type::set_expires definition 131> ≡
int Scan_Parse_Parameter_Type::set_expires(time_t seconds, stringstream *out_strm)
{
    expires = time(0) + seconds;
    return 0;
} /* End of Scan_Parse_Parameter_Type::set_expires definition */
```

This code is used in section 305.

132. Get *expires*. [LDF 2012.07.23.]

Log

[LDF 2012.07.23.] Added this function.

```
<Scan_Parse_Parameter_Type function declarations 38> +≡
time_t get_expires(char *str = 0, size_t str_size = 0, stringstream *out_strm = 0);
```

133.

```
<Scan_Parse_Parameter_Type::get_expires definition 133> ≡
time_t Scan_Parse_Parameter_Type::get_expires(char *str, size_t str_size, stringstream
                                              *out_strm)
{
    /* !! TODO: LDF 2012.07.23. Add error handling, perhaps more options. */
    if (str != 0) {
        struct tm tmp;
        struct tm *tmp_ptr = &tmp;
        localtime_r(&expires, tmp_ptr);
        strftime(str, str_size, "%Y-%m-%d %H:%M:%S %Z", tmp_ptr);
    }
    if (out_strm != 0) {
        *out_strm << expires;
    }
    return expires;
}    /* End of Scan_Parse_Parameter_Type::get_expires definition */
```

This code is used in section 305.

134. ls. [LDF 2012.07.13.]**Log**

[LDF 2012.07.13.] Added this function.
 [LDF 2012.07.20.] Moved the definition of this function from this file (**scprpmtp.web**) to **spptjnif.web**.
 [LDF 2012.09.18.] Restored this function declaration. Apparently, I had removed it at some time in the past.
 [LDF 2013.01.10.] Added optional arguments **string args = ""** and **bool do_response = true**.
 [LDF 2013.04.25.] Replaced optional argument **string args** with optional argument **Response_Type *response = 0**. Added optional argument **string filename_1 = ""**.

135.

```
<Scan_Parse_Parameter_Type function declarations 38> +≡
int ls(char *buffer_ptr, unsigned int buff_size, string *filename, Response_Type *response, string
      filename_1 = "", bool do_response = true);
```

136.

```
< Scan_Parse_Parameter_Type::ls definition 136 > ≡
int Scan_Parse_Parameter_Type::ls(char *buffer_ptr, unsigned int buff_size, string
    *filename, Response_Type *response, string filename_1, bool do_response){ bool
    DEBUG = false; /* true */
set_debug_level(DEBUG, 0, 0);
int status = 0;
stringstream temp_strm;
temp_strm << "[Thread_" << thread_ctr << "] ";
string thread_ctr_str = temp_strm.str();
temp_strm.str("");
*filename = "";
/* Will be set if this function creates a temporary file. It should already be empty when passed to
   this function, but setting it to empty string here just to be sure. [LDF 2012.11.28.] */
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "Entering 'Scan_Parse_Parameter_Type::ls'." << endl;
    if (response != 0) {
        cerr << "response->string_vector.size() == " << response->string_vector.size() << endl;
        if (response->string_vector.size() > 0)
            for (vector<string>::iterator iter = response->string_vector.begin();
                 iter != response->string_vector.end(); ++iter) cerr << "*iter == " << *iter << endl;
        unlock_cerr_mutex();
    } /* if (response != 0) */
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

See also sections 137, 138, 139, 140, 141, and 142.

This code is used in section 305.

137. `char buffer[]` is used for the output of the `ils` command, i.e., either the icommand `ils` or the function `lsUtil`, depending on what interface to the iRODS server is used. The size, i.e., 1048576 is equal to $2^{20} = 1\text{MB}$, so it should be large enough.

If more bytes are returned than the `buff_size` argument (which should normally be the global constant `BUFFER_SIZE` = 2048 as of this date), the output will be dumped to a temporary file and the filename will be stored in `*filename`. [LDF 2012.11.28.]

```
< Scan_Parse_Parameter_Type::ls definition 136 > +≡
char buffer[1048576];
memset(buffer, 0, 1048576);
```

138.

Log

[LDF 2013.01.11.] Removed code for decrypting the scrambled iRODS password. It's not needed, and it shouldn't work anyway, because in this case, `ils` would require the unscrambled password.

[LDF 2013.05.23.] Removed “`echo $?`” command from the end of the string passed to `popen`. I wasn't testing the exit status of the shell command, anyway, and the trailing 0 was sent to the client and displayed, which was confusing.

```
( Scan_Parse_Parameter_Type::ls definition 136 ) +≡
    errno = 0;
    temp_strm.str("");
    temp_strm << "env_irodsEnvFile=" << irods_env_filename << " " << "ils";
    if (response ≠ 0) {
        if (!response→flags.empty()) temp_strm << " " << response→flags;
        for (vector<string>::iterator iter = response→string_vector.begin();
            iter ≠ response→string_vector.end(); ++iter) temp_strm << " " << *iter;
    } /* if (response ≠ 0) */
    if (!filename_1.empty()) temp_strm << " " << filename_1;
    temp_strm << "\2>&1";
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::ls':" << endl <<
            "temp_strm.str() == " << temp_strm.str() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
FILE *fp = popen(temp_strm.str().c_str(), "r");
if (fp == 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'Scan_Parse_Parameter_Type::ls':" << endl <<
        "'popen' failed, returning NULL. Can't list directory contents." << endl;
    if (errno ≠ 0) cerr << strerror(errno);
    cerr << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    if (do_response) strcpy(buffer_ptr, "LS_RESPONSE_1\\\"Server-side error:\\\"'popen' failed\\\"");
    ++errors_occurred;
    return 1;
} /* if (fp == 0) */
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::ls':" << endl <<
                "'popen' succeeded. 'fp' is non-null." << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

139.

```
< Scan_Parse_Parameter_Type::ls definition 136 > +≡
    memset(buffer, 0, 1048576);
    status = fread(buffer, 1, 1048576, fp);
    if (status == 0) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "ERROR! In 'Scan_Parse_Parameter_Type::ls':"
            << endl <<
            "'fread' failed, returning 0. Can't list directory contents."
            << endl <<
            "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
    if (do_response) strcpy(buffer_ptr, "LS RESPONSE 2 \"Server-side error: 'fread' failed\"");
    pclose(fp);
    ++errors_occurred;
    return 1;
} /* if (status == 0) */
```

140.

```
< Scan_Parse_Parameter_Type::ls definition 136 > +≡
else
    if (status == 1048576) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "ERROR! In 'Scan_Parse_Parameter_Type::ls':"
            << endl <<
            "'fread' returned " << status << " == 1048576."
            << endl <<
            "Output from pipe exceeds maximum length (i.e., 1048576 - 1 == 1048575)."
            << endl <<
            "Can't list files." << endl <<
            "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
    if (do_response) strcpy(buffer_ptr,
        "LS RESPONSE 3 \"Server-side error: 'ls' output too long (1048576 bytes)\"");
    pclose(fp);
    ++errors_occurred;
    return 1;
} /* if (status == 1048576) */
```

141. Success. [LDF 2012.11.28.]

```
<Scan_Parse_Parameter_Type::ls definition 136> +≡
else {
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::ls':" << endl << "'buffer'" << endl
            << buffer << endl << "'strlen(buffer)' == " << strlen(buffer) << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    if (do_response) strcpy(buffer_ptr, "LS_RESPONSE_0");
    int temp_ctr = strlen(buffer) + strlen(buffer_ptr) + 2;
    if (temp_ctr < buff_size - 1) {
#endif DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::ls':" << endl <<
                "'buffer' short enough. Not dumping to file." << endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        strcat(buffer_ptr, "\\");
        strcat(buffer_ptr, buffer);
        strcat(buffer_ptr, "\\");
#endif DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::ls':" << endl <<
                "buffer_ptr == " << buffer_ptr << endl <<
                "Exiting function successfully with return value 0." << endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        pclose(fp);
        return 0;
    } /* if */
}
```

142. !! TODO: In this case, *buffer* should be dumped to a file, which should be sent to the client, which must call **Scan_Parse_Parameter_Type**::*receive_file* to receive it. However, it is very unlikely that the *lls* command (or its equivalent) will produce that many bytes. Currently, *buff_size* should normally be **BUFFER_SIZE**, which is 2048 as of this date, but I may increase it. [LDF 2012.11.28.]

```
< Scan_Parse_Parameter_Type ::ls definition 136 > +≡
else {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::ls':"
    endl << "'buffer' too long. Not yet dumping to a file."
    "This may be implemented in the future." << endl <<
    "Exiting function unsuccessfully with return value 1." << endl;
unlock_cerr_mutex();
temp_strm.str("");
if (do_response) temp_strm << "LS RESPONSE 4";
temp_strm << "\"Response length exceeds maximum length:" << endl <<
    "Response length == " << temp_ctr << " bytes" << endl << "Maximum length == "
    buff_size - 1 << " bytes\"";
strcpy(buffer_ptr, temp_strm.str().c_str());
pclose(fp);
++errors_occurred;
return 1;
} /* else */
} /* else */
} /* End of Scan_Parse_Parameter_Type ::ls definition */
```

143. *pwd*. [LDF 2013.04.03.]

Log

[LDF 2013.04.03.] Added this function.

144.

```
< Scan_Parse_Parameter_Type function declarations 38 > +≡
int pwd(char *buffer_ptr, unsigned int buff_size, string args = "");
```

145.

```
<Scan_Parse_Parameter_Type::pwd definition 145> ≡
int Scan_Parse_Parameter_Type::pwd(char *buffer_ptr, unsigned int buff_size, string args){ bool
    DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status = 0;
    stringstream temp_strm;
    temp_strm << "[Thread_]" << thread_ctr << "] ";
    string thread_ctr_str = temp_strm.str();
    temp_strm.str("");
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "Entering 'Scan_Parse_Parameter_Type::pwd'." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    memset(buffer_ptr, 0, buff_size);
    char buffer[2048];
    /* Should be more than large enough for the name of the working directory. [LDF 2012.09.19.] */
}
```

See also sections 146, 147, 148, and 149.

This code is used in section 305.

146.

```

⟨ Scan_Parse_Parameter_Type::pwd definition 145 ⟩ +≡
    temp_strm << "env_irodsEnvFile=" << irods_env_filename << "ipwd";
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::pwd':"
            << endl <<
            "'temp_strm.str()' == "
            << temp_strm.str() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    errno = 0;
FILE *fp = popen(temp_strm.str().c_str(), "r");
memset(buffer, 0, 2048);
if (fp == 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'Scan_Parse_Parameter_Type::pwd':"
        << endl <<
        "'popen' failed, returning NULL. Can't show working directory."
        << endl;
    if (errno != 0) perror("popen_error");
    cerr << "Exiting function unsuccessfully with return value 1."
        << endl;
    unlock_cerr_mutex();
    strcpy(buffer_ptr, "PWD_FAILED_1");
    ++errors_occurred;
    return 1;
} /* if (fp == 0) */
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::pwd':"
                << endl <<
                "'popen' succeeded. 'fp' is non-null."
                << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    status = fread(buffer, 1, 2048, fp);
    if (status == 0) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "ERROR! In 'Scan_Parse_Parameter_Type::pwd':"
            << endl <<
            "'fread' failed, returning 0. Can't show working directory."
            << endl <<
            "Exiting function unsuccessfully with return value 1."
            << endl;
        unlock_cerr_mutex();
        strcpy(buffer_ptr, "PWD_FAILED_1");
        ++errors_occurred;
        return 1;
} /* if (status == 0) */

```

147.

```
< Scan_Parse_Parameter_Type :: pwd definition 145 > +≡
else
  if (status ≡ 2048) {
    lock_cerr_mutex();
    cerr ≪ thread_ctr_str ≪ "ERROR! In 'Scan_Parse_Parameter_Type::pwd':"
    endl ≪ "'fread' returned" ≪ status ≪ " == 2048." ≪
    endl ≪ "Output from pipe exceeds maximum length." ≪
    endl ≪ "Can't show working directory." ≪ endl ≪
    "Exiting function unsuccessfully with return value 1." ≪ endl;
    unlock_cerr_mutex();
    pclose(fp);
    fp = 0;
    strcpy(buffer_ptr, "PWD FAILED 1");
    ++errors_occurred;
    return 1;
} /* if (status ≡ 2048) */
```

148.

```
< Scan_Parse_Parameter_Type :: pwd definition 145 > +≡
if (buffer[status - 1] ≡ '\n') {
  buffer[status - 1] = '\0'; /* Remove trailing newline, if present. [LDF 2013.04.03.] */
}
temp_strm.str("");
temp_strm ≪ "PWD RESPONSE" ≪ buffer ≪ "\n";
if (temp_strm.str().size() ≥ buff_size) {
  lock_cerr_mutex();
  cerr ≪ thread_ctr_str ≪ "ERROR! In 'Scan_Parse_Parameter_Type::pwd':"
  endl ≪ "Response string length == 'temp_strm.str().size()' == "
  temp_strm.str().size() ≪ endl ≪ "> 'buff_size' == " ≪ buff_size ≪
  "." ≪ endl ≪ "Can't show working directory." ≪ endl ≪
  "Exiting function unsuccessfully with return value 1." ≪ endl;
  unlock_cerr_mutex();
  strcpy(buffer_ptr, "PWD FAILED 1");
  pclose(fp);
  fp = 0;
  ++errors_occurred;
  return 1;
} /* if (temp_strm.str().size() ≥ buff_size) */
```

149.

```
< Scan_Parse_Parameter_Type ::pwd definition 145 > +≡
else {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::pwd':" << endl <<
            "'buffer'" <=> buffer << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
strcpy(buffer_ptr,temp_strm.str().c_str());
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::pwd':" << endl <<
            "'buffer_ptr'" <=> buffer_ptr << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* else */
pclose(fp);
fp = 0;
return 0; } /* End of Scan_Parse_Parameter_Type ::pwd definition */
```

150. put. [LDF 2012.09.27.]

Log

[LDF 2012.09.27.] Added this function.

[LDF 2013.01.09.] Added code for creating or updating rows in the `Irods_Objects` database table and creating rows in the `Irods_Objects_Handles` table, if a PID is generated.

[LDF 2013.04.04.] Changed the way `response.local_filename` is handled. This is the path of the iRODS object to be “put”. Now, if it’s not an absolute path, `irods_current_dir` is inserted at the front. `response.local_filename` can contain directories, e.g., a path like `subdir_1/subdir_2/abc.txt` can also be specified.

```
< Scan_Parse_Parameter_Type function declarations 38 > +≡
int put(Response_Type &response);
```

151.

```

⟨ Scan_Parse_Parameter_Type::put definition 151 ⟩ ≡
int Scan_Parse_Parameter_Type::put(Response_Type &response){ bool DEBUG = false;
    /* true */
    set_debug_level(DEBUG, 0, 0);
    stringstream temp_strm;
    int status = 0;
    Response_Type new_response;
    new_response.type = Response_Type::COMMAND_ONLY_TYPE;
    Irods_Object_Type curr_irods_object;
    Irods_AVU_Type curr_avu;
    unsigned long int curr_metadata_handle_id;
    vector<unsigned long int> metadata_handle_id_vector;
    vector<Handle_Type> handle_vector;
    vector<Handle_Type> metadata_handle_vector;
    MYSQL_RES * result = 0;
    MYSQL_ROW curr_row;
    unsigned int row_ctr;
    unsigned int field_ctr;
    long int affected_rows;
    stringstream sql_strm;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] Entering " 'Scan_Parse_Parameter_Type::put' . " <<
            endl;
        cerr << "'irods_current_dir'" << irods_current_dir << endl <<
            "'response.remote_filename'" << response.remote_filename << endl <<
            "'response.local_filename'" << response.local_filename << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

See also sections 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, and 199.

This code is used in section 305.

152. Check *response.remote_filename* to see if it contains a filename. If it doesn't, exit unsuccessfully. [LDF 2012.11.23.]

Log

[LDF 2012.11.23.] Added this section.

```
<Scan_Parse_Parameter_Type::put definition 151> +≡
    string remote_plain_filename = response.remote_filename;
    size_t pos = remote_plain_filename.find_last_of("/");
    if (pos ≡ string::npos) { /* response.remote_filename doesn't contain any directories */
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] In 'Scan_Parse_Parameter_Type::put'" <<
            endl << "'response.remote_filename' == " << response.remote_filename <<
            "\ndoesn't contain any directories." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (pos ≡ string::npos) */
```

153. This code should never be reached, because this case is caught in *yyparse*. [LDF 2012.11.23.]

```
<Scan_Parse_Parameter_Type::put definition 151> +≡
else
    if (remote_plain_filename[remote_plain_filename.size() - 1] ≡ '/') {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] ERROR! In 'Scan_Parse_Parameter_Type::put'" <<
            endl << "'response.remote_filename' == " << response.remote_filename <<
            "\nis a directory and cannot be 'put'." << endl <<
            "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        ++errors_occurred;
        temp_strm.str("");
        temp_strm << "PUT_RESPONSE\\"" << response.local_filename << "\\1\\Input_error:\\" <<
            "Client-side_filename_is_a_directory:\\" << response.remote_filename << "\"";
        new_response.command = temp_strm.str();
        response_deque.push_back(new_response);
        ++errors_occurred;
        return 1;
    } /* else if (remote_plain_filename[remote_plain_filename.size() - 1] ≡ '/') */
```

154.

```
<Scan_Parse_Parameter_Type::put definition 151> +≡
else {
    remote_plain_filename.erase(0, pos + 1);
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] In " 'Scan_Parse_Parameter_Type::put' : " << endl <<
            "'remote_plain_filename'" <=> remote_plain_filename << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* else */
```

155. Check *response.local_filename* to see if it's a directory. If it is, append the filename portion of *response.remote_filename* to it. [LDF 2012.11.23.]

Log

[LDF 2012.11.23.] Added this section.

```
<Scan_Parse_Parameter_Type::put definition 151> +≡
if (response.local_filename[response.local_filename.size() - 1] == '/') {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] In " 'Scan_Parse_Parameter_Type::put' : " << endl <<
            "'response.local_filename' is a directory: " <=> response.local_filename << endl <<
            "Will append filename part of 'response.remote_filename' to it." << endl <<
            "'response.remote_filename'" <=> response.remote_filename << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    response.local_filename += remote_plain_filename;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] In " 'Scan_Parse_Parameter_Type::put' : " << endl <<
            "After appending 'remote_plain_filename', " <=> "'response.local_filename'" <=> "
            response.local_filename << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (response.local_filename[response.local_filename.size() - 1] == '/') */
```

156.

Log

[LDF 2013.04.04.] Added this section.

```
<Scan_Parse_Parameter_Type::put definition 151> +≡
if (response.local_filename[0] ≠ '/') {
    response.local_filename.insert(0, "/");
    response.local_filename.insert(0, irods_current_dir);
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "After inserting 'irods_current_dir': " << "response.local_filename==" <<
        response.local_filename << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (response.local_filename[0] ≠ '/') */
```

157.

Log

[LDF 2013.01.03.] Now using *access* instead of *fopen* to test for existence of file.

```
<Scan_Parse_Parameter_Type::put definition 151> +≡
errno = 0;
status = access(response.temporary_filename.c_str(), F_OK);
if (status == -1) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] In" << "'Scan_Parse_Parameter_Type::put':" << endl << "'access' returned -1:" << endl << strerror(errno) << endl <<
        "Can't put \"file.\" Will send failure notice to client and continue." << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    temp_strm.str("");
    temp_strm << "PUT RESPONSE" << response.local_filename << "\\" << "Server error:" <<
        "File" << response.temporary_filename << "\\" << "doesn't exist";
    new_response.command = temp_strm.str();
    response_deque.push_back(new_response);
    ++errors_occurred;
    return 1;
} /* if (status == -1) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] In" << "'Scan_Parse_Parameter_Type::put':" << endl <<
            "'access' succeeded, returning 0. File" << "\\" << response.temporary_filename <<
            "\ exists." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

158.

```

⟨ Scan_Parse_Parameter_Type::put definition 151 ⟩ +≡
temp_strm.str("");
/* !! TODO: response.flags may have leading or trailing spaces. This bothers me a little, but I'm not
sure whether it's worth doing anything about it. Check how it's set and where it's used. */
temp_strm << "env_irodsEnvFile=" << irods_env_filename << "input" << response.flags << " "
<< response.temporary_filename << " " << response.local_filename << "2>&1;echo$?";
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "temp_strm.str() == " << temp_strm.str() << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
errno = 0;
FILE *fp = popen(temp_strm.str().c_str(), "r");
if (fp == 0) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] ERROR! In 'Scan_Parse_Parameter_Type::put':"
        "'popen' failed, returning NULL: " << strerror(errno) << endl <<
        "Can't \"put\" file. Will send failure notice to client." << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    temp_strm.str("");
    temp_strm << "PUT RESPONSE " << response.local_filename << "\1\Server\error: "
        "Failed to execute 'input' command";
    new_response.command = temp_strm.str();
    response_deque.push_back(new_response);
    ++errors_occurred;
    ++errors_occurred;
    return 1;
} /* if (fp == 0) */
else /* fp != 0 */
{
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] In 'Scan_Parse_Parameter_Type::put':"
        "'popen' succeeded." << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
char buffer[1024];
memset(buffer, 0, 1024);
status = fread(buffer, 1, 1024, fp);
if (status == 0) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] ERROR! " << "In 'Scan_Parse_Parameter_Type::put':"
        "'fread' failed, returning 0. Failed to read from pipe." << endl <<
        "Failed to \"put\" file. Will send failure notice" << "to client." << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
}

```

```

unlock_cerr_mutex();
++errors_occurred;
temp_strm.str("");
temp_strm << "PUT\u005cRESPONSE\u005c" << response.local_filename << "\u005cu1\u005cServer\u005cerror:\u005cu" <<
    "Failed\u005cto\u005cread\u005coutput\u005band/or\u005cexit\u005cstatus\u005cfrom\u005cu" << "'input'\u005ccommand.\u005cu";
new_response.command = temp_strm.str();
response_deque.push_back(new_response);
pclose(fp);
++errors_occurred;
return 1;
} /* if (status == 0) */
else if (status == 1024) {
    lock_cerr_mutex();
    cerr << "[Thread\u005cu" << thread_ctr << "] \u005cERROR!\u005cu" << "In\u005c'Scan_Parse_Parameter_Type::put':\u005cu" <<
        "'fread'\u005ctransferred 1024:\u005cu Pipe\u005coutput\u005cexceeds\u005cmaximum\u005clength." << endl <<
        "Failed\u005cto\u005c\"put\"\u005cfile.\u005cu" << "Will\u005csend\u005cfailure\u005cnotice\u005cto\u005cclient." << endl <<
        "Exiting\u005cfunction\u005cunsuccessfully\u005cwith\u005creturn\u005cvalue\u005cu 1." << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    temp_strm.str("");
    temp_strm << "PUT\u005cRESPONSE\u005c" << response.local_filename << "\u005cu1\u005cServer\u005cerror:\u005cu" <<
        "Output\u005cfrom\u005cu 'input'\u005ccommand\u005cexceeds\u005cmaximum\u005clength\u005cu" << "(1023\u005ccharacters).\u005cu";
    new_response.command = temp_strm.str();
    response_deque.push_back(new_response);
    pclose(fp);
    ++errors_occurred;
    return 1;
} /* else if (status == 1024) */
else /* fread succeeded and read valid number of characters */
{
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread\u005cu" << thread_ctr << "] \u005c" << "In\u005c'Scan_Parse_Parameter_Type::put':\u005cu" <<
            "'popen'\u005csucceeded." << endl << "'buffer'\u005cu==\u005cu" << buffer << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

159.

```

⟨ Scan_Parse_Parameter_Type::put definition 151 ⟩ +≡
    /* Get the return value of iput command. */
    char *ret_val_ptr = buffer + strlen(buffer) - 1;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "ret_val_ptr==\" << ret_val_ptr << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    --ret_val_ptr;
    while (isdigit(ret_val_ptr[0])) --ret_val_ptr;
    if (¬(ret_val_ptr[0] ≡ '-' ∨ ret_val_ptr[0] ≡ '+')) ++ret_val_ptr;
    int ret_val;
    sscanf(ret_val_ptr, "%d", &ret_val);
    ret_val_ptr[0] = '\0'; /* Remove exit status from end of buffer */
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::put':"
            << endl << "ret_val_ptr==\" << ret_val_ptr << endl << "ret_val==\" << ret_val << endl <<
            "'buffer'" << buffer << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

160.

```

⟨ Scan_Parse_Parameter_Type::put definition 151 ⟩ +≡
    if (ret_val ≡ 0) {
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "[Thread" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::put':"
                << endl << "'iinput' succeeded." << endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        temp_strm.str("");
        temp_strm << "PUT_RESPONSE\" " << response.local_filename << "\\" \0 \\
            << "'iinput' command succeeded, returning" << ret_val;
        if (strlen(buffer) > 0) temp_strm << ":" << buffer << "\"";
        else temp_strm << "\"";
        new_response.command = temp_strm.str();
        response_deque.push_back(new_response);
        pclose(fp);
        fp = 0;
    } /* if (ret_val ≡ 0) */

```

161.

```
<Scan_Parse_Parameter_Type::put definition 151> +≡
else /* ret_val ≠ 0 */
{
    lock_cerr_mutex();
    cerr << "[Thread_" << thread_ctr << "]_ERROR!" << "In_`Scan_Parse_Parameter_Type::put':"
        "'input'_failed:" << endl << "'buffer'_==" << buffer <<
        endl << "Will_send_failure_message_to_client." << endl <<
        "Exiting_function_unsuccessfully_with_return_value_1." << endl;
    unlock_cerr_mutex();
    temp_strm.str("");
    temp_strm << "PUT_RESPONSE\" << response.local_filename << "\"1\"Server_error:" <<
        "'input'_command_failed,_returning" << ret_val;
if (strlen(buffer) > 0) temp_strm << ":" << endl << buffer << "\"";
else temp_strm << "\"";
new_response.command = temp_strm.str();
response_deque.push_back(new_response);
++errors_occurred;
pclose(fp);
++errors_occurred;
return 1;
} /* else (ret_val ≠ 0) */
} /* else. fread succeeded and read valid number of characters */
} /* else (fp ≠ 0). popen succeeded */
```

162. *irods_object_filename* is the filename that should be stored in the `Irods_Objects` table. If *response.local_filename* contains at least one directory, *irods_object_filename* is set to *response.local_filename*. Otherwise, i.e., if *response.local_filename* is a plain filename with no directories, *irods_object_filename* is set to *irods_homedir* followed by a slash, followed by *response.local_filename*. [LDF 2013.01.07.] [LDF 2013.01.09.]

```
<Scan_Parse_Parameter_Type::put definition 151> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "irods_homedir==" << irods_homedir << endl << "response.local_filename==" <<
            response.local_filename << endl;
        response.show("response:");
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

163.

```

<Scan_Parse_Parameter_Type::put definition 151> +≡
status = curr_irods_object.set(user_id, response.local_filename);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ "[Thread" ≪ thread_ctr ≪ "] " ≪ "ERROR! In 'Scan_Parse_Parameter_Type::put':"
        endl ≪ "'Irods_Object_Type::set' failed, returning " ≪ status ≪ "."
        endl ≪ "Failed to set 'Irods_Object_Type' object for iRODS object "
        endl ≪ " " ≪ response.local_filename ≪ "." ≪ endl ≪
        "Exiting function unsuccessfully with return value 1." ≪ endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    temp_strm.str("");
    temp_strm ≪ "PUT_RESPONSE \""
    temp_strm ≪ response.local_filename ≪ "\" "
    temp_strm ≪ "1\" Server error: "
    temp_strm ≪ "C++ error: 'Irods_Object_Type' constructor failed";
    new_response.command = temp_strm.str();
    response_deque.push_back(new_response);
    ++errors_occurred;
    return 1;
} /* if (status ≠ 0) */
#if DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "[Thread" ≪ thread_ctr ≪ "] " ≪ "In 'Scan_Parse_Parameter_Type::put':"
        endl ≪ "'Irods_Object_Type::set' succeeded, returning 0."
        endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

164. Handle *pid_options*. [LDF 2012.09.28.]

A PID (a.k.a. “handle”) is only created if the `+pid` option is specified in the `put` command in the input. If the `+gen` option is also specified, the PID is generated. Otherwise, a particular PID may be specified. [LDF 2013.01.09.]

```

<Scan_Parse_Parameter_Type::put definition 151> +≡
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread_" << thread_ctr << "] " << "In " 'Scan_Parse_Parameter_Type::put' : "
        endl << "response.pid_options" == oct << response.pid_options ==
        dec << endl << "'response.pid_str'" == response.pid_str << endl <<
        "'response.pid_prefix_str'" == response.pid_prefix_str << endl <<
        "'response.pid_suffix_str'" == response.pid_suffix_str << endl <<
        "'response.pid_institute_str'" == response.pid_institute_str << endl;
    unlock_cerr_mutex();
}
#endif /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

165.

```
<Scan_Parse_Parameter_Type::put definition 151> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread_"
            << thread_ctr << "]_In_`Scan_Parse_Parameter_Type::put':"
            << endl;
        if (response.pid_options & 1U & !response.pid_str.empty()) {
            cerr << "response.pid_options&1U" << (response.pid_options & 1U) <<
                "and `response.pid_str' is" << "non-empty." << endl << " `response.pid_str' "
                << response.pid_str << endl;
        } /* if (response.pid_options & 1U) */
        else if (response.pid_options & 1U & response.pid_str.empty()) {
            cerr << "response.pid_options&1U" << (response.pid_options & 1U) <<
                "and `response.pid_str' is" << "empty." << endl << "Will generate a PID string."
                << endl;
        } /* if (response.pid_options & 1U) */
        else /* !(response.pid_options & 1U) */
        {
            cerr << "(response.pid_options&1U)" << (response.pid_options & 1U) << endl <<
                "Not generating a PID." << endl;
        } /* else !(response.pid_options & 1U) */
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    if (response.pid_options & 1U) { string prefix_str;
        if (!response.pid_prefix_str.empty()) prefix_str = response.pid_prefix_str;
```

166. If *response.pid_str* contains a slash, then the substring before it is a prefix. In this case, *prefix_str* should be empty and *not* be set to *default_handle_prefix* before passing it to *generate_pids*. Otherwise, if *prefix_str* and the prefix in *response.pid_str* differ, this will cause an error in *generate_pids*, which is not what we want. That is, it should be possible to choose a non-default prefix by including it at the front of *response.pid_str*. This is an alternative to using the +pid option. [LDF 2012.10.08.]

```
<Scan_Parse_Parameter_Type::put definition 151> +≡
    else {
        pos = response.pid_str.find("/");
        if (pos == string::npos) prefix_str = default_handle_prefix;
    }
```

167. !! TODO: Add similar code for setting a **string** *institute_str*. Currently, the institute abbrev. is not used in the handle. [LDF 2012.10.08.]

```
<Scan_Parse_Parameter_Type::put definition 151> +≡
    status = generate_pids(mysql_ptr, prefix_str, response.pid_str, 0,      /* PID vector pointer */
    1,      /* Number of PIDS */
    &curr_irods_object.handle_id_vector, &curr_irods_object.handle_value_id_vector, true,
    /* standalone handle server */
    response.pid_institute_str, response.pid_suffix_str, &handle_vector, "",      /* fifo.pathname */
    user_id, username);
```

168.

Log

[LDF 2012.10.08.] Now sending return value of *generate_pids* as error code in *new_response*.

```
⟨ Scan_Parse_Parameter_Type::put definition 151 ⟩ +≡
if (status ≠ 0) {
    cerr << "[Thread" << thread_ctr << "] " << "ERROR! In 'Scan_Parse_Parameter_Type::put':"
    endl << "'generate_pids' failed, returning" << status << ". " << endl <<
    "Exiting function unsuccessfully with exit status 1." << endl;
    ++errors_occurred;
    temp_strm.str("");
    temp_strm << "PUT RESPONSE \" " << response.local_filename << "\" " << status <<
        "\Server_error:" << "Failed to generate PID\" ";
    new_response.command = temp_strm.str();
    response_deque.push_back(new_response);
    ++errors_occurred;
    return 1;
} /* if (status ≠ 0) */
```

169.

```

⟨ Scan_Parse_Parameter_Type::put definition 151 ⟩ +≡
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "[Thread" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::put':" << endl <
                "generate_pids' succeeded, returning 0." << endl << "'response.pid_str'" << endl <
                response.pid_str << endl << "'curr_irods_object.handle_id_vector.size()'" << endl <
                curr_irods_object.handle_id_vector.size() << endl;
            cerr << "'handle_vector.size()'" << endl << handle_vector.size() << endl;
            if (curr_irods_object.handle_id_vector.size() > 0) {
                cerr << "Showing 'curr_irods_object.handle_id_vector':" << endl;
                for (vector<unsigned long int>::const_iterator iter = curr_irods_object.handle_id_vector.begin();
                     iter != curr_irods_object.handle_id_vector.end(); ++iter) {
                    cerr << *iter << endl;
                }
                cerr << endl;
            } /* if */
            else cerr << "'curr_irods_object.handle_id_vector' is empty." << endl;
            if (curr_irods_object.handle_value_id_vector.size() > 0) {
                cerr << "Showing 'curr_irods_object.handle_value_id_vector':" << endl;
                for (vector<unsigned long int>::const_iterator iter =
                     curr_irods_object.handle_value_id_vector.begin(); iter != curr_irods_object.handle_value_id_vector.end(); ++iter) {
                    cerr << *iter << endl;
                }
                cerr << endl;
            } /* if */
            else cerr << "'curr_irods_object.handle_value_id_vector' is empty." << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
temp_strm.str("");
temp_strm << "PUT RESPONSE" << endl << response.local_filename << "\0" << "Success:" << endl <
    "Generated PID" << response.pid_str << endl;
new_response.command = temp_strm.str();
response_deque.push_back(new_response);

```

170.

Log

[LDF 2013.01.31.] Added this section.

```
⟨ Scan_Parse_Parameter_Type::put definition 151 ⟩ +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "response.local_filename=" << response.local_filename << endl <<
            "handle_vector.size()=" << handle_vector.size() << endl;
        handle_vector.back().show("handle_vector.back():");
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    if (handle_vector.size() > 0) {
```

171. *handle_vector* should only contain one element, so we use *back()* to access it. That is, there's no need to iterate through *handle_vector*. [LDF 2013.02.05.]

Log

[LDF 2013.07.03.] Added code for adding handle value with *type* ≡ "CREATOR".

```

⟨Scan_Parse_Parameter_Type::put definition 151⟩ +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        distinguished_name.show("distinguished_name:");
        cerr << "distinguished_name.output()<=>" << distinguished_name.output() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
vector<Handle_Value_Triple> hvt_vector;
hvt_vector.push_back(Handle_Value_Triple(Handle_Value_Type::IRODS_OBJECT_INDEX,
    "IRODS_OBJECT", response.local_filename));
hvt_vector.push_back(Handle_Value_Triple(Handle_Value_Type::CREATOR_INDEX, "CREATOR",
    distinguished_name.output()));
status = handle_vector.back().add_values(mysql_ptr, hvt_vector, user_id);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "]<< "ERROR! In 'Scan_Parse_Parameter_Type::\n
        put':<< "'Handle_Type::add_values' failed, returning" <<
        status << "." << endl << "Failed to add handle values." <<
        endl << "Exiting 'Scan_Parse_Parameter_Type::put'" <<
        "unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    temp_strm.str("");
    ++errors_occurred;
    temp_strm << "PUT RESPONSE<< " << response.local_filename << "\n" << "1\"Server error:<<
        "Failed to add handle values\"";
    new_response.command = temp_strm.str();
    response_deque.push_back(new_response);
    ++errors_occurred;
    return 1;
} /* if (status ≠ 0) */

```

172.

```
<Scan_Parse_Parameter_Type::put definition 151> +≡
else /* status ≡ 0 */
{
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] " << "In `Scan_Parse_Parameter_Type::put':"
        "Handle_Type::add_values' succeeded, returning 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    temp_strm.str("");
    temp_strm << "PUT RESPONSE"
    << response.local_filename << "\"
    "0\"Added handle values with type == 'IRODS_OBJECT' and"
    "type == 'CREATOR_INDEX' successfully\"";
    new_response.command = temp_strm.str();
    response_deque.push_back(new_response);
}
/* else (status ≡ 0) */
```

173.

```

⟨ Scan_Parse_Parameter_Type ::put definition 151 ⟩ +≡
    sql_strm.str("");
    if (result) {
        mysql_free_result(result);
        result = 0;
    }
    sql_strm << "select distinct handle_id from handlesystem_standalone.handles" <<
        "where type='IRODS_OBJECT_REF'" << "and data=''" << response.local_filename << "";
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "sql_strm.str() == " << sql_strm.str() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    status = submit_mysql_query(sql_strm.str().c_str(), result, &row_ctr, &field_ctr);
    if (status != 0) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] ERROR!" << "In 'Scan_Parse_Parameter_Type::put':"
            << endl << "'Scan_Parse_Parameter_Type::submit_mysql_query' failed, returning"
            << status << "." << endl << mysql_error(mysql_ptr) << endl <<
            "Failed to test for existing metadata handles." << endl <<
            "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
    }
    if (result) {
        mysql_free_result(result);
    }
    temp_strm.str("");
    ++errors_occurred;
    temp_strm << "PUT_RESPONSE\\"" << response.local_filename << "\"\"
        << "1\"Server_error:\\" <<
        "Failed to test for existing metadata handles\"";
    new_response.command = temp_strm.str();
    response_deque.push_back(new_response);
    ++errors_occurred;
    return 1;
} /* if (status != 0) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "'submit_mysql_query' succeeded." << endl << "'row_ctr' == "
            << row_ctr << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

174.

```
< Scan_Parse_Parameter_Type ::put definition 151 > +≡
    if (row_ctr > 0) {
#if DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "'row_ctr' == " << row_ctr << "(>0)." << endl << row_ctr <<
            "metadata_handle(s)_exist(s)." << endl << "Will_create_cross-references." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

175.

```
< Scan_Parse_Parameter_Type ::put definition 151 > +≡
    for (int i = 0; i < row_ctr; ++i) {
        if ((curr_row = mysql_fetch_row(result)) == 0) {
            lock_cerr_mutex();
            cerr << "[Thread]" << thread_ctr << "] ERROR!" << "In 'Scan_Parse_Parameter_Type::put':" <<
                endl << "'mysql_fetch_row' failed:" << endl << mysql_error(mysql_ptr) << endl <<
                "Exiting function unsuccessfully with return value 1." << endl;
            unlock_cerr_mutex();
            mysql_free_result(result);
            temp_strm.str("");
            ++errors_occurred;
            temp_strm << "PUT_RESPONSE\"" << response.local_filename << "\" " << "1\"Server_error:" <<
                "MySQL_error:\u00d7Failed to fetch rows for existing" << "metadata_handles\"";
            new_response.command = temp_strm.str();
            response_deque.push_back(new_response);
            ++errors_occurred;
        }
        return 1;
    } /* if (curr_row = mysql_fetch_row(result) == 0) */
#endif DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "'mysql_fetch_row' succeeded." << endl << "'curr_row[0]' == " << curr_row[0] << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

176.

```

⟨ Scan_Parse_Parameter_Type ::put definition 151 ⟩ +≡
errno = 0;
curr_metadata_handle_id = strtoul(curr_row[0], 0, 10);
if (curr_metadata_handle_id == ULONG_MAX) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] ERROR!" << "In 'Scan_Parse_Parameter_Type::put':"
        << endl << "'strtoul' failed, returning ULONG_MAX:" << endl << strerror(errno) << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
unlock_cerr_mutex();
mysql_free_result(result);
temp_strm.str("");
++errors_occurred;
temp_strm << "PUT_RESPONSE\" << response.local_filename << "\" << "1\"Server_error:" <<
    "Failed to convert 'handle_id' for existing" << "metadata_handle\"";
new_response.command = temp_strm.str();
response_deque.push_back(new_response);
++errors_occurred;
return 1;
} /* if */
#endif /* DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "'strtoul' succeeded. curr_metadata_handle_id'" << curr_metadata_handle_id <<
        "." << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
metadata_handle_id_vector.push_back(curr_metadata_handle_id); } /* for */
} /* if (row_ctr > 0) */
#endif /* DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "'row_ctr'" << 0 << "No existing metadata handles for"
        "iRODS object 'response.local_filename'." << endl <<
        "No need to create cross-references." << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

177.

```
<Scan_Parse_Parameter_Type::put definition 151> +≡
    mysql_free_result(result);
    result = 0;
    sqlStrm.str("");
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "metadata_handle_id_vector.size() == " << metadata_handle_id_vector.size() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

178.

```
<Scan_Parse_Parameter_Type::put definition 151> +≡
    if (metadata_handle_id_vector.size() > 0) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "'metadata_handle_id_vector': " << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

179.

```

⟨ Scan_Parse_Parameter_Type::put definition 151 ⟩ +≡
status = fetch_handles_from_database(metadata_handle_id_vector, metadata_handle_vector);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << "[Thread_" << thread_ctr << "]_ERROR!" << "In_‘Scan_Parse_Parameter_Type::put’:"
        endl << "‘Scan_Parse_Parameter_Type::fetch_handles_from_database’_"
        "failed,_returning_" << status << "." << endl <<
        "Exiting_function_unsuccessfully_with_return_value_1." << endl;
    unlock_cerr_mutex();
    temp_strm.str("");
    ++errors_occurred;
    temp_strm << "PUT_RESPONSE\\"" << response.local_filename << "\\" << "1\Server_error:" <<
        "Failed_fetch_metadata_existing_metadata_handle(s)" << "from_database\"";
new_response.command = temp_strm.str();
response_deque.push_back(new_response);
++errors_occurred;
return 1;
} /* if (status ≠ 0) */
#endif /* DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread_" << thread_ctr << "]_In_‘Scan_Parse_Parameter_Type::put’:"
        endl << "‘Scan_Parse_Parameter_Type::fetch_handles_from_database’_"
        "succeeded,_returning_0." << endl << "metadata_handle_vector.size() == "
        metadata_handle_vector.size() << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
/* if (metadata_handle_id_vector.size() > 0) */
#endif /* DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "‘metadata_handle_id_vector’_is_empty._Not_creating_cross-references." << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
/* if (handle_vector.size() > 0) */

```

180. *handle_vector.size()* ≡ 0. This probably shouldn't ever happen. However, I am not treating it as an error case at present. [LDF 2013.01.31.]

```
<Scan_Parse_Parameter_Type::put definition 151> +≡
  else /* handle_vector.size() ≡ 0 */
  {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] " << "In `Scan_Parse_Parameter_Type::put': "
    "handle_vector.size() == 0. Not adding handle value." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* else (handle_vector.size() ≡ 0) */
```

181. If `metadata_handle_id_vector` was empty, then `metadata_handle_vector` will also be empty. However, I've put this conditional here in order not to have the code nested too deeply. [LDF 2013.02.07.]

```

⟨ Scan_Parse_Parameter_Type::put definition 151 ⟩ +≡
  if (metadata_handle_vector.size() > 0) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::put': " <<
      "'metadata_handle_vector.size()' == " << metadata_handle_vector.size() << "(>0)." <<
      endl << "Will add cross-references." << endl;
    handle_vector.back().show("handle_vector.back():");
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  for (vector<Handle_Type>::iterator iter = metadata_handle_vector.begin();
       iter != metadata_handle_vector.end(); ++iter) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    iter->show("*iter:");
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  status = handle_vector.back().add_value(mysql_ptr, Handle_Value_Type::DC_METADATA_PID_INDEX,
                                         "DC_METADATA_PID", iter->handle, user_id);
  if (status != 0) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] ERROR! " << "In 'Scan_Parse_Parameter_Type::put': "
    endl << "'Handle_Type::add_value'" << "failed, returning" << status << ". " << endl <<
    "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    temp_strm.str("");
    ++errors_occurred;
    temp_strm << "PUT_RESPONSE\\"" << response.local_filename << "\" " << "1\"Server_error:" <<
      "Failed to add cross-reference to existing" << "metadata_handle\"";
    new_response.command = temp_strm.str();
    response_deque.push_back(new_response);
    ++errors_occurred;
  }
  return 1;
} /* if (status != 0) */
else {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::put': " <<
      "'Handle_Type::add_value'succeeded, returning 0." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  temp_strm.str("");
  temp_strm << "PUT_RESPONSE\\"" << response.local_filename << "\" " << "0\"Success:" <<
    "Added cross-reference to existing" << "metadata_handle" << iter->handle << "\\"";
}

```

```

new_response.command = temp_strm.str();
response_deque.push_back(new_response);
} /* else */

```

182. Call `Handle_Value_Type::add_value` to add a cross-reference in the metadata handles. [LDF 2013.02.27.] ■

```

<Scan_Parse_Parameter_Type::put definition 151> +≡
status = iter->add_value(mysql_ptr, Handle_Value_Type::IRODS_OBJECT_PID_INDEX,
    "IRODS_OBJECT_PID", handle_vector.back().handle, user_id);
cerr << "WARNING! Testing: Not calling |Handle_Value_Type::add_value| ."
     << endl <<
     "Working on this. Setting 'status' to 0 and continuing." << endl;
if (status != 0) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] ERROR! " << "In 'Scan_Parse_Parameter_Type::put': "
        << endl << "'Handle_Value_Type::add_value'" << " failed, returning " << status << ". "
        << endl << "Exiting function unsuccessfully with return value 1." << endl;
unlock_cerr_mutex();
temp_strm.str("");
++errors_occurred;
temp_strm << "PUT_RESPONSE" << response.local_filename << "\" " << "1\"Server_error:" <<
    "Failed to add cross-reference to new" << " iRODS object PID in existing" <<
    "metadata handle\"";
new_response.command = temp_strm.str();
response_deque.push_back(new_response);
++errors_occurred;
return 1;
} /* if (status != 0) */
else {
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::put': "
            << endl << "'Handle_Value_Type::add_value' succeeded, returning 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
temp_strm.str("");
temp_strm << "PUT_RESPONSE" << response.local_filename << "\" " <<
    "0\"Success:" << " In existing metadata handle" << iter->handle <<
    ":" << " Added cross-reference to new iRODS object PID" << " " <<
    handle_vector.back().handle << "\n";
new_response.command = temp_strm.str();
response_deque.push_back(new_response);
} /* else */

```

183.

Log

[LDF 2013.06.06.] BUG FIX: Now calling `imeta rm` to remove the AVU, if it already exists.

```

⟨ Scan_Parse_Parameter_Type::put definition 151 ⟩ +≡
curr_avu.clear();
curr_avu.set("DC_METADATA_PID", iter->handle);
temp_strm.str("");
temp_strm << "env_irodsEnvFile=" << irods_env.filename << "imeta_rm-d\\"" <<
    response.local_filename << "\DC_METADATA_PID\" << iter->handle << "\>/dev/null2>&1;\" <<
    "env_irodsEnvFile=" << irods_env.filename << "imeta_add-d\\"" << response.local_filename <<
    "\DC_METADATA_PID\" << iter->handle << "";
#endif /* DEBUG_COMPILE */
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "temp_strm.str() == " << temp_strm.str() << endl;
    cerr << "response.pid_str == " << response.pid_str << endl;
    cerr << "pid_str == " << pid_str << endl;
    cerr << "iter->handle == " << iter->handle << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

184.

```

⟨ Scan_Parse_Parameter_Type::put definition 151 ⟩ +≡
status = system(temp_strm.str().c_str());
if (status == -1 || !WIFEXITED(status)) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] " << "ERROR! In 'Scan_Parse_Parameter_Type::"
        "put' : system failed, returning " << status << ". " << endl;
    if (WIFEXITED(status)) cerr << "WEXITSTATUS(status) == " << WEXITSTATUS(status) << endl;
    else cerr << "Process failed to exit." << endl;
    cerr << "Exiting 'Scan_Parse_Parameter_Type::put' "
        "unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    temp_strm.str("");
    ++errors_occurred;
    temp_strm << "PUT RESPONSE\\"" << response.local_filename << "\\" <<
        "2\\"Server\error:\\" << "Failed to store Dublin Core Metadata PID\\"" <<
        iter->handle << '\in iRODS\object\metadata\'";
    new_response.command = temp_strm.str();
    response_deque.push_back(new_response);
    ++errors_occurred;
    return 1;
} /* (status == -1 || !WIFEXITED(status)) */

```

185.

```

⟨ Scan_Parse_Parameter_Type::put definition 151 ⟩ +≡
else
  if (WEXITSTATUS(status) ≠ 0) {
    lock_cerr_mutex();
    cerr << "[Thread]" << thread_ctr << "] " << "ERROR! In 'Scan_Parse_Parameter_Type::"
    put' : "imeta' command (called via 'system') failed, " << "returning" <<
    WEXITSTATUS(status) << ". " << endl << "Exiting 'Scan_Parse_Parameter_Type::put'" <<
    "unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    temp_strm.str("");
    ++errors_occurred;
    temp_strm << "PUT_RESPONSE" << response.local.filename << "\n" <<
    "2\"Server_error:" << "Failed to store Dublin Core Metadata PID" <<
    iter-handle << "' in iRODS object metadata\"";
    new_response.command = temp_strm.str();
    response_deque.push_back(new_response);
    ++errors_occurred;
    return 1;
} /* else if (WEXITSTATUS(status) ≠ 0) */

```

186.

```

⟨ Scan_Parse_Parameter_Type::put definition 151 ⟩ +≡
else /* imeta command succeeded. */
{
#if DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread]" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::put'" <<
    "'system' succeeded WEXITSTATUS(status) == " << WEXITSTATUS(status) << ". " << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  curr_irods_object.avu_vector.push_back(curr_avu);
  curr_avu.clear();
  temp_strm.str("");
  temp_strm << "PUT_RESPONSE" << response.local.filename << "\n" << "0\"Success:" <<
  "Stored_Dublin_Core_Metadata_PID" << iter-handle << "' in iRODS object metadata\"";
  new_response.command = temp_strm.str();
  response_deque.push_back(new_response);
} /* else (imeta command succeeded.) */
} /* for */
#endif DEBUG_COMPILE
if (DEBUG) {
  lock_cerr_mutex();
  cerr << endl;
  unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (metadata_handle_vector.size() == 0) */

```

187.

```
<Scan_Parse_Parameter_Type::put definition 151> +≡
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "[Thread" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::put':"
                " metadata_handle_vector.size() == 0. Not adding cross-references." << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

188.

Log

[LDF 2013.06.06.] BUG FIX: Now calling imeta rm to remove the AVU, if it already exists.

```
<Scan_Parse_Parameter_Type::put definition 151> +≡
    curr_avu.clear();
    curr_avu.set("PID", response.pid_str);
    temp_strm.str("");
    temp_strm << "env_irodsEnvFile=" << irods_env_filename << "imeta rm -d\""
        << response.local_filename << "\" PID\" " << response.pid_str << "\">/dev/null2>&1;" <<
        "env_irodsEnvFile=" << irods_env_filename << "imeta add -d\""
        << response.local_filename << "\" PID\" " << response.pid_str << "\"";
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "temp_strm.str() == " << temp_strm.str() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

189.

```

⟨ Scan_Parse_Parameter_Type::put definition 151 ⟩ +≡
status = system(temp_strm.str().c_str());
if (status ≡ -1 ∨ ¬WIFEXITED(status)) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] " << "ERROR! In 'Scan_Parse_Parameter_Type::\n"
        put': " << "'system' failed, returning" << status << ". " << endl;
    if (WIFEXITED(status)) cerr << "WEXITSTATUS(status)==" << WEXITSTATUS(status) << endl;
    else cerr << "Process failed to exit." << endl;
    cerr << "Exiting 'Scan_Parse_Parameter_Type::put'" <<
        "unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    temp_strm.str("");
    ++errors_occurred;
    temp_strm << "PUT_RESPONSE" << response.local_filename << "\n" << "2 \"Server error: " <<
        "Failed to store PID" << response.pid_str << " in iRODS object metadata\"";
    new_response.command = temp_strm.str();
    response_deque.push_back(new_response);
    ++errors_occurred;
    return 1;
} /* (status ≡ -1 ∨ ¬WIFEXITED(status)) */

```

190.

Log

[LDF 2013.01.31.] Added this section.

[LDF 2013.01.31.] BUG FIX: Now testing WEXITSTATUS(status).

```

⟨ Scan_Parse_Parameter_Type::put definition 151 ⟩ +≡
else
    if (WEXITSTATUS(status) ≠ 0) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] " << "ERROR! In 'Scan_Parse_Parameter_Type::\n"
            put': " << "'imeta' command (called via 'system') failed, " << "returning" <<
            WEXITSTATUS(status) << ". " << endl << "Exiting 'Scan_Parse_Parameter_Type::put'" <<
            "unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        temp_strm.str("");
        ++errors_occurred;
        temp_strm << "PUT_RESPONSE" << response.local_filename << "\n" << "2 \"Server error: " <<
            "Failed to store PID" << response.pid_str << " in iRODS object metadata\"";
        new_response.command = temp_strm.str();
        response_deque.push_back(new_response);
        ++errors_occurred;
        return 1;
} /* else if (WEXITSTATUS(status) ≠ 0) */

```

191.

```
< Scan_Parse_Parameter_Type::put definition 151 > +≡
else      /* imeta command succeeded. */
{
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::put': "
            "system'succeeded" << WEXITSTATUS(status) << " == " << WEXITSTATUS(status) << ". " << endl;
        unlock_cerr_mutex();
    }      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */
    curr_irods_object.avu_vector.push_back(curr_avu);
    curr_avu.clear();
    temp_strm.str("");
    temp_strm << "PUT RESPONSE " << response.local_filename << "\00\Success:" <<
        "Stored_PID" << response.pid.str << "' in iRODS_object_metadata'";
    new_response.command = temp_strm.str();
    response_deque.push_back(new_response);
}      /* else (imeta command succeeded.) */
}      /* if (response.pid.options & 1) */
```

192. At this point, we only need to know whether a row corresponding to *curr_irods_object* exists in the *Irods_Objects* table and if so, what its *id* is; We don't need to have the data from the database entry fetched into *curr_irods_object*. We therefore pass *true* as the *id_only* argument to **Irods_Object_Type::get_from_database**. [LDF 2013.01.09.]

```
< Scan_Parse_Parameter_Type::put definition 151 > +≡
status = curr_irods_object.get_from_database(mysql_ptr, true);
```

193.

```
< Scan_Parse_Parameter_Type::put definition 151 > +≡
if (status == 0)      /* curr_irods_object.get_from_database returned 0 */
{
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::put':"
            endl << "'Irods_Object_Type::get_from_database' succeeded, returning 0." <<
            endl << "No row exists for iRODS_object" << " " << response.local_filename << "' "
            "in 'Irods_Objects' table yet." << endl << "Will create." << endl;
        unlock_cerr_mutex();
    }      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */
```

194.

```

⟨ Scan_Parse_Parameter_Type ::put definition 151 ⟩ +≡
status = curr_irods_object.write_to_database(mysql_ptr);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << "[Thread_" << thread_ctr << "] " << "ERROR! In 'Scan_Parse_Parameter_Type::put':"
        endl << "'Irods_Object_Type::write_to_database' failed, returning " << status <<
        "." << endl << "Failed to write data for 'Irods_Object_Type' curr_irods_object' "
        "to 'gwiridsif.Irods_Objects' database table." << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
unlock_cerr_mutex();
++errors_occurred;
temp_strm.str("");
temp_strm << "PUT_RESPONSE \" " << response.local_filename << "\" " << "1\"Server_error:" <<
    "C++ error: Failed to write data for 'Irods_Object_Type' object"
    "to 'gwiridsif.Irods_Objects' database table";
new_response.command = temp_strm.str();
response_deque.push_back(new_response);
++errors_occurred;
return 1;
} /* if (status ≠ 0) */
#endif /* DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread_" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::put':"
        endl << "'Irods_Object_Type::write_to_database' succeeded, returning 0." << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
/* if (status == 0) (curr_irods_object.get_from_database returned 0) */
else if (status == 1) /* curr_irods_object.get_from_database returned 1 */
{
#endif /* DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread_" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::put':"
        endl << "'Irods_Object_Type::get_from_database' succeeded, returning 1." << endl <<
        "A row already exists for iRODS object " << response.local_filename << " "
        "in 'Irods_Objects' table." << endl << "Will modify." << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

195. The *last_modified* field in the *Irods_Objects* table must be updated. If a new PID has been generated, an entry must be written in *Irods_Objects_Handles*. [LDF 2013.01.09.]

```

⟨ Scan_Parse_Parameter_Type::put definition 151 ⟩ +≡
    status = curr_irods_object.update(mysql_ptr);
    if (status ≠ 0) {
        lock_cerr_mutex();
        cerr << "[Thread_" << thread_ctr << "] " << "ERROR! In 'Scan_Parse_Parameter_Type::put':"
            endl << "'Irods_Object_Type::update' failed, returning " << status << "."
            endl << "Failed to update 'gwirdsif.Irods_Objects' database table."
            endl << "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        ++errors_occurred;
        temp_strm.str("");
        temp_strm << "PUT RESPONSE \" " << response.local_filename << "\" "
            << "1\" " << "Server error: "
            "C++ error: Failed to update 'Irods_Objects' database table";
        new_response.command = temp_strm.str();
        response_deque.push_back(new_response);
        ++errors_occurred;
        return 1;
    } /* if (status ≠ 0) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread_" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::put':"
            endl << "'Irods_Object_Type::update' succeeded, returning 0."
            endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* else if (status ≡ 1) (curr_irods_object.get_from_database returned 1) */

```

196.

```
< Scan_Parse_Parameter_Type::put definition 151 > +≡
else
  if (status > 1) {
    lock_cerr_mutex();
    cerr << "[Thread_" << thread_ctr << "] " << "ERROR! In 'Scan_Parse_Parameter_Type::\n"
      put' : " << endl << 'Irods_Object_Type::get_from_database' _returned " <<
      status << "(>1)." << endl << "Multiple_rows_exists_for_iRODS_object" " <<
      response.local_filename << ' " << "in 'Irods_Objects' _table." << endl <<
      "This_is_currently_not_allowed. It_may_be_in_the_future, if " <<
      "there's_a_reason_to_do_so." << endl << "Exiting_function_unsuccessfully_with_re\"
      turn_value_1." << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    temp_strm.str("");
    temp_strm << "PUT_RESPONSE" << response.local_filename << "\\" << "1" \\"Server_error:\\" <<
      "Database_error:\\"Multiple_rows_in 'Irods_Objects' _table" << "for_filename" " <<
      response.local_filename << '\\"';
    new_response.command = temp_strm.str();
    response_deque.push_back(new_response);
    ++errors_occurred;
    return 1;
} /* else if (status > 1) */
```

197.

```
< Scan_Parse_Parameter_Type::put definition 151 > +≡
else /* status < 0 */
{
  lock_cerr_mutex();
  cerr << "[Thread_" << thread_ctr << "] " << "ERROR! In 'Scan_Parse_Parameter_Type::put' : " <<
    endl << 'Irods_Object_Type::get_from_database' _failed, returning " << status <<
    "(<0)." << endl << "Exiting_function_unsuccessfully_with_return_value_1." << endl;
  unlock_cerr_mutex();
  ++errors_occurred;
  temp_strm.str("");
  temp_strm << "PUT_RESPONSE" << response.local_filename << "\\" << "1" \\"Server_error:\\" <<
    "Database_error:\\"Failed_to_check 'Irods_Objects' _database_table" <<
    "for_filename" " << response.local_filename << '\\"';
  new_response.command = temp_strm.str();
  response_deque.push_back(new_response);
  ++errors_occurred;
  return 1;
} /* else (status < 0) */
```

198. Clean up. [LDF 2012.09.27.]

```
< Scan_Parse_Parameter_Type::put definition 151 > +≡
```

199.

```
<Scan_Parse_Parameter_Type::put definition 151> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] Exiting 'Scan_Parse_Parameter_Type::put'" <<
            " successfully with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; } /* End of Scan_Parse_Parameter_Type::put definition */
```

200. get. [LDF 2012.10.02.]

Log

[LDF 2012.10.02.] Added this function.

[LDF 2013.04.04.] Added arguments `char *buffer_ptr` and `unsigned int buff_size`. Made `string flags` and `string client_side_filename` non-optional. Now calling this function from `exchange_data_with_client` instead of `yyparse`. Now using `buffer_ptr` to pass back a text response to `exchange_data_with_client`. Currently, this is only done in the case of an error. Otherwise, a response of `type ≡ Response_Type::SEND_FILE_TYPE` is pushed onto `response_deque`, as before.

Calling this function from `exchange_data_with_client` ensures that commands such as `cd`, that can affect the operation of this and other functions, are performed first. When functions are called in the parser rules, they are executed while operations may be pending because the corresponding responses are on `response deque` waiting to be processed.

```
<Scan_Parse_Parameter_Type function declarations 38> +≡
```

```
int get(string filename, string flags, string client_side_filename, char *buffer_ptr, unsigned int
buff_size);
```

201.

```
<Scan_Parse_Parameter_Type::get definition 201> ≡
int Scan_Parse_Parameter_Type::get(string filename, string flags, string client_side_filename, char
*buffer_ptr, unsigned int buff_size){ bool DEBUG = false; /* true */
set_debug_level(DEBUG, 0, 0);
stringstream temp_strm;
int status = 0;
ofstream out_strm;
size_t pos;
Response_Type response;
response.type = Response_Type::COMMAND_ONLY_TYPE;
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] Entering 'Scan_Parse_Parameter_Type::get'." <<
            endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

See also sections 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, and 215.

This code is used in section 305.

202.

```
< Scan_Parse_Parameter_Type::get definition 201 > +≡
    memset(buffer_ptr, 0, buff_size);
```

203. Test *flags* for **-f** (force) option. If present, set **bool overwrite** to *true*. !! PLEASE NOTE: This only works if **-f** is used separately from other options, or if it is the first option in a list preceded by a single hyphen.

Log

[LDF 2012.11.19.] Added this section.

```
< Scan_Parse_Parameter_Type::get definition 201 > +≡
    bool overwrite = false;
    pos = flags.find("-f");
    if (pos != string::npos) {
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "[Thread" << thread_ctr << "] In 'Scan_Parse_Parameter_Type::get':"
                << endl <<
                "'flags' contains '-f' (force) option. Setting 'bool overwrite' to 'true'." <<
                endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        overwrite = true;
    } /* if (pos != string::npos) */
    else /* The temporary file already exists, so the -f option must be used to overwrite it. This does
           not affect whether the remote (client-side) file will be overwritten! [LDF 2012.11.19.] */
    {
        flags.insert(0, "-f");
#endif DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "[Thread" << thread_ctr << "] In 'Scan_Parse_Parameter_Type::get':"
                << endl <<
                "'flags' does not contain '-f' (force) option. 'bool overwrite' "
                "remains set to 'false'." << endl <<
                "Using the '-f' option to overwrite the local (server-side) temporary file, "
                "but the remote (client-side) file will not be overwritten, if it already "
                "exists." << endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    } /* else */
}
```

204. Create temporary file. The name will be stored in *response.local_filename*. [LDF 2012.11.16.]

Log

[LDF 2012.11.16.] Added this section.

```

⟨ Scan_Parse_Parameter_Type::get definition 201 ⟩ +≡
char temp_filename[] = "/tmp/gwirdsif.XXXXXX";
int fd = mkstemp(temp_filename);
if (fd == -1) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] ERROR! In 'Scan_Parse_Parameter_Type::get':"
        << endl << "'mkstemp' failed, returning -1. Failed to create temporary file."
        << endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    return 1;
} /* if (fd == -1) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] In 'Scan_Parse_Parameter_Type::get':"
            << endl << "'mkstemp' succeeded. 'temp_filename' == " << temp_filename << "."
            << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
close(fd);
fd = 0;
response.local_filename = temp_filename;
temp_file_vector.push_back(temp_filename);

```

205.

```

⟨ Scan_Parse_Parameter_Type::get definition 201 ⟩ +≡
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] In 'Scan_Parse_Parameter_Type::get':"
        << endl << "'filename' == " << filename << endl << "'irods_current_dir' == "
        << irods_current_dir << endl << "response.local_filename == " << response.local_filename << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

206.

```
<Scan_Parse_Parameter_Type::get definition 201> +≡
  pos = filename.find_last_of("/");
  if (pos == string::npos) { /* filename doesn't contain any directories */
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] In 'Scan_Parse_Parameter_Type::get':" << endl <<
      "'filename'" << filename << " doesn't contain any directories." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (pos == string::npos) */
```

207.

```
<Scan_Parse_Parameter_Type::get definition 201> +≡
else
  if (filename[filename.size() - 1] == '/') {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] ERROR! In 'Scan_Parse_Parameter_Type::get':" <<
      endl << "'filename'" << filename << " is a directory and cannot be 'gotten'." <<
      endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    temp_strm.str("");
    temp_strm << "GET_FILE_RESPONSE" << filename << "\1\Error:" <<
      "Filename is a directory:" << filename << ". Can't get.";
    strcpy(buffer_ptr, temp_strm.str().c_str());
    ++errors_occurred;
    return 1;
} /* else if (filename[filename.size() - 1] == '/') */
```

208.

```
<Scan_Parse_Parameter_Type::get definition 201> +≡
  if (filename[0] == '/') {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] In 'Scan_Parse_Parameter_Type::get':" <<
      endl << "'filename'" << filename << ", is an absolute path." << endl <<
      "Not inserting 'irods_current_dir'" << irods_current_dir << "' at the front of it." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (filename[0] == '/') */
```

209.

```
<Scan_Parse_Parameter_Type::get definition 201> +≡
else /* filename[0] ≠ '/' */
{
    filename.insert(0, "/");
    filename.insert(0, irods_current_dir);
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "After inserting 'irods_current_dir': " << "filename==" << filename << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* else (filename[0] ≠ '/') */
```

210. !! TODO: I could try to implement some way of changing the directory on the client-side during the dialogue between client and server. I'm not sure that this is really practicable (or even possible), though. [LDF 2013.04.05.]

Log

[LDF 2013.04.04.] Added this section.

```
<Scan_Parse_Parameter_Type::get definition 201> +≡
if (client_side_filename.empty()) {
    client_side_filename = basename(filename.c_str());
    if (client_side_filename.empty()) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] ERROR! In 'Scan_Parse_Parameter_Type::get': "
            << endl << "'basename' failed, returning the empty string. " <<
            endl << "'filename'" <= " filename << " is invalid. " << endl <<
            "Exiting function unsuccessfully with return value 1. " << endl;
        unlock_cerr_mutex();
        ++errors_occurred;
        temp_strm.str("");
        temp_strm << "GETFILERESPONSE" << filename << "\1\Error:" <<
            "Filename is invalid: " << filename << ". Can't get.";
        strcpy(buffer_ptr, temp_strm.str().c_str());
        ++errors_occurred;
        return 1;
    } /* if (client_side_filename.empty()) (inner) */
} /* if (client_side_filename.empty()) (outer) */
response.remote_filename = client_side_filename;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "response.remote_filename==" << response.remote_filename << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

211.

```

⟨ Scan_Parse_Parameter_Type :: get definition 201 ⟩ +≡
temp_strm.str("");
temp_strm << "env_irodsEnvFile=" << irods_env_filename << "ugetu" << flags << "u" << filename <<
"u" << response.local_filename << "u2>&1;uecho$?";
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "temp_strm.str() == " << temp_strm.str() << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
errno = 0;
FILE *fp = popen(temp_strm.str().c_str(), "r");
if (fp == 0) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] ERROR! In 'Scan_Parse_Parameter_Type::get':"
        "'popen' failed, returning NULL: " << strerror(errno) << endl <<
        "Can't get \\"file.uWillSendFailureNoticeToClient." << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    temp_strm.str("");
    temp_strm << "GET_FILE_RESPONSE" << filename << "\u1\u" "Server_error:" <<
        "Failed to retrieve file " << filename << "\ufrom\u" "irods" "\u";
    if (!client_side_filename.empty()) {
        /* client_side_filename should never be empty now, since it is set above. [LDF 2013.04.04.] */
        temp_strm << "\uCLIENT_SIDE_FILENAME" << client_side_filename << "\u";
    }
    response.command = temp_strm.str();
    response_deque.push_back(response);
    ++errors_occurred;
    return 1;
} /* if (fp == 0) */
else /* fp != 0 */
{
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] In 'Scan_Parse_Parameter_Type::get':"
        "'popen' succeeded." << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
char buffer[1024];
memset(buffer, 0, 1024);
status = fread(buffer, 1, 1024, fp);
if (status == 0) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] ERROR! " << "In 'Scan_Parse_Parameter_Type::get':"
        "'fread' failed, returning 0. Failed to read from pipe." << endl <<

```

```

    "Failed_to_get\"file.Will_send_failure_notice" << "to_client." << endl <<
    "Exiting_function_unsuccessfully_with_return_value_1." << endl;
unlock_cerr_mutex();
temp_strm.str("");
temp_strm << "GET_RESPONSE_FILE\"" << filename << "\"1\"Server_error:" <<
    "Failed_to_read_output_and/or_exit_status_from" << "'iget' command.\\"";
if (!client_side_filename.empty()) {
    temp_strm << "CLIENT_SIDE_FILENAME\"" << client_side_filename << "\"";
}
response.command = temp_strm.str();
response_deque.push_back(response);
pclose(fp);
++errors_occurred;
return 1;
} /* if (status == 0) */
else if (status == 1024) {
lock_cerr_mutex();
cerr << "[Thread" << thread_ctr << "] ERROR!" << "In 'Scan_Parse_Parameter_Type::get':"
    "'fread' returned 1024: Pipe output exceeds maximum length." << endl <<
    "Failed_to_get\"file." << "Will_send_failure_notice_to_client." << endl <<
    "Exiting_function_unsuccessfully_with_return_value_1." << endl;
unlock_cerr_mutex();
temp_strm.str("");
temp_strm << "GETFILE_RESPONSE\"" << filename << "\"1\"Server_error:" <<
    "Output from 'iget' command exceeds maximum length" << "(1023 characters).\\"";
if (!client_side_filename.empty()) {
    temp_strm << "CLIENT_SIDE_FILENAME\"" << client_side_filename << "\"";
}
response.command = temp_strm.str();
response_deque.push_back(response);
pclose(fp);
++errors_occurred;
return 1;
} /* else if (status == 1024) */
else /* fread succeeded and read valid number of characters */
{
#if DEBUG_COMPILE
if (DEBUG) {
lock_cerr_mutex();
cerr << "[Thread" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::get':"
    "'popen' succeeded." << endl << "'buffer' == " << buffer << endl;
unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
}

```

212.

```
<Scan_Parse_Parameter_Type::get definition 201> +≡
    /* Get the return value of igure command. */
    char *ret_val_ptr = buffer + strlen(buffer) - 1;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "ret_val_ptr== " << ret_val_ptr << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    --ret_val_ptr;
    while (isdigit(ret_val_ptr[0])) --ret_val_ptr;
    if (¬(ret_val_ptr[0] ≡ '-' ∨ ret_val_ptr[0] ≡ '+')) ++ret_val_ptr;
    int ret_val;
    sscanf(ret_val_ptr, "%d", &ret_val);
    ret_val_ptr[0] = '\0'; /* Remove exit status from end of buffer */
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::get':"
            << endl << "ret_val_ptr== " << ret_val_ptr << endl << "ret_val== " << ret_val << endl <<
            "'buffer'" << buffer << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

213.

```

⟨ Scan_Parse_Parameter_Type::get definition 201 ⟩ +≡
  if (ret_val == 0) {
#if DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread_"
    << thread_ctr << "] "
    << "In_`Scan_Parse_Parameter_Type::get':"
    << "'iget'_succeeded." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  pclose(fp);
  fp = 0;
  temp_strm.str("");
  temp_strm << "GETFILERESPONSE\" "
  << filename << "\0\" "
  << "'iget'_command_succeeded,_returning\0";
  if (strlen(buffer) > 0) temp_strm << ":" << buffer;
  else temp_strm << ".";
  temp_strm << "\n";
  if (!client_side_filename.empty()) {
    temp_strm << "_CLIENT_SIDE_FILENAME\" "
    << client_side_filename << "\n";
  }
  if (overwrite) temp_strm << "_OVERWRITE";
  response.command = temp_strm.str();
  response.type = Response_Type::SEND_FILE_TYPE;
  response_deque.push_back(response);
  temp_strm.str("");
} /* if (ret_val == 0) */

```

214.

```

⟨ Scan_Parse_Parameter_Type::get definition 201 ⟩ +≡
  else /* ret_val ≠ 0 */
  {
    lock_cerr_mutex();
    cerr << "[Thread_"
    << thread_ctr << "] "
    << "ERROR! "
    << "In_`Scan_Parse_Parameter_Type::get':"
    << "'iget'_failed,_returning"
    << ret_val << ":" << endl << "'buffer'_=="
    << buffer << endl << "Will_send_failure_message_to_client."
    << endl << "Exiting_function_unsuccessfully_with_return_value_1."
    << endl;
    unlock_cerr_mutex();
    temp_strm.str("");
    temp_strm << "GETFILERESPONSE\" "
    << filename << "\1\"Server_error:\n"
    << "'iget'_command_failed,_returning"
    << ret_val << ":" << buffer << "\n";
    if (!client_side_filename.empty())
      temp_strm << "_CLIENT_SIDE_FILENAME\" "
      << client_side_filename << "\n";
    pclose(fp);
    strcpy(buffer_ptr, temp_strm.str().c_str());
    ++errors_occurred;
    return 1;
} /* else (ret_val ≠ 0) */
} /* else. fread succeeded and read valid number of characters */
} /* else (fp ≠ 0). popen succeeded */

```

215. Exit. [LDF 2012.10.02.]

```
<Scan_Parse_Parameter_Type::get definition 201> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] Exiting 'Scan_Parse_Parameter_Type::get'" <<
            " successfully with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; } /* End of Scan_Parse_Parameter_Type::get definition */
```

216. Send to peer. [LDF 2012.10.02.]

217. Main version. [LDF 2012.10.02.]

Log

[LDF 2012.10.02.] Added this function.

[LDF 2012.11.20.] Changed so that *buffer_ptr and the contents of filename can both be sent to the peer in a single call to this function.

<Scan_Parse_Parameter_Type function declarations 38> +≡

```
int send_to_peer(char **buffer_ptr, unsigned int char_ctr = 0, string filename = "");
```

218.

```
<Scan_Parse_Parameter_Type::send_to_peer definitions 218> ≡
int Scan_Parse_Parameter_Type::send_to_peer(char **buffer_ptr, unsigned int char_ctr, string
    filename){ bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    stringstream temp_strm;
    int status = 0;
    string send_func_str = (remote_connection == true) ? "'gnutls_record_send'" : "'send'";
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] Entering '" <<
            "'Scan_Parse_Parameter_Type::send_to_peer'" << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

See also sections 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, and 233.

This code is used in section 305.

219.

```
< Scan_Parse_Parameter_Type::send_to_peer definitions 218 > +≡
  if (buffer_ptr ≡ 0 ∧ char_ctr ≠ 1 ∧ filename.empty()) {
    lock_cerr_mutex();
    cerr ≪ "[Thread" ≪ thread_ctr ≪ "]WARNING!" ≪
      "In‘Scan_Parse_Parameter_Type::send_to_peer’:"
    ≪ endl ≪ "'buffer_ptr'==0,‘char_ctr’=="
    ≪ char_ctr ≪ "<>1" ≪
    "and‘filename’isempty." ≪ endl ≪ "Nothingto sendto peer." ≪ endl ≪
    "Exitingfunctionunsuccessfullywithreturnvalue2." ≪ endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    return 2;
} /* if (buffer_ptr ≡ 0 ∧ filename.empty()) */
```

220.

```
< Scan_Parse_Parameter_Type::send_to_peer definitions 218 > +≡
  else
    if (buffer_ptr ≡ 0 ∧ char_ctr ≡ 1 ∧ filename.empty()) {
#define DEBUG_COMPILE
      if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ "[Thread" ≪ thread_ctr ≪ "]In‘Scan_Parse_Parameter_Type::send_to_peer’:"
        ≪ endl ≪ "'buffer_ptr'==0,‘char_ctr’==1and‘filename’isempty."
        ≪ endl ≪ "SendingsingleNULLbyte to peer." ≪ endl;
        unlock_cerr_mutex();
      } /* if (DEBUG) */
#undef DEBUG_COMPILE
    } /* else if (buffer_ptr ≡ 0 ∧ char_ctr ≡ 1 ∧ filename.empty()) */
```

221.

```
< Scan_Parse_Parameter_Type::send_to_peer definitions 218 > +≡
  if (buffer_ptr ≠ 0) {
#define DEBUG_COMPILE
    if (DEBUG) {
      lock_cerr_mutex();
      cerr ≪ "[Thread" ≪ thread_ctr ≪ "]In‘Scan_Parse_Parameter_Type::send_to_peer’:"
      ≪ endl ≪ "'buffer_ptr'!=0,‘char_ctr’=="
      ≪ char_ctr ≪ "."
      ≪ endl ≪ "Sendingallorpartof‘*buffer_ptr’topeer." ≪ endl;
      unlock_cerr_mutex();
    } /* if (DEBUG) */
#undef DEBUG_COMPILE
  } /* if (buffer_ptr ≠ 0) */
```

222.

```

⟨ Scan_Parse_Parameter_Type :: send_to_peer definitions 218 ⟩ +≡
  if (filename.empty()) {
#if DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] In 'Scan_Parse_Parameter_Type::send_to_peer':"
      << endl << "'char_ctr'" << char_ctr << " and 'filename'" << "is non-empty."
      << endl << "Sending all or part of file" << filename << " to peer." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (filename.empty()) */

```

223.

```

⟨ Scan_Parse_Parameter_Type :: send_to_peer definitions 218 ⟩ +≡
  char buffer[BUFFER_SIZE];
  memset(buffer, 0, BUFFER_SIZE);
  if (buffer_ptr ≡ 0 ∧ filename.empty() ∧ char_ctr ≡ 1) {
    if (remote_connection ≡ true) {
      status = gnutls_record_send(session, buffer, 1);
    }
    else {
      status = send(sock, buffer, 1, 0);
    }
    if (status ≡ -1) {
      lock_cerr_mutex();
      cerr << "[Thread" << thread_ctr << "] ERROR! In 'Scan_Parse_Parameter_Type\
        ::send_to_peer': failed, returning -1." << endl;
      if (remote_connection ≡ true) cerr << "Error:" << gnutls_strerror(status) << endl;
      else cerr << "Error:" << strerror(errno) << endl;
      cerr << "Exiting function unsuccessfully with return value 1." << endl;
      unlock_cerr_mutex();
      close(sock);
      sock = 0;
      ++errors_occurred;
      return 1;
    } /* if (status ≡ -1) */
#endif DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "[Thread" << thread_ctr << "] In 'Scan_Parse_Parameter_Type::send_t\
        o_peer': " << send_func_str << " succeeded, returning" << status << "." << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (buffer_ptr ≡ 0 ∧ filename.empty() ∧ char_ctr ≡ 1) */

```

224.

```
( Scan_Parse_Parameter_Type::send_to_peer definitions 218 ) +≡
    if (buffer_ptr ≠ 0 ∧ strlen(*buffer_ptr) < BUFFER_SIZE) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≈ "[Thread_]" ≈ thread_ctr ≈ "]_In_‘Scan_Parse_Parameter_Type::send_to_peer’:uu" ≈
            "‘buffer_ptr’_!=_0_&&_‘strlen(*buffer_ptr)’_<_‘BUFFER_SIZE’.“ ≈ endl ≈
            “‘strlen(*buffer_ptr)’_==_” ≈ strlen(*buffer_ptr) ≈ endl ≈ “‘BUFFER_SIZE’_==_” ≈
            BUFFER_SIZE ≈ endl ≈ “‘char_ctr’_==_” ≈ char_ctr ≈ “.uu” ≈ endl;
        if (char_ctr ≡ 0 ∨ char_ctr ≥ strlen(*buffer_ptr))
            cerr ≈ "Will_send_entire_contents_of_‘*buffer_ptr’_to_peer.“ ≈ endl;
        else cerr ≈ "Will_send_the_first_“ ≈ char_ctr ≈ “_characters_u“ ≈
            “of_‘*buffer_ptr’_to_peer.“ ≈ endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

225. BUG FIX: If **buffer_ptr* is empty, now setting *char_ctr* to 1, so that at least one byte will be sent to the peer. Previously, *char_ctr* was set to *strlen(*buffer_ptr)* ≡ 0, so that 0 bytes were “sent”, which caused the peer to block. Of course, **buffer_ptr* should normally never be empty in this case, but it can happen by mistake. [LDF 2012.11.21.]

```
( Scan_Parse_Parameter_Type::send_to_peer definitions 218 ) +≡
    if (char_ctr ≡ 0 ∨ char_ctr > strlen(*buffer_ptr)) char_ctr = strlen(*buffer_ptr);
    if (strlen(*buffer_ptr) ≡ 0) char_ctr = 1;
    if (remote_connection ≡ true) {
        status = gnutls_record_send(session, *buffer_ptr, char_ctr);
    }
    else {
        status = send(sock, *buffer_ptr, char_ctr, 0);
    }
    if (status < 0) {
        lock_cerr_mutex();
        cerr ≈ "[Thread_]" ≈ thread_ctr ≈ "]_ERROR!_In_‘Scan_Parse_Parameter_Type\_
            ::send_to_peer’:uu“ ≈ send_func_str ≈ “_failed,_returning_“ ≈ status ≈ “.“ ≈ endl;
        if (remote_connection ≡ true) cerr ≈ “Error:_“ ≈ gnutls_strerror(status) ≈ endl;
        else cerr ≈ “Error:_“ ≈ strerror(errno) ≈ endl;
        cerr ≈ “Exiting_function_unsuccessfully_with_return_value_1.“ ≈ endl;
        unlock_cerr_mutex();
        close(sock);
        sock = 0;
        ++errors_occurred;
    return 1;
} /* if (status ≡ -1) */
else if (DEBUG) {
    lock_cerr_mutex();
    cerr ≈ "[Thread_]" ≈ thread_ctr ≈ "]_In_‘Scan_Parse_Parameter_Type::send_to_peer’:uu“ ≈
        send_func_str ≈ “_succeeded,_returning_“ ≈ status ≈ “.“ ≈ endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
/* if (buffer_ptr ≠ 0 ∧ strlen(*buffer_ptr) < BUFFER_SIZE) */
```

226. !! TODO: [LDF 2012.11.20.] Account for cases *buffer_ptr is larger than BUFFER_SIZE. Must send contents in loop. After loop, must make sure that number of characters send is not BUFFER_SIZE, as above, so peer won't block.

Normally, this case should never occur, because the bytes to be transferred would have been stored in a file. [LDF 2013.06.06.]

```
<Scan_Parse_Parameter_Type::send_to_peer definitions 218> +≡
else
  if (buffer_ptr ≠ 0) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "]WARNING! In 'Scan_Parse_Parameter_Ty\
      pe::send_to_peer': " << endl << "This case hasn't been accounted for:" <<
      "'*buffer_ptr'!=0&&'strlen(*buffer_ptr')>='BUFFER_SIZE'" << endl <<
      "'strlen(*buffer_ptr')==" << strlen(*buffer_ptr) << endl << "'BUFFER_SIZE'==" <<
      BUFFER_SIZE << endl << "Exiting function unsuccessfully with return value 3." <<
      endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    return 3;
} /* else if (buffer_ptr ≠ 0) */
```

227. Send file *filename* to peer. [LDF 2012.11.16.]

Log

[LDF 2012.11.16.] Added this section.

```

⟨ Scan_Parse_Parameter_Type::send_to_peer definitions 218 ⟩ +≡
  if (¬filename.empty()) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] In 'Scan_Parse_Parameter_Type::send_to_peer':"
      " 'filename'" << filename << ". Will send file contents to peer." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  errno = 0;
  FILE *fp = fopen(filename.c_str(), "r");
  if (fp == 0) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] ERROR! In 'Scan_Parse_Parameter_Type\
      ::send_to_peer':"
      "'fopen' failed, returning NULL:" << endl <<
      strerror(errno) << endl << "Failed to open file"
      " " << filename << "."
      " Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    return 1;
  } /* if (fp == 0) */
#endif DEBUG_COMPILE
  else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] In 'Scan_Parse_Parameter_Type::send_to_peer':"
      "'fopen' succeeded. Opened file"
      " " << filename << " successfully."
      " << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  for ( ; ; ) { memset(buffer, 0, BUFFER_SIZE);
    char_ctr = fread(buffer, 1, BUFFER_SIZE, fp);
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] In 'Scan_Parse_Parameter_Type::send_to_peer':"
      " 'char_ctr'" << char_ctr << "."
      " << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  if (char_ctr == 0) {
    break;
  } /* if (char_ctr == 0) */
  if (remote_connection == true) {
    status = gnutls_record_send(session, buffer, char_ctr);
  }
}

```

{

228. Local connection using Unix domain socket. [LDF Date unknown.]

!! KLUDGE: Calling *sleep* to allow server and client to synchronize. Currently sleeping for 1 second. Problems may occur if this isn't long enough. If the program calling this function doesn't sleep briefly when UNIX domain sockets are being used, then the call to *recv* on the side of the peer may fail to receive the bytes sent by the *send* function and block waiting for bytes. This problem does not occur when using Inet sockets with the GNUTLS functions, which should be the normal usage.

!! TODO: Look up how *send* and *recv* work. I don't think *recv* should be blocking. [LDF 2012.11.19.]

```
< Scan_Parse_Parameter_Type::send_to_peer definitions 218 > +≡
else {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] In " 'Scan_Parse_Parameter_Type::send_to_peer' : " <<
            "'remote_connection'" == "false". Using a UNIX domain socket. " << endl <<
            "Going to sleep for 1/2 second in order to synchronize with client." <<
            endl << "If this isn't done, the call to 'recv' on the client-side" <<
            "may fail to receive the bytes sent and block." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    struct timespec t;
    t.tv_sec = 0;
    t.tv_nsec = 1000000000/2; /* sleep for .5 seconds */
    nanosleep(&t, 0);
    status = send(sock, buffer, char_ctr, 0);
}
```

229.

```
{ Scan_Parse_Parameter_Type::send_to_peer definitions 218 } +≡
if (status < 0) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] ERROR! In 'Scan_Parse_Parameter_Type'\
        ::send_to_peer'" << send_func_str << " failed, returning" << status << "."
    if (remote_connection ≡ true) cerr << "Error:" << gnutls_strerror(status) << endl;
    else cerr << "Error:" << strerror(errno) << endl;
    cerr << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    close(sock);
    sock = 0;
    ++errors_occurred;
    return 1;
} /* if (status ≡ -1) */
else if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] In 'Scan_Parse_Parameter_Type::send_to_peer'" <<
        send_func_str << " succeeded, returning" << status << "."
    unlock_cerr_mutex();
} /* else if (DEBUG) */
} /* for */
fclose(fp);
fp = 0;
```

230. If *send* sent exactly BUFFER_SIZE bytes on the final iteration of the loop above, then we send a single NULL byte, so the client won't block. [LDF 2012.07.27.] [LDF 2012.12.13.]

```

⟨ Scan_Parse_Parameter_Type::send_to_peer definitions 218 ⟩ +≡
  if (status ≡ BUFFER_SIZE) {
#if DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] In 'Scan_Parse_Parameter_Type::send_to_peer':"
    send_func_str << " sent exactly " << BUFFER_SIZE << " bytes."
    " Sending a single NULL byte, so the client won't block, waiting"
    " for more bytes." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  memset(buffer, 0, BUFFER_SIZE);
  if (remote_connection ≡ true) {
    status = gnutls_record_send(session, buffer, 1);
  }
  else {
    status = send(sock, buffer, 1, 0);
  }
  if (status ≡ -1) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] ERROR! In 'Scan_Parse_Parameter_Type\
      ::send_to_peer':"
    send_func_str << " failed, returning -1." << endl;
    if (remote_connection ≡ true) cerr << "Error:" << gnutls_strerror(status) << endl;
    else cerr << "Error:" << strerror(errno) << endl;
    cerr << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    close(sock);
    sock = 0;
    ++errors_occurred;
    return 1;
  } /* if (status ≡ -1) */
#endif DEBUG_COMPILE
  else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] In 'Scan_Parse_Parameter_Type::send_t\
      o_peer':"
    send_func_str << " succeeded, returning "
    status << "."
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (status ≡ BUFFER_SIZE) */
} /* if (!filename.empty()) */

```

231.

```
<Scan_Parse_Parameter_Type::send_to_peer definitions 218> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] " << "Exiting " 'Scan_Parse_Parameter_Type::sen\
            d_to_peer' " " << "successfully with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; /* Scan_Parse_Parameter_Type::send_to_peer */
}
```

232. Version with **const Response_Type** & argument. [LDF 2012.11.22.]

Log

[LDF 2012.11.22.] Added this function.

[LDF 2012.12.13.] BUG FIX: Now checking whether *send* or *gnutls_record_send* sent **BUFFER_SIZE** longer correct, because when **Scan_Parse_Parameter_Type::receive_file** or any other function tries to read in a loop, it reads **BUFFER_SIZE** bytes at a time, not **BUFFER_SIZE** – 1 bytes. **BUFFER_SIZE** – 1 is only used when calling *recv* or *gnutls_record_recv* a single time, i.e., not in a loop, so it won't block, unless there are no bytes to receive at all. Using **BUFFER_SIZE** – 1 ensures that there will be a NULL byte at the end of the buffer, so that it can be used as a C string (assuming it's been set to NULL before the call to *recv* or *gnutls_record_recv*).

```
<Scan_Parse_Parameter_Type function declarations 38> +≡
int send_to_peer(const Response_Type &response);
```

233.

```
<Scan_Parse_Parameter_Type::send_to_peer definitions 218> +≡
int Scan_Parse_Parameter_Type::send_to_peer(const Response_Type &response)
{
    char buffer[BUFFER_SIZE];
    memset(buffer, 0, BUFFER_SIZE);
    char *buff_ptr = buffer;
    if (!response.command.empty()) strcpy(buffer, response.command.c_str());
    return send_to_peer(&buff_ptr, 0, response.local_filename);
} /* End of Scan_Parse_Parameter_Type::send_to_peer(const Response_Type &response)
   definition */
```

234. Receive file. [LDF 2012.09.27.]

The automatic variable **bool** *gen_temp_file* is initialized to *false* and set to *true* if the **string** arguments *remote_filename* and *local_filename* are both empty. This makes it possible to have the file contents written to a temporary file. If *temp_filename_ptr* is non-null, the name of the temporary is stored in **temp_filename_ptr*, so that it can be made available to the caller. [LDF 2012.11.22.]

Return values:

- 0: Success
- 1: *access* error for local filename
- 2: Local file already exists and *overwrite* \equiv *false*
- 3: *mkdir* error
- 4: *access* error for local path
- 5: **ofstream** :: *open* error for local file

Log

[LDF 2012.09.27.] Added this function.

[LDF 2012.11.19.] Added arguments.

[LDF 2012.11.22.] Added automatic variable **bool** *gen_temp_file* = *false*. See TEX text above for explanation.

⟨ Scan_Parse_Parameter_Type function declarations 38 ⟩ +≡

```
int receive_file(string remote_filename = "", string local_filename = "", bool overwrite = false, string
    *new_local_filename_ptr = 0, string *temp_filename_ptr = 0);
```

235.

```

⟨ Scan_Parse_Parameter_Type::receive_file definition 235 ⟩ ≡
int Scan_Parse_Parameter_Type::receive_file(string remote_filename, string local_filename, bool
    overwrite, string *new_local_filename_ptr, string *temp_filename_ptr){ bool DEBUG = false;
/* true */
set_debug_level(DEBUG, 0, 0);
int status = 0;
int err_ret_val = 0;
bool discard = false;
bool gen_temp_file = false;
ofstream out_strm;
size_t pos;
string local_path;
string temp_path;
string temp_str; deque < string > dir_list;
char temp_filename[21];
if (is_gwirdcli) strcpy(temp_filename, "/tmp/gwirdcli.XXXXXX");
else strcpy(temp_filename, "/tmp/gwirdsif.XXXXXX");
int fd;
string save_local_filename;
char *pwd_str;
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] " << "Entering"
        << "Scan_Parse_Parameter_Type::receive_file". " << endl << "'remote_filename'" <<
        remote_filename << endl << "'local_filename'" << local_filename << endl <<
        "'overwrite'" << overwrite << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
if (new_local_filename_ptr != 0) *new_local_filename_ptr = "";

```

See also sections 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, and 252.

This code is used in section 305.

236.

Log

[LDF 2012.11.22.] Added this section.

```

⟨ Scan_Parse_Parameter_Type::receive_file definition 235 ⟩ +≡
if (local_filename.empty() & remote_filename.empty()) {
    gen_temp_file = true;
    goto RECEIVE_FILE_CONTENTS;
}

```

237.

```
<Scan_Parse_Parameter_Type::receive_file definition 235> +≡
else
  if (local_filename.empty()) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "‘local_filename’ is empty. Setting ‘remote_filename’ == "
        << remote_filename << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  local_filename = remote_filename;
} /* else if (local_filename.empty()) */
```

238. If *local_filename* is a path ending in a directory, i.e., if it ends with a slash, append the filename part of *remote_filename* to it. That is, if *remote_filename* is a path containing directories, only append the actual filename without the directories to *local_filename*. [LDF 2012.11.20.]

Log

[LDF 2012.11.20.] Added this section.

```
<Scan_Parse_Parameter_Type::receive_file definition 235> +≡
else
  if (local_filename[local_filename.size() - 1] == '/') {
    pos = remote_filename.find_last_of("/");
    if (pos != string::npos) {
#ifndef 0
    cerr << "temp_str == " << temp_str << endl;
#endif
    temp_str = remote_filename.substr(pos + 1);
    local_filename += temp_str;
} /* if (pos != string::npos) */
else local_filename += remote_filename;
#ifndef 0
    cerr << "remote_filename == " << remote_filename << endl << "local_filename == " <<
    local_filename << endl;
#endif
} /* else if (local_filename[local_filename.size() - 1] == '/') */
```

239. If *overwrite* \equiv *false*, check *local_filename*. If the file already exists, issue an error message and write file contents to a temporary file in */tmp/*. If the latter can't be opened for some reason, the file contents are discarded.

!! PLEASE NOTE: This function cannot just exit, because the file contents must be handled somehow. Otherwise, they will be read when the scanner tries to read input again, which will cause an error. [LDF 2012.11.19.]

 Log

[LDF 2012.11.19.] Added this section.

```

⟨ Scan_Parse_Parameter_Type::receive_file definition 235 ⟩ +≡
  save_local_filename = local_filename;
  pwd_str = getenv("PWD");
  if (pwd_str) {
    local_filename.insert(0, "/");
    local_filename.insert(0, pwd_str);
  }
#ifndef 0 /* 1 */
  cerr << "local_filename" << local_filename << endl;
#endif
  if (overwrite == false) {
    errno = 0;
    status = access(local_filename.c_str(), F_OK);
    if (status == -1 & errno == ENOENT) {
#ifndef DEBUG_COMPILE
      if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] In" <<
          "'Scan_Parse_Parameter_Type::receive_file':" << endl <<
          "'access' returned -1 and 'errno' == 'ENOENT'." << endl <<
          "File doesn't already exist. Will create." << endl;
        unlock_cerr_mutex();
      } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    } /* if (status == -1 & errno == ENOENT) */
    else if (status == -1) {
      lock_cerr_mutex();
      cerr << "[Thread" << thread_ctr << "] WARNING! In" <<
        "'Scan_Parse_Parameter_Type::receive_file':" << endl <<
        "'access' returned -1 and 'errno' != 'ENOENT'." << endl <<
        strerror(errno) << endl <<
        "Setting 'discard' == 'true'." << endl;
      unlock_cerr_mutex();
      err_ret_val = 1;
      ++warnings_occurred;
      discard = true;
      goto RECEIVE_FILE_CONTENTS;
    } /* else if (status == -1) */
    else if (status == 0) {
      lock_cerr_mutex();
      cerr << "[Thread" << thread_ctr << "] WARNING! In" <<
        "'Scan_Parse_Parameter_Type::receive_file':" << endl <<
        "File"

```

```

local_filename << "'already_exists_and_overwrite'=='false'." << endl <<
"Setting 'discard'=='true'." << endl;
unlock_cerr_mutex();
err_ret_val = 2;
++warnings_occurred;
discard = true;
goto RECEIVE_FILE_CONTENTS;
} /* else if (status == 0) */
} /* if (overwrite == false) */

```

240. Check whether *local_filename* is a path containing one or more directories. [LDF 2012.11.20.]

Log

[LDF 2012.11.20.] Added this section.

```

⟨ Scan_Parse_Parameter_Type :: receive_file definition 235 ⟩ +≡
local_filename = save_local_filename;
pos = local_filename.find_last_of("/");
if (pos != string::npos) {
    local_path += local_filename.substr(0, pos);
    local_filename.erase(0, pos + 1);
#endif /* DEBUG_COMPILE */
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "local_path==" << local_path << endl << "local_filename==" << local_filename << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

241. Test whether *local_path* already exists below the current working directory. [LDF 2012.11.20.]

Log

[LDF 2012.11.20.] Added this section.

```

⟨ Scan_Parse_Parameter_Type :: receive_file definition 235 ⟩ +≡
string save_local_path = local_path;
if (pwd_str) {
    local_path.insert(0, "/");
    local_path.insert(0, pwd_str);
}
#endif /* DEBUG_COMPILE */
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "local_path==" << local_path << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
errno = 0;
status = access(local_path.c_str(), F_OK);

```

242.

```
<Scan_Parse_Parameter_Type::receive_file definition 235> +≡
    if (status ≡ -1 ∧ errno ≡ ENOENT) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ "[Thread_"
        cerr ≪ thread_ctr ≪ "]_In_"
        cerr ≪ "Scan_Parse_Parameter_Type::receive_file:'"
        endl ≪ "'access'_"
        cerr ≪ "returned_-1_and_"
        cerr ≪ "'errno'==_"
        cerr ≪ "'ENOENT'." ≪ endl ≪ "Path_"
        cerr ≪ "'"
        local_path ≪ "'_"
        cerr ≪ "doesn't_already_exist._"
        cerr ≪ "Will_create." ≪ endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

243.

```
<Scan_Parse_Parameter_Type::receive_file definition 235> +≡
    temp_path = save_local_path;
    temp_path.insert(0, "/");
    for ( ; ; ) {
        pos = temp_path.find_last_of("/");
        if (pos ≡ string::npos) break;
        temp_str = temp_path.substr(pos + 1);
#ifndef 0 /* 1 */
        cerr ≪ "temp_path.substr(pos+1)=="
        cerr ≪ temp_path.substr(pos + 1) ≪ endl;
#endif
        if (!temp_str.empty())
            dir_list.push_front(temp_str);
        temp_path.erase(pos);
#ifndef 0 /* 1 */
        cerr ≪ "temp_path=="
        cerr ≪ temp_path ≪ endl;
#endif
    }
    /* for */
    temp_path = pwd_str;
```

244.

```
<Scan_Parse_Parameter_Type::receive_file definition 235> +≡
    for ( deque<string>::const_iterator iter = dir_list.begin();
          iter ≠ dir_list.end(); ++iter ) { temp_path += "/";
    temp_path += *iter;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ "*iter=="
        cerr ≪ *iter ≪ endl ≪ "temp_path=="
        cerr ≪ temp_path ≪ endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

245.

Log

[LDF 2012.11.29.] Now calling `::mkdir` instead of `mkdir`. This is necessary because I've added a member function `Scan_Parse_Parameter_Type::mkdir`.

```

⟨ Scan_Parse_Parameter_Type::receive_file definition 235 ⟩ +≡
    errno = 0;
    status = ::mkdir(temp_path.c_str(), S_IRWXU | S_IRGRP | S_IXGRP);
    if (status == -1 & errno == EEXIST) {
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "[Thread" << thread_ctr << "] In" <<
                "'Scan_Parse_Parameter_Type::receive_file':"
                << endl <<
                "'mkdir' failed, returning -1 and 'errno' == 'EEXIST'." << endl << "Path"
                << temp_path << "' already exists. Not creating." << endl << "Continuing."
                << endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        continue;
    } /* if (status == -1 & errno == EEXIST) */
else if (status == -1) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] ERROR! In"
        << "'Scan_Parse_Parameter_Type::receive_file':"
        << endl <<
        "'mkdir' failed, returning -1: " << strerror(errno) << endl <<
        "Setting 'discard' == 'true'." << endl;
    unlock_cerr_mutex();
    err_ret_val = 3;
    ++errors_occurred;
    discard = true;
    goto RECEIVE_FILE_CONTENTS;
} /* if (status == -1) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] In"
            << "'Scan_Parse_Parameter_Type::receive_file':"
            << endl <<
            "'mkdir' succeeded. Created path " << temp_path << "'"
            << "successfully." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
/* for */
/* if (status == -1 & errno == ENOENT) */
}

```

246.

```
< Scan_Parse_Parameter_Type::receive_file definition 235 > +≡
else
  if (status ≡ -1) {
    lock_cerr_mutex();
    cerr ≪ "[Thread" ≪ thread_ctr ≪ "] In" ≪
      "'Scan_Parse_Parameter_Type::receive_file':" ≪ endl ≪
      "'access' returned -1 and 'errno' != 'ENOENT':" ≪ endl ≪ strerror(errno) ≪ endl ≪
      "Setting 'discard' == 'true' ." ≪ endl;
    unlock_cerr_mutex();
    err_ret_val = 4;
    ++warnings_occurred;
    discard = true;
    goto RECEIVE_FILE_CONTENTS;
} /* if (status ≡ -1) */
```

247.

```
< Scan_Parse_Parameter_Type::receive_file definition 235 > +≡
else
  if (status ≡ 0) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "[Thread" ≪ thread_ctr ≪ "] In" ≪
      "'Scan_Parse_Parameter_Type::receive_file':" ≪ endl ≪ "Path" ≪
      local_path ≪ "'already exists. Not creating.'" ≪ endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* else if (status ≡ 0) */
} /* if (pos ≠ string::npos) */
```

248.

Log

[LDF 2012.11.19.] Added this section.

```
< Scan_Parse_Parameter_Type::receive_file definition 235 > +≡
RECEIVE_FILE_CONTENTS:
  if (!local_path.empty()) {
    local_filename.insert(0, "/");
    local_filename.insert(0, local_path);
  }
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "local_filename" ≪ local_filename ≪ endl ≪ "local_path" ≪ local_path ≪ endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  if (new_local_filename_ptr ≠ 0) *new_local_filename_ptr = local_filename;
```

249.

```
<Scan_Parse_Parameter_Type::receive_file definition 235> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "About_to_open_file:" << local_filename << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    if (!discard & !gen_temp_file) {
        out_strm.open(local_filename.c_str(), ios_base::out | ios_base::binary);
        if (!out_strm.is_open()) {
            lock_cerr_mutex();
            cerr << "[Thread" << thread_ctr << "] In" <<
                "Scan_Parse_Parameter_Type::receive_file:" << endl << "Failed_to_open_file" <<
                local_filename << endl << "Will_discard_file_contents." << endl;
            unlock_cerr_mutex();
            err_ret_val = 5;
            ++errors_occurred;
            discard = true;
        } /* if (!out_strm.is_open()) */
        else {
#ifndef DEBUG_COMPILE
            if (DEBUG) {
                lock_cerr_mutex();
                cerr << "[Thread" << thread_ctr << "] In" <<
                    "Scan_Parse_Parameter_Type::receive_file:" << endl << "Opened_file" <<
                    local_filename << " successfully." << endl;
                unlock_cerr_mutex();
            } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        } /* else */
    } /* if (!discard & !gen_temp_file) */
```

250.

```

⟨ Scan_Parse_Parameter_Type::receive_file definition 235 ⟩ +≡
else      /* discard == true ∨ gen_temp_file == true */
{
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread]" << thread_ctr << "] In " << 'Scan_Parse_Parameter_Type::receive_file'
            e' :" << endl << 'discard' == 'true' || 'gen_temp_file' == 'true' ." << endl <<
                "Will create temporary file to store file contents" << "sent from server." << endl;
        unlock_cerr_mutex();
    }      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */
errno = 0;
fd = mkstemp(temp_filename);
if (fd == -1) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] ERROR! In "
        "'Scan_Parse_Parameter_Type::receive_file':" << endl <<
        "'mkstemp' failed, returning -1:" << endl << strerror(errno) << endl <<
        "Can't store file contents to temporary file." << "Continuing." << endl;
    unlock_cerr_mutex();
    if (temp_filename_ptr) *temp_filename_ptr = "";
    strcpy(temp_filename, "");
}      /* if (fd == -1) */
else      /* fd != -1 */
{
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] In "
            "'Scan_Parse_Parameter_Type::receive_file':" << endl <<
            "'mkstemp' succeeded. " << temp_filename << temp_filename << ". " << endl;
        unlock_cerr_mutex();
    }      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */
close(fd);
fd = 0;
out_strm.open(temp_filename, ios_base::out | ios_base::binary);
if (!out_strm.is_open()) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] ERROR! In "
        "'Scan_Parse_Parameter_Type::receive_file':" << endl <<
        "Failed to open file " << temp_filename << ". " << endl <<
        "Will discard file contents." << endl;
    unlock_cerr_mutex();
    if (temp_filename_ptr) *temp_filename_ptr = "";
    strcpy(temp_filename, "");
}      /* if (!out_strm.is_open()) */
else {
    if (temp_filename_ptr) *temp_filename_ptr = temp_filename;
#endif DEBUG_COMPILE
    if (DEBUG) {

```

```
lock_cerr_mutex();
cerr << "[Thread_"
     << thread_ctr << "]_In_"
     << "Scan_Parse_Parameter_Type::receive_file':"
     << endl <<
"Opened_file"
     << temp_filename << "'_successfully."
     << endl <<
"Will_store_file_contents_sent_from_server_in_this_file."
     << endl;
unlock_cerr_mutex();
#endif /* if (DEBUG) */
} /* DEBUG_COMPILE */
} /* else */
} /* else (fd != -1) */
} /* else (discard == true || gen_temp_file == true) */
```

251.

Log

[LDF 2012.12.13.] BUG FIX: Now calling `ofstream::write` to write `buffer` to `out_strm`. `write` uses unformatted output. Previously, I used `operator<<`, which uses formatted output, which didn't work if `buffer` was full, i.e., without a NULL byte at the end.

```

⟨ Scan_Parse_Parameter_Type::receive_file definition 235 ⟩ +≡
    char buffer[BUFFER_SIZE];
    do {
        memset(buffer, 0, BUFFER_SIZE);
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            cerr << "In 'Scan_Parse_Parameter_Type::receive_file':";
            if (remote_connection) cerr << "Calling 'gnutls_record_recv'..." << endl;
            else cerr << "Calling 'recv'..." << endl;
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        if (remote_connection == true) {
            status = gnutls_record_recv(session, buffer, BUFFER_SIZE);
        }
        else {
            status = recv(sock, buffer, BUFFER_SIZE, 0);
        }
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "[Thread" << thread_ctr << "] In 'receive_file':"
                << endl;
            if (remote_connection) cerr << "'gnutls_record_recv'";
            else cerr << "'recv'";
            cerr << "returned" << status << "." << endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        if (status > 0) {
#ifndef DEBUG_COMPILE
            if (DEBUG) {
                lock_cerr_mutex();
                cerr << "[Thread" << thread_ctr << "] In 'receive_file':"
                    << endl <<
                    "Received text from client: buffer==" << endl;
                fwrite(buffer, 1, status, stderr);
                unlock_cerr_mutex();
            } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
            out_strm.write(buffer, status);
        } /* if (status > 0) */
    } /* do */ while (status == BUFFER_SIZE);
    if (out_strm.is_open()) out_strm.close();
    if (discard) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::receive_file':"
            << endl << "'discard'" << endl << discard << endl << "'gen_temp_file'" << endl << gen_temp_file << endl;
    }
}

```

```

if (temp_filename ∧ strlen(temp_filename) > 0) {
    cerr ≪ "Wrote_file_contents_sent_from_server_to_temporary_file" ≪ '"' ≪
        temp_filename ≪ '.' ≪ endl;
}
else cerr ≪ "Discarded_file_contents_sent_from_server." ≪ endl;
cerr ≪ "Exiting_function_unsuccessfully_with_return_value" ≪ err_ret_val ≪ '.' ≪ endl;
unlock_cerr_mutex();
return err_ret_val;
} /* if (discard) */

```

252.

```

⟨ Scan_Parse_Parameter_Type::receive_file definition 235 ⟩ +≡
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "[Thread" ≪ thread_ctr ≪ "] " ≪ "Exiting‘Scan_Parse_Parameter_Type::rec\
        eive_file’" ≪ "successfully_with_return_value_0." ≪ endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
return 0; } /* Scan_Parse_Parameter_Type::receive_file */

```

253. Get metadata. [LDF 2012.10.09.]

Log

[LDF 2012.10.09.] Added this function.

[LDF 2012.12.21.] Added the optional arguments **bool send_response**, **bool do_output**, **bool do_irods_user_metadata** and **vector<Dublin_Core_Metadata_Type> *dc_metadata_type_vector_ptr**. Adapting this function for use with **Scan_Parse_Parameter_Type::add_metadata**. For this use, *get_metadata* should store the data from the database in **Dublin_Core_Metadata_Type** objects in ***dc_metadata_type_vector_ptr** but neither output XML code to a temporary file nor send responses to the peer.

[LDF 2012.12.31.] Added optional argument **unsigned int options** with the default 0.

[LDF 2013.01.04.] Added code for sending responses in the error cases.

[LDF 2013.02.13.] Moved the definition of this function from this file (**scprpmtp.web**) to **spptfnc1.web**.

[LDF 2013.05.23.] Replaced the optional argument **bool send_response = true** with the optional arguments **char *buffer_ptr = 0** and **size_t buffer_size = 0**.

⟨ Scan_Parse_Parameter_Type function declarations 38 ⟩ +≡

```

int get_metadata(string filename, unsigned int flags, int *ctr = 0, unsigned int options = 0, char
    *buffer_ptr = 0, size_t buffer_size = 0, bool do_output = true, bool do_irods_user_metadata = true,
    vector<Dublin_Core_Metadata_Type> *dc_metadata_type_vector_ptr = 0);

```

254. Get handle. (*get_handle*). [LDF 2012.10.15.]

Log

[LDF 2012.10.15.] Added this function.

[LDF 2013.02.13.] Moved the definition of this function from this file (**scprpmtp.web**) to **spptfnc1.web**.

⟨ Scan_Parse_Parameter_Type function declarations 38 ⟩ +≡

```

int get_handle(string s, unsigned int flags, string filename_1 = "");

```

255. Change working directory. (*cd*). [LDF 2012.11.23.]

Log

[LDF 2012.11.23.] Added this function.

[LDF 2012.11.27.] Made argument **string** *dir* optional with default "".

[LDF 2013.04.03.] Made argument **string** *dir* non-optional again. Added arguments **char *buffer_ptr** and **unsigned int buff_size**. Rewrote this function. Now calling it in *exchange_data_with_client* instead of *yyparse*.

[LDF 2013.04.03.] BUG FIX: Removed code for setting the environment variable *irodsCwd*. This won't work properly, because the environment is shared by all threads!

⟨ **Scan_Parse_Parameter_Type** function declarations 38 ⟩ +≡

```
int cd(string dir, char *buffer_ptr, unsigned int buff_size);
```

256.

⟨ **Scan_Parse_Parameter_Type**::*cd* definition 256 ⟩ ≡

```
int Scan_Parse_Parameter_Type::cd(string dir, char *buffer_ptr, unsigned int buff_size){ bool
    DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    size_t pos;
    stringstream temp_strm;
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] " "Entering " "Scan_Parse_Parameter_Type::cd" " <<
            endl << "'dir'" " == " << dir << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
}
```

See also sections 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, and 272.

This code is used in section 305.

257.

Log

[LDF 2013.04.04.] Added this section.

⟨ **Scan_Parse_Parameter_Type**::*cd* definition 256 ⟩ +≡

```
if (buffer_ptr == 0) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] " "ERROR! " "In " "Scan_Parse_Parameter_Type::cd" " <<
        endl << "'char*buffer_ptr'" " == " "NULL. " "This isn't permitted." << endl <<
        "Exiting " "function " "unsuccessfully " "with " "return " "value " "3." << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    return 3;
} /* if (buff_size < 1024) */
```

258.

Log

[LDF 2013.04.04.] Added this section.

```
<Scan_Parse_Parameter_Type::cd definition 256> +≡
if (buff_size < 1024) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] ERROR! In 'Scan_Parse_Parameter_Type::cd':"
        << endl << "'buff_size' < 1024. This isn't permitted."
        << endl << "Exiting function unsuccessfully with return value 3." << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    return 3;
} /* if (buff_size < 1024) */
```

259.

```
<Scan_Parse_Parameter_Type::cd definition 256> +≡
memset(buffer_ptr, 0, buff_size);
if (dir.empty()) {
    dir = irods_homedir;
} /* Erase trailing slash, if any. [LDF 2012.11.27.] */
else if (dir[dir.size() - 1] == '/') {
    dir.erase(dir.size() - 1);
#ifndef 0 /* 1 */
    cerr << "dir after erasing trailing slash: " << dir << endl;
#endif
}
pos = dir.find("$HOME");
if (pos != string::npos) {
    dir.replace(pos, 5, irods_homedir);
#ifndef 0 /* 1 */
    cerr << "dir after replace == " << dir << endl;
#endif
}
```

260.

```
<Scan_Parse_Parameter_Type::cd definition 256> +≡
temp_strm.str("");
temp_strm << "env_irodsEnvFile=" << irods_env_filename << " "
        << "icd-v" << dir <<
        "' ; r=$?; echo $r";
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] In 'Scan_Parse_Parameter_Type::cd':"
            << endl << "'temp_strm.str()' == "
            << temp_strm.str() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
FILE *fp = popen(temp_strm.str().c_str(), "r");
```

261.

```
< Scan_Parse_Parameter_Type :: cd definition 256 > +≡
  if (fp == 0) {
    lock_cerr_mutex();
    cerr << "[Thread:" << thread_ctr << "] In 'Scan_Parse_Parameter_Type::cd':"
    << endl << "'popen' failed, returning NULL. 'icd' command failed."
    << endl << "Can't change iRODS current working directory to "
    << dir << "."
    << endl << "Exiting function unsuccessfully with return value 1."
    << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    temp_strm.str("");
    temp_strm << "CD RESPONSE 2 \""
    << dir << "\" "
    << "\" "
    << irods_current_dir << "\" "
    << "\" "
    << irods_current_dir << "\" "
    << "\" "
    << "'popen' failed\"";
    strcpy(buffer_ptr, temp_strm.str().c_str());
    ++errors_occurred;
    return 1;
} /* if (fp == 0) */
```

262.

```
< Scan_Parse_Parameter_Type :: cd definition 256 > +≡
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "[Thread:" << thread_ctr << "] In 'Scan_Parse_Parameter_Type::cd':"
      << endl << "'popen' succeeded."
      << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

263.

```
< Scan_Parse_Parameter_Type :: cd definition 256 > +≡
  char buffer[1024];
  memset(buffer, 0, 1024);
  status = fread(buffer, 1, 1024, fp);
```

264.

```

⟨ Scan_Parse_Parameter_Type :: cd definition 256 ⟩ +≡
  if (status ≡ 0) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] ERROR! In 'Scan_Parse_Parameter_Type::cd':"
    endl << "'fread' read_0_bytes:" << "'feof(fp)' ==" << feof(fp) << endl <<
    "'ferror(fp)' ==" << ferror(fp) << endl << "'icd' failed." << endl <<
    "iRODS current_directory is in an undefined state." << endl <<
    "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    temp_strm.str("");
    temp_strm << "CD RESPONSE 3" << dir << "\\" << "\\" <<
    irods_current_dir << "\\" << "\\" << irods_current_dir << "\\" <<
    "\Failed to read output and/or exit status of 'icd' command\"";
    pclose(fp);
    fp = 0;
    strcpy(buffer_ptr, temp_strm.str().c_str());
    ++errors_occurred;
    return 1;
} /* if (status ≡ 0) */

```

265.

```

⟨ Scan_Parse_Parameter_Type :: cd definition 256 ⟩ +≡
  else
    if (status ≡ 1024) {
      lock_cerr_mutex();
      cerr << "[Thread" << thread_ctr << "] ERROR! In 'Scan_Parse_Parameter_Type::cd':"
      endl << "'fread' read_1024_bytes. This exceeds the maximum number of"
      endl << "bytes for the output of 'icd'." << endl <<
      "iRODS current_directory is in an undefined state." << endl <<
      "Exiting function unsuccessfully with return value 1." << endl;
      unlock_cerr_mutex();
      ++errors_occurred;
      temp_strm.str("");
      temp_strm << "CD RESPONSE 4" << dir << "\\" << "\\" <<
      irods_current_dir << "\\" << "\\" << irods_current_dir << "\\" <<
      "\'icd' command produced too much output (>= 1024 bytes)\";
      strcpy(buffer_ptr, temp_strm.str().c_str());
      pclose(fp);
      fp = 0;
      ++errors_occurred;
      return 1;
} /* else if (status ≡ 1024) */

```

266.

```
<Scan_Parse_Parameter_Type::cd definition 256> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] In " 'Scan_Parse_Parameter_Type::cd' : " << endl <<
            "'fread'" << "read" << "status" << "bytes." << endl << "'buffer'" == " << endl << buffer << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    while (buffer[strlen(buffer) - 1] == '\n') buffer[strlen(buffer) - 1] = '\0';
    buffer[strlen(buffer)] = '\n'; /* Put one newline back on the end of buffer. There will always be
        enough room for this newline, because the echo command will have produced one. We therefore
        don't need to test for this. [LDF 2012.11.23.] */
#endif /* 0 */ /* 1 */
    cerr << "After removing trailing newlines: " << buffer << (delimited with single quotes) <<
        endl << "," << buffer << "," << endl;
#endif
```

267.

```
<Scan_Parse_Parameter_Type::cd definition 256> +≡
    string output_str = buffer;
    long int ret_val;
    pos = output_str.find_last_of ('\n', output_str.size() - 2);
    errno = 0;
    if (pos == string::npos) {
#ifndef 0 /* 1 */
        cerr << "Only one line" << endl;
#endif
        ret_val = strtol(output_str.c_str(), 0, 10);
        output_str = "";
    }
    else {
#ifndef 0 /* 1 */
        cerr << "More than one line." << endl;
#endif
        output_str.erase(0, pos);
        ret_val = strtol(output_str.c_str(), 0, 10);
        output_str = buffer;
        output_str.erase(pos);
#ifndef 0 /* 1 */
        cerr << "output_str after second erase == " << output_str << endl;
#endif
#endif
    } /* else */
```

268.

```
<Scan_Parse_Parameter_Type::cd definition 256> +≡
#ifndef 0
    cerr << "ret_val==\" << ret_val << endl;
#endif
    if (ret_val == LONG_MIN || ret_val == LONG_MAX) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] ERROR! In 'Scan_Parse_Parameter_Type::cd' <<
            endl << 'strtol' failed, returning";
        if (ret_val == LONG_MIN) cerr << "'LONG_MIN': " << endl;
        else cerr << "'LONG_MAX': " << endl;
        cerr << strerror(errno) << endl << "Failed to read return value of 'icd' command, executed in pipe." << endl <<
            "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        ++errors_occurred;
        temp_strm.str("");
        temp_strm << "CD RESPONSE" << dir << "\\" << irods_current_dir << "\\" << "\\" <<
            irods_current_dir << "\\" << "\"Server-side error: Failed to read return value of 'icd' command, executed in pipe\"";
#endif /* 0 */
    cerr << "temp_strm.str() == " << temp_strm.str() << endl;
#endif
    strcpy(buffer_ptr, temp_strm.str().c_str());
    pclose(fp);
    fp = 0;
    ++errors_occurred;
    return 1;
} /* if (ret_val == LONG_MIN || ret_val == LONG_MAX) */
```

269.

```

⟨ Scan_Parse_Parameter_Type :: cd definition 256 ⟩ +≡
  if (ret_val ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ "[Thread" ≪ thread_ctr ≪ "] ERROR! In 'Scan_Parse_Parameter_Type::cd':"
    endl ≪ "'icd' command, executed_in_pipe, failed, returning" ≪ ret_val ≪ "."
    endl;
    if (¬output_str.empty()) cerr ≪ "Error output:" ≪ output_str ≪ endl;
    cerr ≪ "Exiting function unsuccessfully with return value 1."
    endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    temp_strm.str("");
    temp_strm ≪ "CD RESPONSE" ≪ dir ≪ "\\" ≪ "\\" ≪
    irods_current_dir ≪ "\\" ≪ "\\" ≪ irods_current_dir ≪ "\\" ≪
    "\"Server-side error: 'icd' command, executed_in_pipe, " ≪ "failed, returning"
    ret_val ≪ ".";
    if (¬output_str.empty())
      temp_strm ≪ "\n" ≪ right ≪ setw(35) ≪ " " ≪ "Error output:" ≪ output_str ≪ endl;
    temp_strm ≪ "\n";
  #if 0 /* 1 */
    cerr ≪ "temp_strm.str() == " ≪ temp_strm.str() ≪ endl;
  #endif
    strcpy(buffer_ptr, temp_strm.str().c_str());
    pclose(fp);
    fp = 0;
    ++errors_occurred;
    return 1;
  } /* if (ret_val ≠ 0) */
  #if DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr ≪ "[Thread" ≪ thread_ctr ≪ "] In 'Scan_Parse_Parameter_Type::cd':"
      endl ≪
        "'icd' command, executed_in_pipe, succeeded, returning 0."
      endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
  #endif /* DEBUG_COMPILE */
  pclose(fp);
  fp = 0;

```

270. If *dir* starts with a slash, it is interpreted as an absolute path. Otherwise, it is interpreted as a path below *irods_current_dir*. That is, *dir* is in effect appended to *irods_current_dir*.

Log

[LDF 2012.11.27.] Added this section.

```
<Scan_Parse_Parameter_Type::cd definition 256> +≡
    temp_strm.str("");
    string save_irods_current_dir = irods_current_dir;
    if (dir[0] == '/') temp_strm << dir;
    else temp_strm << irods_current_dir << "/" << dir;
#ifndef 0 /* 1 */
    cerr << "temp_strm.str() == " << temp_strm.str() << endl;
#endif
    errno = 0;
    irods_current_dir = temp_strm.str();
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "irods_current_dir == " << irods_current_dir << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

271.

```
<Scan_Parse_Parameter_Type::cd definition 256> +≡
    temp_strm.str("");
    temp_strm << "CD_RESPONSE\b0\" << dir << "\b" << "b" << save_irods_current_dir << "\b" <<
        "\b" << irods_current_dir << "\b";
    if (!output_str.empty()) temp_strm << "\bCommand\boutput:\b" << output_str << "\b";
    else temp_strm << "\bSUCCESS\b(No\bcommand\boutput).\b";
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "temp_strm.str() == " << temp_strm.str() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    strcpy(buffer_ptr, temp_strm.str().c_str());
```

272.

```
<Scan_Parse_Parameter_Type::cd definition 256> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread\b" << thread_ctr << "]Exiting\bScan_Parse_Parameter_Type::cd\b" <<
            "successfully\bwith\breturn\bvalue\b0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; /* End of Scan_Parse_Parameter_Type::cd definition */
```

273. Create directory. (`mkdir`). [LDF 2012.11.29.]

The length of the string to be copied into `buffer_ptr` is tested against `buffer_size` before copying. If the string is too long, `buffer_ptr` is set to the empty string. However, this should never occur. Unlike the output of `ils`, which depends on the number of existing iRODS objects, the output of `imkdir` is not likely to be very long. Currently, this function is only called in `exchange_data_with_client` where the `buffer_size` argument is the preprocessor macro `BUFFER_SIZE` defined in `gblvrb1.web`. As of this date, the value of `BUFFER_SIZE` is 2048, which should be much more than enough for the output of `imkdir`. [LDF 2013.04.25.]

Log

[LDF 2012.11.29.] Added this function.

[LDF 2012.11.30.] Removed argument `string flags = ""`. Now using data member `Scan_Parse_Parameter_Type::ic` instead.

[LDF 2013.04.25.] Added code for disabling thread cancellation and restoring the prior thread cancellation state before exiting. Removed code that tests the value of `icommands`: Currently, only icommands are used for accessing the iRODS server.

[LDF 2013.04.25.] Replaced the argument `string flags` with the arguments `Response_Type &response`, `char *buffer_ptr` and `size_t buffer_size`.

`< Scan_Parse_Parameter_Type function declarations 38 > +≡`

`int mkdir(Response_Type &response, char *buffer_ptr, size_t buffer_size);`

274.

`< Scan_Parse_Parameter_Type::mkdir definition 274 > ≡`

```
int Scan_Parse_Parameter_Type::mkdir(Response_Type &response, char *buffer_ptr, size_t
                                     buffer_size){ bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    size_t pos;
    stringstream temp_strm;
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] " "Entering " 'Scan_Parse_Parameter_Type::mkdir' . " <<
            endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
    string directory_list;
    for (vector<string>::const_iterator iter = response.string_vector.begin();
         iter != response.string_vector.end(); ++iter) {
        directory_list += *iter;
        directory_list += " ";
    } /* for */
    directory_list.erase(directory_list.length() - 1); /* Remove trailing space. [LDF 2013.04.25.] */
    response.string_vector.clear();
```

See also sections 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, and 288.

This code is used in section 305.

275. Disable thread cancellation. [LDF 2013.04.25.]

Log

[LDF 2013.04.25.] Added this section.

```
<Scan_Parse_Parameter_Type::mkdir definition 274> +≡
    int old_pthread_cancel_state = 0;
    status = pthread_setcancelstate(PTHREAD_CANCEL_DISABLE, &old_pthread_cancel_state);
    if (status != 0) {
        lock_cerr_mutex();
        cerr << "[Thread]" << thread_ctr << "] In 'Scan_Parse_Parameter_Type::mkdir':"
            << endl << "'pthread_setcancelstate' failed, returning" << status << ":" << endl <<
            strerror(status) << endl << "Failed to disable thread cancellation." << endl <<
            "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        ++errors_occurred;
        temp_strm.str("");
        temp_strm << "MKDIR_RESPONSE\7\" << directory_list << "\" << "\\" <<
            "\'pthread_setcancelstate\' failed";
        if (temp_strm.str().length() < buffer_size) strcpy(buffer_ptr, temp_strm.str().c_str());
        else strcpy(buffer_ptr, "");
        ++errors_occurred;
        return 1;
    } /* if (status != 0) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread]" << thread_ctr << "] In 'Scan_Parse_Parameter_Type::mkdir':"
            << endl << "'pthread_setcancelstate' succeeded, returning 0."
            << endl << "Disabled thread cancellation successfully." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

276.

```

⟨ Scan_Parse_Parameter_Type :: mkdir definition 274 ⟩ +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread_"
        << thread_ctr << "]_In_`Scan_Parse_Parameter_Type::mkdir':"
        << endl <<
        "'icommands'" == 'true' . "
        << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    string new_flags;
    pos = response.flags.find("-p");
    if (pos != string::npos) new_flags = "-p";
    pos = response.flags.find("-v");
    if (pos != string::npos) new_flags = "-v";
    pos = response.flags.find("-V");
    if (pos != string::npos) new_flags = "-V";
    response.flags = "";
    temp_strm.str("");
    temp_strm << "env_irodsEnvFile=" << irods_env_filename << " "
    << "imkdir" << new_flags <<
    directory_list << "2>&1;echo$?";
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "temp_strm.str() == "
        << temp_strm.str() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
FILE *fp = popen(temp_strm.str().c_str(), "r");

```

277.

```

⟨ Scan_Parse_Parameter_Type :: mkdir definition 274 ⟩ +≡
    if (fp == 0) {
        lock_cerr_mutex();
        cerr << "[Thread_"
        << thread_ctr << "]_ERROR!_In_`Scan_Parse_Parameter_Type::mkdir':"
        << endl <<
        "'popen' failed, returning NULL._`imkdir' command failed."
        << endl <<
        "Failed to create new directory or directories." << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        ++errors_occurred;
        temp_strm.str("");
        temp_strm << "MKDIR_RESPONSE" << directory_list << "\n"
        << "\n" << "\n" <<
        "\n`popen' failed\n";
        if (temp_strm.str().length() < buffer_size) strcpy(buffer_ptr, temp_strm.str().c_str());
        else strcpy(buffer_ptr, "");
        pthread_setcancelstate(old_thread_cancel_state, 0);
        ++errors_occurred;
        return 1;
    } /* if (fp == 0) */

```

278.

```
<Scan_Parse_Parameter_Type::mkdir definition 274> +≡
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "[Thread_"
            << thread_ctr << "]_In_`Scan_Parse_Parameter_Type::mkdir':"
            << endl <<
                " `popen`_succeeded."
            << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

279.

```
<Scan_Parse_Parameter_Type::mkdir definition 274> +≡
char buffer[1024];
memset(buffer, 0, 1024);
status = fread(buffer, 1, 1024, fp);
```

280.

```
<Scan_Parse_Parameter_Type::mkdir definition 274> +≡
if (status ≡ 0) {
    lock_cerr_mutex();
    cerr << "[Thread_"
    << thread_ctr << "]_ERROR!_In_`Scan_Parse_Parameter_Type::mkdir':"
    << endl <<
        " `fread`_read_0_bytes:"
        << " `feof(fp)`_==_"
        << feof(fp) << endl <<
        " `ferror(fp)`_==_"
        << ferror(fp) << endl <<
        " `imkdir`_failed."
        << endl <<
        "Exiting_function_unsuccessfully_with_return_value_1."
        << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    temp_strm.str("");
    temp_strm << "MKDIR_RESPONSE_3\\\""
    << directory_list << "\\\""
    << "\\\""
    << "\\\"Failed_to_read_output_and/or_exit_status_of_imkdir_command\\\"";
    if (temp_strm.str().length() < buffer_size) strcpy(buffer_ptr, temp_strm.str().c_str());
    else strcpy(buffer_ptr, "");
    pclose(fp);
    pthread_setcancelstate(old(pthread_cancel_state, 0));
    ++errors_occurred;
    return 1;
} /* if (status ≡ 0) */
```

281.

```

⟨ Scan_Parse_Parameter_Type :: mkdir definition 274 ⟩ +≡
else
  if (status ≡ 1024) {
    lock_cerr_mutex();
    cerr ≪ "[Thread]" ≪ thread_ctr ≪ "] ERROR! In 'Scan_Parse_Parameter_Type::mkdir':"
    endl ≪ "'fread' read 1024 bytes. This exceeds the maximum number of
    bytes for the output of 'imkdir'." ≪ endl ≪
    "Exiting function unsuccessfully with return value 1." ≪ endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    temp_strm.str("");
    temp_strm ≪ "MKDIR_RESPONSE" ≪ directory_list ≪ "\n" ≪ "\n\n" ≪
    "\n'mkdir' command produced too much output (>= 1024 bytes)\n";
    if (temp_strm.str().length() < buffer_size) strcpy(buffer_ptr, temp_strm.str().c_str());
    else strcpy(buffer_ptr, "");
    pclose(fp);
    fp = 0;
    pthread_setcancelstate(old(pthread_cancel_state, 0));
    ++errors_occurred;
    return 1;
} /* else if (status ≡ 1024) */

```

282.

```

⟨ Scan_Parse_Parameter_Type :: mkdir definition 274 ⟩ +≡
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "[Thread]" ≪ thread_ctr ≪ "] In 'Scan_Parse_Parameter_Type::mkdir':"
    endl ≪ "'fread' read " ≪ status ≪ " bytes." ≪ endl ≪ "'buffer'" ≪ endl ≪ buffer ≪ endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  while (buffer[strlen(buffer) - 1] ≡ '\n') buffer[strlen(buffer) - 1] = '\0';
  buffer[strlen(buffer)] = '\n'; /* Put one newline back on the end of buffer. There will always be
  enough room for this newline, because the echo command will have produced one. We therefore
  don't need to test for this. [LDF 2012.11.23.] */
#endif /* 1 */
  cerr ≪ "After removing trailing newlines: " ≪ buffer ≪ "(delimited with single quotes)" ≪
  endl ≪ "\n" ≪ buffer ≪ "\n" ≪ endl;
#endif

```

283.

```
<Scan_Parse_Parameter_Type::mkdir definition 274> +≡
  string output_str = buffer;
  long int ret_val;
  pos = output_str.find_last_of ('\n', output_str.size() - 2);
  errno = 0;
  if (pos == string::npos) {
#ifndef 0 /* 1 */
    cerr << "Only\u00e7one\u00e7line" << endl;
#endif
    ret_val = strtol(output_str.c_str(), 0, 10);
    output_str = "";
}
else {
#ifndef 0 /* 1 */
    cerr << "More\u00e7than\u00e7one\u00e7line." << endl;
#endif
    output_str.erase(0, pos);
    ret_val = strtol(output_str.c_str(), 0, 10);
    output_str = buffer;
    output_str.erase(pos);
#ifndef 0 /* 1 */
    cerr << "output\u00e7str\u00e7after\u00e7second\u00e7erase\u00e7==\u00e7" << endl << output_str << endl <<
        "(End\u00e7of\u00e7'output\u00e7str')" << endl;
#endif
#endif
} /* else */
```

284.

```

⟨Scan_Parse_Parameter_Type::mkdir definition 274⟩ +≡
#ifndef /* 1 */
    cerr << "ret_val==\" << ret_val << endl;
#endif
    if (ret_val == LONG_MIN || ret_val == LONG_MAX) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] ERROR! In 'Scan_Parse_Parameter_Type::mkdir':"
            << endl << "'strtol' failed, returning";
        if (ret_val == LONG_MIN) cerr << "'LONG_MIN': " << endl;
        else cerr << "'LONG_MAX': " << endl;
        cerr << strerror(errno) << endl << "Failed to read return value of command, "
            << "imkdir" << endl << "in pipe." << endl <<
            "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        ++errors_occurred;
        temp_strm.str("");
        temp_strm << "MKDIR_RESPONSE\5\" " << directory_list << "\" " << "\" " <<
            "\"Server-side error: Failed to read return value of command, "
            << "imkdir" << endl << "in pipe\"";
#endif /* 1 */
    cerr << "temp_strm.str() == " << temp_strm.str() << endl;
#endif
    if (temp_strm.str().length() < buffer_size) strcpy(buffer_ptr, temp_strm.str().c_str());
    else strcpy(buffer_ptr, "");
    pclose(fp);
    fp = 0;
    pthread_setcancelstate(old_pthread_cancel_state, 0);
    ++errors_occurred;
    return 1;
} /* if (ret_val == LONG_MIN || ret_val == LONG_MAX) */

```

285.

```

⟨ Scan_Parse_Parameter_Type :: mkdir definition 274 ⟩ +≡
  if (ret_val ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ "[Thread" ≪ thread_ctr ≪ "] ERROR! In 'Scan_Parse_Parameter_Type::mkdir':"
    endl ≪ "'imkdir' command, executed_in_pipe, failed, returning" ≪ ret_val ≪ "."
    endl;
    if (¬output_str.empty()) cerr ≪ "Error_output:" ≪ output_str ≪ endl;
    cerr ≪ "Exiting function unsuccessfully with return_value 1." ≪ endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    temp_strm.str("");
    temp_strm ≪ "MKDIR_RESPONSE" ≪ directory_list ≪ "\\" ≪ "\\" ≪ output_str ≪
      "\\" ≪ "\"Server-side_error: " ≪ "'imkdir' command, executed_in_pipe, "
      "failed, returning" ≪ ret_val ≪ "\"";
#define DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "temp_strm.str() == " ≪ temp_strm.str() ≪ endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  if (temp_strm.str().length() < buffer_size) strcpy(buffer_ptr, temp_strm.str().c_str());
  else strcpy(buffer_ptr, "");
  pclose(fp);
  fp = 0;
  pthread_setcancelstate(old(pthread_cancel_state, 0));
  ++errors_occurred;
  return 1;
} /* if (ret_val ≠ 0) */
#define DEBUG_COMPILE
else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "[Thread" ≪ thread_ctr ≪ "] In 'Scan_Parse_Parameter_Type::mkdir':"
    endl ≪ "'imkdir' command, executed_in_pipe, succeeded, returning 0." ≪ endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

286.

```

⟨ Scan_Parse_Parameter_Type :: mkdir definition 274 ⟩ +≡
  temp_strm.str("");
  temp_strm ≪ "MKDIR_RESPONSE" ≪ directory_list ≪ "\\" ≪ "\\" ≪ "Success";
  if (temp_strm.str().length() < buffer_size) strcpy(buffer_ptr, temp_strm.str().c_str());
  else strcpy(buffer_ptr, "");
  temp_strm.str("");

```

287. Restore prior thread cancellation state. [LDF 2013.04.25.]

Log

[LDF 2013.04.25.] Added this section.

```

⟨ Scan_Parse_Parameter_Type :: mkdir definition 274 ⟩ +≡
status = pthread_setcancelstate(old_thread_cancel_state, 0);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << "[Thread]" << thread_ctr << "] In 'Scan_Parse_Parameter_Type::mkdir':"
        endl << "'pthread_setcancelstate' failed, returning" << status << ":" << endl <<
        strerror(status) << endl << "Failed to restore prior thread cancellation state."
        endl << "Exiting function unsuccessfully with return value 3." << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    ++errors_occurred;
    return 3;
} /* if (status ≠ 0) */
#endif /* DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread]" << thread_ctr << "] In 'Scan_Parse_Parameter_Type::mkdir':"
        endl << "'pthread_setcancelstate' succeeded, returning 0." << endl <<
        "Restored prior thread cancellation state successfully." << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

288.

```

⟨ Scan_Parse_Parameter_Type :: mkdir definition 274 ⟩ +≡
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread]" << thread_ctr << "] Exiting 'Scan_Parse_Parameter_Type::mkdir'" <<
        "successfully with return value 0." << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
return 0; } /* End of Scan_Parse_Parameter_Type :: mkdir definition */

```

289. Add metadata. [LDF 2012.12.14.]

Log

[LDF 2012.12.14.] Added this function.

[LDF 2013.02.13.] Moved the definition of this function from this file (`scprpmtp.web`) to `spptfnc1.web`.

```

⟨ Scan_Parse_Parameter_Type function declarations 38 ⟩ +≡
int add_metadata(Response_Type &response);

```

290. Store Dublin Core metadata (*store_dc_metadata*). [LDF 2013.03.01.]

Log

[LDF 2013.03.01.] Added this function. It's defined in `spptfnc1.web`.

[LDF 2013.03.07.] Changed arguments: Now passing **Handle_Type** references instead of a string representing the handle and **unsigned long int** values representing the handle ID.

`<Scan_Parse_Parameter_Type function declarations 38> +≡`

```
int store_dc_metadata(const Response_Type &response, Handle_Type &irods_object_handle,
                      Handle_Type &dc_metadata_handle, bool force, string &irod_object_path);
```

291. Parse metadata.. [LDF 2012.12.14.]

Log

[LDF 2012.12.14.] Added this function.

[LDF 2013.02.13.] Moved the definition of this function from this file (`scprpmtp.web`) to `spptfnc1.web`.

`<Scan_Parse_Parameter_Type function declarations 38> +≡`

```
int parse_metadata(vector<Dublin_Core_Metadata_Type> &dc_metadata_vector, Response_Type
&response);
```

292. Fetch handle from database. [LDF 2013.02.07.]

Log

[LDF 2013.02.07.] Added this function.

[LDF 2013.02.13.] Moved the definition of this function from this file (`scprpmtp.web`) to `spptfnc1.web`.

[LDF 2013.02.28.] Added argument **Handle_Type** &*handle*. It replaces the old **Handle_Value_Type** & argument.

`<Scan_Parse_Parameter_Type function declarations 38> +≡`

```
int fetch_handle_from_database(unsigned long int handle_id, Handle_Type &handle, string
                               type = "");
```

293. Fetch handle from database. [LDF 2013.06.16.]

Log

[LDF 2013.06.16.] Added this function declaration.

`<Scan_Parse_Parameter_Type function declarations 38> +≡`

```
int fetch_handle_from_database(string handle_str, Handle_Type &handle, string type = "");
```

294. Fetch handles from database. [LDF 2013.02.07.]

Log

[LDF 2013.02.07.] Added this function.
 [LDF 2013.02.13.] Moved the definition of this function from this file (`scprpmtp.web`) to `spptfnc1.web`.
 [LDF 2013.02.28.] Added argument `vector<Handle_Type> &handle_vector`. It replaces the old `vector<Handle_Value> &argument`.

`<Scan_Parse_Parameter_Type function declarations 38> +≡`
`int fetch_handles_from_database(vector<unsigned long int> &handle_id_vector, vector<Handle_Type> &handle_vector, string type = "");`

295. Get highest value (`get_highest_value`). [LDF 2013.06.06.]

Log

[LDF 2013.06.06.] Added this function declaration. The definition is in `spptfnc1.web`.

`<Scan_Parse_Parameter_Type function declarations 38> +≡`
`static int get_highest_value(MYSQL *mysql_ptr, string table, string column, unsigned long int &val, bool incr = false);`

296. Show certificates. [LDF 2013.05.15.]

Log

[LDF 2013.05.15.] Added this function declaration. It is defined in `spptfnc1.web`.

`<Scan_Parse_Parameter_Type function declarations 38> +≡`
`int show_certificates(Response_Type &response, char *buffer, size_t buffer_size, string &filename);`

297. Show privileges. (`show_privileges`). [LDF 2013.05.22.]

This is a **static** function. The **unsigned int** value representing the privileges must be passed to it.
 [LDF 2013.05.22.]

Log

[LDF 2013.05.22.] Added this function declaration. The definition is in `spptfnc1.web`.

`<Scan_Parse_Parameter_Type function declarations 38> +≡`
`static int show_privileges(unsigned int privileges, ostream *strm = 0, bool verbose = false);`

298. Get input (`get_input`) declaration. [LDF 2013.07.11.]

Log

[LDF 2013.07.11.] Added this function declaration. The definition is in `spptfnc1.web`.

`<Scan_Parse_Parameter_Type function declarations 38> +≡`
`int get_input(void);`

- 299.** Mark iRODS objects for deletion. [LDF 2013.08.07.]

Log

[LDF 2013.08.07.] Added this function declaration. The definition is in **spptfnc1.web**.

{ Scan_Parse_Parameter_Type function declarations 38 } +≡

int mark_irods_objects_for_deletion(Response_Type &response, char *buffer_ptr, size_t buffer_size);

- 300.** Undelete files (*undelete_files*). [LDF 2013.08.16.]

Log

[LDF 2013.08.16.] Added this function declaration. The definition is in **spptfnc2.web**.

{ Scan_Parse_Parameter_Type function declarations 38 } +≡

int undelete_files(Response_Type &response, string thread_str = "");

301. Server action function declarations. [LDF 2013.05.27.]

Log

[LDF 2013.05.27.] [LDF 2013.05.29.] Added this section. The definitions of these functions are in `srvractn.web`. Pointers to these functions are inserted into `Scan_Parse_Parameter_Type::server_action_map` by `initialize_maps` (see above).

[LDF 2013.06.15.] Added declaration of `Scan_Parse_Parameter_Type::server_action_add_handle_value`.
[LDF 2013.07.04.] Added declaration of `Scan_Parse_Parameter_Type::server_action_delete_handle`.
[LDF 2013.08.07.] Added declaration of `Scan_Parse_Parameter_Type::server_action_mark_irods_objects_for_deletion`.
[LDF 2013.08.15.] Added declaration of `Scan_Parse_Parameter_Type::server_action_undelete_file`.
[LDF 2013.08.21.] Added declaration of `Scan_Parse_Parameter_Type::server_action_undelete_handle`.
[LDF 2013.08.30.] Added declaration of `Scan_Parse_Parameter_Type::server_action_delete_handle_value` and `Scan_Parse_Parameter_Type::server_action_undelete_handle_value`.

```
{ Scan_Parse_Parameter_Type function declarations 38 } +≡
int server_action_command_only(Response_Type &response);
int server_action_send_file(Response_Type &response);
int server_action_receive_put_file(Response_Type &response);
int server_action_receive_metadata_file(Response_Type &response);
int server_action_send_handle(Response_Type &response);
int server_action_ls(Response_Type &response);
int server_action_pwd(Response_Type &response);
int server_action_cd(Response_Type &response);
int server_action_mkdir(Response_Type &response);
int server_action_mark_irods_objects_for_deletion(Response_Type &response);
int server_action_get(Response_Type &response);
int server_action_send_metadata(Response_Type &response);
int server_action_end_server(Response_Type &response);
int server_action_sleep(Response_Type &response);
int server_action_show_certificate(Response_Type &response);
int server_action_get_metadata(Response_Type &response);
int server_action_get_handle(Response_Type &response);
int server_action_send_tan_list(Response_Type &response);
int server_action_process_pending(Response_Type &response);
int server_action_get_user_info(Response_Type &response);
int server_action_create_handle(Response_Type &response);
int server_action_add_handle_value(Response_Type &response);
int server_action_delete_handle(Response_Type &response);
int server_action_undelete_handle(Response_Type &response);
int server_action_delete_handle_value(Response_Type &response);
int server_action_undelete_handle_value(Response_Type &response);
int server_action_undelete_file(Response_Type &response);
int server_action_unknown(Response_Type &response);
```

302. Client action functions. [LDF 2013.05.30.]**Log**

[LDF 2013.05.30.] Added this section. These functions are defined in `clntactn.web`.

`<Scan_Parse_Parameter_Type function declarations 38> +≡`

```
int client_action_command_only(Response_Type &response);
int client_action_send_file(Response_Type &response);
int client_action_unknown(Response_Type &response);
```

303.

`<Garbage 303> ≡`

See also sections 542, 558, 747, 1407, 1474, 1492, 1510, 1542, and 1574.

This code is used in sections 305, 544, 560, 748, 1409, 1476, 1494, 1512, 1544, and 1576.

304. Putting `scprpmtp.web` together.**305.** This is what's compiled.

```
<Include files 3>
using namespace std;
using namespace gwrdifpk;
class User_Info_Type;

<External function declarations 32>
<class Scan_Parse_Parameter_Type declaration 33>
<Initialize Scan_Parse_Parameter_Type static data members 36>
<Scan_Parse_Parameter_Type constructor definitions 39>
<Scan_Parse_Parameter_Type destructor definition 51>
<Initialize Scan_Parse_Parameter_Type maps 57>
<Scan_Parse_Parameter_Type::get_database_username definition 62>
<Scan_Parse_Parameter_Type::get_user definition 70>
<Scan_Parse_Parameter_Type::show definition 106>
<Scan_Parse_Parameter_Type::submit_mysql_query definition 108>
<Scan_Parse_Parameter_Type::send_tan_list definition 115>
<Scan_Parse_Parameter_Type::set_expires definition 131>
<Scan_Parse_Parameter_Type::get_expires definition 133>
<Scan_Parse_Parameter_Type::put definition 151>
<Scan_Parse_Parameter_Type::get definition 201>
<Scan_Parse_Parameter_Type::send_to_peer definitions 218>
<Scan_Parse_Parameter_Type::receive_file definition 235>
<Scan_Parse_Parameter_Type::cd definition 256>
<Scan_Parse_Parameter_Type::ls definition 136>
<Scan_Parse_Parameter_Type::pwd definition 145>
<Scan_Parse_Parameter_Type::mkdir definition 274>
#ifndef 0
<Garbage 303>
#endif
```

306. This is what's written to the header file `scprpmtp.h`. [LDF 2012.06.27.]

```
<scprpmtp.h 306>≡  
#ifndef SCPRPMTP_H  
#define SCPRPMTP_H 1  
using namespace std;  
using namespace gwrdifpk;  
class User_Info_Type;  
class X509_Cert_Type;  
< class Scan_Parse_Parameter_Type declaration 33 >  
#endif
```

307. class User_Info_Type (usrinftp.web).

308. Include files.

```
<Include files 3> +≡
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <math.h>
#include <time.h>
#include <gcrypt.h> /* for gcry_control */
#include <gnutls/gnutls.h>
#include <iomanip>
#include <iostream>
#include <map>
#include <fstream>
#include <sstream>
#include <string>
#include <vector>
#include <deque>
#include <stack>
#include <set>
#include <pthread.h>
#include <expat.h>
#if HAVE_CONFIG_H
#include <config.h>
#endif
#include <mysql.h>
#ifndef _XOPEN_SOURCE
#define _XOPEN_SOURCE
#endif
#include "glblcnst.h++"
#include "glblvrbl.h++"
#include "excptntp.h++"
#include "utilfncs.h++"
#include "grouptp.h++"
#include "hdlvltp.h++"
#include "irdsavtp.h++"
#include "helper.h++"
#include "tanfncs.h++"
#include "pidfncs.h++"
#include "parser.h++"
#include "scanner.h++"
#include "rspnstp.h++"
#include "irdsobtp.h++"
#include "hndltype.h++"
#include "dcmtddtp.h++"
#include "x509cert.h++"
#include "dstngnmt.h++"
#include "scprpmtp.h++"
```

309. class User_Info_Type declaration. [LDF 2013.05.14.]

Log

[LDF 2013.05.14.] Added this **class** declaration.
 [LDF 2013.05.16.] Added the data member **unsigned int** *privileges*.
 [LDF 2013.05.17.] Added the data members **X509_Cert_Type** *certificate*.
 [LDF 2013.05.22.] Added **friend** declarations for *distinguished_name_rule_func* and *get_user_info_func*.
 [LDF 2013.07.18.] Added **string irods_env_filename** and **string irods_auth_filename**.
 [LDF 2013.09.20.] Added **string public_key_id**.

```
< class User_Info_Type declaration 309 > ≡
class User_Info_Type {
    friend class Scan_Parse_Parameter_Type;
    friend int yyparse(yyscan_t parameter);
    friend int distinguished_name_rule_func(Scan_Parse_Parameter_Type *, const char *);
    friend int get_user_info_func(Scan_Parse_Parameter_Type *, const char *);
    int user_id;
    string username;
    Distinguished_Name_Type distinguished_name;
    unsigned int privileges;
    string irods_password_encrypted;
    string irods_password_encrypted_timestamp;
    string irods_current_dir;
    string irods_homedir;
    string irods_zone;
    string irods_default_resource;
    string irods_env_filename;
    string irods_auth_filename;
    unsigned int default_handle_prefix_id;
    string default_handle_prefix;
    unsigned int default_institute_id;
    string default_institute_name;
    X509_Cert_Type certificate;

public: < User_Info_Type constructor declarations 314 >
    < User_Info_Type function declarations 316 >
    int get_user_id(void)
    {
        return user_id;
    }
    string get_irods_env_filename(void)
    {
        return irods_env_filename;
    }
    string get_irods_auth_filename(void)
    {
        return irods_auth_filename;
    }
    string public_key_id;
};
```

This code is used in sections 320 and 321.

310. Global variables. [LDF 2013.07.18.]

Log

[LDF 2013.07.18.] Added this section.

```
< Global variables 310 > ≡
  pthread_mutex_t global_user_info_map_mutex;
  map<int, User_Info_Type> global_user_info_map;
```

See also sections 1372 and 1396.

This code is used in sections 320 and 1409.

311.

```
< extern declarations for global variables 311 > ≡
  extern map<int, User_Info_Type> global_user_info_map;
  extern pthread_mutex_t global_user_info_map_mutex;
```

This code is used in section 321.

312. User_Info_Type functions. [LDF 2013.05.14.]

Log

[LDF 2013.05.14.] Added this section.

313. Constructors and Setting Functions. [LDF 2013.05.14.]

Log

[LDF 2013.05.14.] Added this section.

314. Default constructor. [LDF 2013.05.14.]

Log

[LDF 2013.05.14.] Added this function.

```
< User_Info_Type constructor declarations 314 > ≡
  User_Info_Type(void);
```

This code is used in section 309.

315.

```
< User_Info_Type constructor definitions 315 > ≡
  User_Info_Type::User_Info_Type(void)
  {
    user_id = -1;
    username = "";
    privileges = 0U;
    default_handle_prefix_id = 0;
    default_institute_id = 0;
    return;
  } /* End of User_Info_Type default constructor definition */
```

This code is used in section 320.

316. Clear. [LDF 2013.05.16.]

Log

[LDF 2013.05.16.] Added this function.

{ User_Info_Type function declarations 316 } ≡**void clear(void);**

See also section 318.

This code is used in section 309.

317.**{ User_Info_Type** :: *clear* definition 317 } ≡

```
void User_Info_Type :: clear(void)
{
    user_id = 0;
    username = irods_password_encrypted = irods_password_encrypted_timestamp = irods_current_dir =
        irods_homedir = irods_zone = irods_default_resource = default_handle_prefix =
        default_institute_name = irods_env_filename = irods_auth_filename = public_key_id = "";
    privileges = default_handle_prefix_id = default_institute_id = 0U;
    certificate.clear();
    distinguished_name.clear();
    return;
}
```

/* End of **User_Info_Type** :: *clear* definition */

This code is used in section 320.

318. Show. [LDF 2013.05.14.]

Log

[LDF 2013.05.14.] Added this function.

[LDF 2013.05.17.] Added optional argument **stringstream *strm = 0**.**{ User_Info_Type** function declarations 316 } +≡**void show(string s = "", stringstream *strm = 0, bool brief = false) const;**

319.

```

⟨ User_Info_Type :: show definition 319 ⟩ ≡
void User_Info_Type :: show(string s, stringstream *strm, bool brief) const
{
    if (s == "") s = "User_Info_Type:";

    stringstream temp_strm;
    temp_strm << s << endl << "user_id:_____" << user_id << endl << "username:_____"
        << username << endl;
    if (brief) {
        temp_strm << "Common_Name:__" << certificate.commonName << endl;
    }
    else {
        distinguished_name.show("distinguished_name:", &temp_strm);
        temp_strm << "privileges:" << endl << "___superuser:_____"
            << ((privileges & Scan_Parse_Parameter_Type :: SUPERUSER_PRIVILEGE) ?
                1 : 0) << endl << "___delegate:_____"
            << ((privileges & Scan_Parse_Parameter_Type :: DELEGATE_PRIVILEGE) ?
                1 : 0) << endl << "___delete_handles:_____"
            << ((privileges & Scan_Parse_Parameter_Type :: DELETE_HANDLES_PRIVILEGE) ?
                1 : 0) << endl << "___show_user_info:_____"
            << ((privileges & Scan_Parse_Parameter_Type :: SHOW_USER_INFO_PRIVILEGE) ?
                1 : 0) << endl << "___show_certificates:_____"
            << ((privileges & Scan_Parse_Parameter_Type :: SHOW_CERTIFICATES_PRIVILEGE) ?
                1 : 0) << endl << "___show_distinguished_names:_"
            << ((privileges & Scan_Parse_Parameter_Type :: SHOW_DISTINGUISHED_NAMES_PRIVILEGE) ?
                1 : 0) << endl << "___show_privileges:_____"
            << ((privileges & Scan_Parse_Parameter_Type :: SHOW_PRIVILEGES_PRIVILEGE) ?
                1 : 0) << endl << "irods_password_encrypted:_____"
            << irods_password_encrypted << endl << "irods_password_encrypted_timestamp:__"
            << irods_password_encrypted_timestamp << endl << "irods_homedir:_____"
            << irods_homedir << endl << "irods_current_dir:_____"
            << irods_current_dir << endl << "irods_zone:_____"
            << irods_zone << endl << "irods_default_resource:_____"
            << irods_default_resource << endl << "irods_env_filename:_____"
            << irods_env_filename << endl << "irods_auth_filename:_____"
            << irods_auth_filename << endl << "default_handle_prefix_id:_____"
            << default_handle_prefix_id << endl << "default_handle_prefix:_____"
            << default_handle_prefix << endl << "default_institute_id:_____"
            << default_institute_id << endl << "default_institute_name:_____"
            << default_institute_name << endl << "public_key_id:_____"
            << public_key_id << endl;
        certificate.show("certificate:", &temp_strm, false);
    } /* else (~brief) */
    if (*strm) *strm << temp_strm.str();
    else cerr << temp_strm.str();
    return;
} /* End of User_Info_Type :: show definition */

```

This code is used in section 320.

320. Putting **class User_Info_Type** together. [LDF 2013.05.14.]

```
using namespace std;
⟨ Include files 3 ⟩
using namespace gwrdifpk;
typedef void *yyScan_t;
⟨ class User_Info_Type declaration 309 ⟩
⟨ Global variables 310 ⟩
⟨ User_Info_Type constructor definitions 315 ⟩
⟨ User_Info_Type::clear definition 317 ⟩
⟨ User_Info_Type::show definition 319 ⟩
```

321.

```
⟨ usrinftp.h 321 ⟩ ≡
#ifndef USRINFTP_H
#define USRINFTP_H 1
typedef void *yyScan_t;
using namespace std;
⟨ class User_Info_Type declaration 309 ⟩
⟨ extern declarations for global variables 311 ⟩
#endif
```

322. Scan_Parse_Parameter_Type function definitions 1.

323. Include files.

```
<Include files 3> +≡
#include <stdlib.h>      /* Standard Library for C */
#include <stdio.h>
#include <sys/mman.h>
#include <errno.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <limits.h>
#include <string.h>
#include <cctype.h>
#include <algorithm>      /* Standard Template Library (STL) for C++ */
#include <bitset>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <map>
#include <string>
#include <time.h>
#include <math.h>
#include <sstream>
#include <vector>
#include <deque>
#include <stack>
#include <set>
#include <pthread.h>      /* POSIX threads */
#include <gcrypt.h>        /* for gcry_control */
#include <gnutls/gnutls.h>
#include <gnutls/x509.h>
#include <mysql.h>
#include <expat.h>
#if HAVE_CONFIG_H
#include <config.h>
#endif
#undef NAME_LEN
#undef LOCAL_HOST
#include "rspercds.h++"
#include "glblcnst.h++"
#include "glblvrbl.h++"
#include "excptntp.h++"
#include "utilfncts.h++"
#include "grouptp.h++"
#include "hdlvltp.h++"
#include "irdsavtp.h++"
#include "helper.h++"
#include "tanfncts.h++"
#include "pidfncts.h++"
#include "parser.h++"
#include "scanner.h++"
#include "rspnstp.h++"
```

```
#include "irdsobtp.h++"
#include "hdltype.h++"
#include "dcmtdttp.h++"
#include "x509cert.h++"
#include "dstngnmt.h++"
#include "scprpmtp.h++"
#include "usrinftp.h++"
```

324.

⟨ External function declarations 32 ⟩ +≡
 int yyparse(yyvsp<parameter>);

325. Scanner_Type functions 1. [LDF 2013.02.13.]**326. Get handle. (*get_handle*).** [LDF 2012.10.15.]

Log

[LDF 2012.10.15.] Added this function.

[LDF 2013.02.13.] Moved the definition of this function from `scprpmtp.web` to this file (`spptfnc1.web`).

327.

⟨ `Scan_Parse_Parameter_Type::get_handle` definition 327 ⟩ ≡

```
int Scan_Parse_Parameter_Type::get_handle(string s, unsigned int flags, string filename_1){
    bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread_" << thread_ctr << "]_Entering_`Scan_Parse_Parameter_Type::`"
            get_handle'." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
    int status;
    string filename;
    string handle_str;
    stringstream sql_strm;
    stringstream temp_strm;
    MYSQL_RES * result = 0;
    unsigned int row_ctr;
    unsigned int field_ctr;
    Response_Type response;
    response.type = Response_Type::COMMAND_ONLY_TYPE;
```

See also sections 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, and 339.

This code is used in section 544.

328. Select “handlesystem” or “handlesystem_standalone” database. [LDF 2012.10.15.]

```
<Scan_Parse_Parameter_Type::get_handle definition 327> +≡
  string database_name = (standalone_handle) ? "handlesystem_standalone" : "handlesystem";
  status = mysql_select_db(mysql_ptr, database_name.c_str());
  if (status ≡ 0) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "In 'Scan_Parse_Parameter_Type::get_handle': " ≪
      "'mysql_select_db' succeeded'." ≪ endl ≪ "Selected '" ≪ database_name ≪
      "' database successfully." ≪ endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (status ≡ 0) */
else /* status ≠ 0 */
{
  lock_cerr_mutex();
  cerr ≪ "In 'Scan_Parse_Parameter_Type::get_handle': " ≪
    "'mysql_select_db' failed, returning" ≪ status ≪ endl ≪ "Failed to select '" ≪
    database_name ≪ "' database:" ≪ endl ≪ "Error: " ≪ mysql_error(mysql_ptr) ≪ endl ≪
    "Exiting function unsuccessfully with return value 1." ≪ endl;
  unlock_cerr_mutex();
  temp_strm.str("");
  temp_strm ≪ "GET_HANDLE_RESPONSE_1_HANDLE\" " ≪ s ≪ "\n";
  response.command = temp_strm.str();
  response_deque.push_back(response);
  ++errors_occurred;
  return 1;
} /* else (status ≠ 0) */
```

329.

```

⟨ Scan_Parse_Parameter_Type ::get_handle definition 327 ⟩ +≡
  if (flags & 1U) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread]" << thread_ctr << "] In 'Scan_Parse_Parameter_Type::get_handle':" <<
      endl << "flags&1U" << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  filename = s;
  Irods_Object_Type irods_object;
  irods_object.path = filename;
  temp_strm.str("");
  temp_strm << "env_irodsEnvFile=" << irods_env_filename << "imeta_ls-ld" << filename << "PID";
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "temp_strm.str() == " << temp_strm.str() << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  int ctr;
  status = irods_object.get_avus_from_irods_system(temp_strm.str(),filename,*this,&ctr);
  if (status != 0) {
    cerr << "[Thread" << thread_ctr << "] "
      << "ERROR! In 'Scan_Parse_Parameter_Type::get_handle':" << endl <<
      "'Irods_Object_Type::get_avus_from_irods_system' failed, returning" << status <<
      "." << endl << "Failed to read iRODs user-defined metadata for iRODS object" << " "
      << filename << "." << endl << "Exiting function unsuccessfully with return value" <<
      endl;
    temp_strm.str("");
    temp_strm << "GET_HANDLE_RESPONSE_1";
    response.command = temp_strm.str();
    response_deque.push_back(response);
    ++errors_occurred;
  }
  return 1;
} /* if (status != 0) */

```

330.

```
<Scan_Parse_Parameter_Type::get_handle definition 327> +≡
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "[Thread_"
            << thread_ctr << "]_"
            << "In_`Scan_Parse_Parameter_Type::get_handle':"
            << endl << "``Irods_Object_Type::get_avus_from_irods_system'_succeeded,\n"
            << "returning_0." << endl;
            irods_object.show("irods_object:");
            cerr << "ctr_=="
            << ctr << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

331. Call this function recursively in loop. [LDF 2012.11.22.]

```
<Scan_Parse_Parameter_Type::get_handle definition 327> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Before_loop."
        << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    for (vector<Irods_AVU_Type>::iterator iter = irods_object.avu_vector.begin();
        iter != irods_object.avu_vector.end(); ++iter) {
        status = get_handle(iter->value, 0, filename);
        if (status != 0) {
            lock_cerr_mutex();
            cerr << "[Thread_"
            << thread_ctr << "]_WARNING!"
            << endl <<
            "In_`Scan_Parse_Parameter_Type::get_handle':"
            << endl <<
            "Recursive_call_to_`Scan_Parse_Parameter_Type::get_handle'_failed,\n"
            << "returning_"
            << status << "."
            << endl << "Will_try_to_continue."
            << endl;
            unlock_cerr_mutex();
            ++warnings_occurred;
            continue;
        } /* if (status != 0) */
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread_"
        << thread_ctr << "]_"
        << "In_`Scan_Parse_Parameter_Type::get_handle':"
        << endl <<
        "Recursive_call_to_`Scan_Parse_Parameter_Type::get_handle'_succeeded,\n"
        << "returning_0."
        << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* for */
```

332.

```

⟨ Scan_Parse_Parameter_Type::get_handle definition 327 ⟩ +≡
    return 0; }      /* if (flags & 1U) */
    else      /* (flags & 1U == 0) */
    {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] In 'Scan_Parse_Parameter_Type::get_handle':"
            endl << " flags&1U'" << 0 . Getting handle for PID " << s << "."
            endl;
        unlock_cerr_mutex();
    }      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */
    handle_str = s;
    sql_strm << "select handle_id, handle_value_id, idx, type, "
        << "length(data), data, "
        << "ttl_type, ttl, timestamp, length(refs), refs, admin_read, "
        << "admin_write, pub_read, pub_write, marked_for_deletion, created, "
        << "last_modified, "
        << "handle, created_by_user_id, irods_object_id, delete_from_da\
tabase_timestamp"
        << "from handles where handle='"
        << handle_str << "'"
        << "order by idx";
}      /* else (flags & 1U == 0) */

```

333.

```

⟨ Scan_Parse_Parameter_Type::get_handle definition 327 ⟩ +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "sql_strm.str() == " << sql_strm.str() << endl;
        unlock_cerr_mutex();
    }      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */

```

334.

```
<Scan_Parse_Parameter_Type::get_handle definition 327> +≡
status = submit_mysql_query(sql_strm.str(), result, &row_ctr, &field_ctr);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ "[Thread" ≪ thread_ctr ≪ "] ERROR! In 'Scan_\
Parse_Parameter_Type::get_handle':"
    ≪ endl ≪
    "Scan_Parse_Parameter_Type::submit_mysql_query' failed, returning" ≪ status ≪
    ". " ≪ endl ≪ mysql_error(mysql_ptr) ≪ endl ≪ "Failed to retrieve handle values."
    ≪ endl ≪ "Exiting function unsuccessfully with return value 1." ≪ endl;
unlock_cerr_mutex();
if (result) {
    mysql_free_result(result);
}
temp_strm.str("");
temp_strm ≪ "GET_HANDLE_RESPONSE" ≪ endl ≪ s ≪ "\n";
response.command = temp_strm.str();
response_deque.push_back(response);
++errors_occurred;
return 1;
} /* if (status ≠ 0) */
#endif /* DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "[Thread" ≪ thread_ctr ≪ "] In 'Scan_Parse_Parameter_Type::get_handle':"
    ≪ endl ≪
    "'Scan_Parse_Parameter_Type::submit_mysql_query' succeeded." ≪ endl ≪
    "'row_ctr' == " ≪ row_ctr ≪ endl ≪ "'field_ctr' == " ≪ field_ctr ≪ endl;
unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

335.

```

⟨ Scan_Parse_Parameter_Type ::get_handle definition 327 ⟩ +≡
  if (row_ctr ≡ 0) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "]WARNING!" <<
      "In‘Scan_Parse_Parameter_Type::get_handle’:" << endl <<
      “Scan_Parse_Parameter_Type::submit_mysql_query” returned 0 rows." <<
      endl << "Failed to retrieve handle data." << endl <<
      "Exiting function unsuccessfully with return value 2." << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    temp_strm.str("");
    temp_strm << "GET_HANDLE_RESPONSE_3_HANDLE\"" << s << "\"";
    response.command = temp_strm.str();
    response_deque.push_back(response);
    ++warnings_occurred;
    return 2;
  } /* if (row_ctr ≡ 0) */
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "[Thread" << thread_ctr << "]In‘Scan_Parse_Parameter_Type::get_handle’:" <<
        endl << “Scan_Parse_Parameter_Type::submit_mysql_query” succeeded, returning "
        row_ctr << " rows." << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

336.

Log

[LDF 2013.03.01.] BUG FIX: Changed the type of `temp_handle_value_vector` from `vector<Handle_Value_Type*>` to `vector<Handle_Value_Type>` and that of `handle_obj` from `Handle_Value_Type *` to `Handle_Value_Type`. Previously, I was using pointers and allocating memory on the heap because of errors when using `Handle_Value_Type`. The reason for the errors was bugs in the `Handle_Value_Type` default constructor and destructor with respect to the way the data members `char *data` and `char *refs` were handled. I have since corrected the errors, so there's no need to use pointers now.

```

⟨ Scan_Parse_Parameter_Type::get_handle definition 327 ⟩ +≡
MySQL_ROW curr_row;
vector<Handle_Value_Type> temp_handle_value_vector;
Handle_Value_Type handle_obj;
Handle_Type *curr_handle = new Handle_Type; for (int i = 0; i < row_ctr; ++i) {
if ((curr_row = mysql_fetch_row(result)) ≡ 0) {
    lock_cerr_mutex();
    cerr ≪ "[Thread" ≪ thread_ctr ≪ "] ERROR!" ≪
        "In 'Scan_Parse_Parameter_Type::get_handle':"
        ≪ endl ≪ "'mysql_fetch_row' failed:" ≪ endl ≪ mysql_error(mysql_ptr) ≪ endl ≪
        "Exiting function unsuccessfully with return value 1." ≪ endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    temp_strm.str("");
    temp_strm ≪ "GET_HANDLE_RESPONSE\4_HANDLE\"" ≪ s ≪ "\n";
    response.command = temp_strm.str();
    response_deque.push_back(response);
    ++errors_occurred;
    return 1;
} /* if (curr_row = mysql_fetch_row(result) ≡ 0) */
else if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "[Thread" ≪ thread_ctr ≪ "] In 'Scan_Parse_Parameter_Type::get_handle':"
        ≪ endl ≪ "'mysql_fetch_row' succeeded."
        ≪ endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */

```

337.

```

< Scan_Parse_Parameter_Type ::get_handle definition 327 > +≡
status = handle_obj.set(curr_row, field_ctr, handle_str, thread_ctr);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ "[Thread" ≪ thread_ctr ≪ "] ERROR!" ≪
        "In 'Scan_Parse_Parameter_Type::get_handle':"
        ≪ endl ≪
        "'Handle_Value_Type::set' failed, returning"
        ≪ status ≪ "."
        ≪ endl ≪
        "Exiting function unsuccessfully with return value 1."
        ≪ endl;
    unlock_cerr_mutex();
}
if (result) mysql_free_result(result);
temp_strm.str("");
temp_strm ≪ "GET_HANDLE_RESPONSE_5_HANDLE\""
            ≪ s ≪ "\n";
response.command = temp_strm.str();
response_deque.push_back(response);
++errors_occurred;
return 1;
} /* if (status ≠ 0) */
#endif /* DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "'Handle_Value_Type::set' succeeded, returning 0."
    ≪ endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
if (!filename_1.empty()) {
    handle_obj.filename = filename_1;
}
curr_handle->handle_value_map[handle_obj.idx] = handle_obj;
temp_handle_value_vector.push_back(handle_obj);
handle_obj.clear(); /* for */
mysql_free_result(result);
result = 0;
curr_handle->handle = curr_handle->handle_value_map.begin()→second.handle;
curr_handle->handle_id = curr_handle->handle_value_map.begin()→second.handle_id;
#endif /* DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "temp_handle_value_vector.size() == "
        ≪ temp_handle_value_vector.size()
        ≪ endl;
    for (vector<Handle_Value_Type>::const_iterator iter =
        temp_handle_value_vector.begin();
        iter ≠ temp_handle_value_vector.end(); ++iter) {
        iter->show();
    } /* for */
    curr_handle->show("*curr_handle:");
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

338.

```
{ Scan_Parse_Parameter_Type::get_handle definition 327 } +≡
    response.type = Response_Type::SEND_HANDLE_TYPE;
    response.handle = curr_handle;
    response_deque.push_back(response);
```

339.

```
{ Scan_Parse_Parameter_Type::get_handle definition 327 } +≡
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread_" << thread_ctr << "]_Exiting_`Scan_Parse_Parameter_Type::get_handle'" <<
            "successfully_with_return_value_0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
    return 0; } /* End of Scan_Parse_Parameter_Type::get_handle definition */
```

340. Get metadata. [LDF 2012.10.09.]

Log

[LDF 2012.10.09.] Added this function.

[LDF 2012.12.21.] Added the optional arguments **bool send_response**, **bool do_output**, **bool do_irods_user_metadata** and **vector<Dublin_Core_Metadata_Type> *dc_metadata_type_vector_ptr**. Adapting this function for use with **Scan_Parse_Parameter_Type::add_metadata**. For this use, **get_metadata** should store the data from the database in **Dublin_Core_Metadata_Type** objects in ***dc_metadata_type_vector_ptr** but neither output XML code to a temporary file nor send responses to the peer.

[LDF 2012.12.31.] Added optional argument **unsigned int options** with the default 0.

[LDF 2013.01.04.] Added code for sending responses in the error cases.

[LDF 2013.02.13.] Moved the definition of this function from **scprpmtp.web** to this file (**spptfnc1.web**).

[LDF 2013.05.23.] Replaced the optional argument **bool send_response = true** with the optional arguments **char *buffer_ptr = 0** and **size_t buffer_size = 0**.

341.

```

⟨ Scan_Parse_Parameter_Type ::get_metadata definition 341 ⟩ ≡
int Scan_Parse_Parameter_Type ::get_metadata(string filename, unsigned int
    flags, int *ctr, unsigned int options, char *buffer_ptr, size_t buffer_size, bool
    do_output, bool do_irods_user_metadata, vector<Dublin_Core_Metadata_Type>
    *dc_metadata_type_vector_ptr){ bool DEBUG = false; /* true */
set_debug_level(DEBUG, 0, 0);
/* !! TODO: LDF 2013.03.21. Work on parser rule "get_metadata pid string...". */
stringstream temp_strm;
int status = 0;
Response_Type response;
response.type = Response_Type ::COMMAND_ONLY_TYPE;
unsigned long temp_val;
Dublin_Core_Metadata_Type dcm;
vector<Dublin_Core_Metadata_Type> dc_metadata_type_vector;
Dublin_Core_Metadata_Sub_Type dcm_sub;
multimap<unsigned int, Dublin_Core_Metadata_Sub_Type>::iterator dc_metadata_sub_iter;
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] " <<
        "Entering 'Scan_Parse_Parameter_Type::get_metadata'." << endl <<
        "'filename'" <= filename << endl << "'flags'" <= hex << flags << dec << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
string cmd_str;

```

See also sections 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, and 382.

This code is used in section 544.

342.**Log**

[LDF 2013.01.09.] BUG FIX: Added code for setting *cmd_str*. Similar code must have existed before; I must have removed it by mistake sometime.

```

⟨ Scan_Parse_Parameter_Type ::get_metadata definition 341 ⟩ +≡
if (flags & 1U) cmd_str = "PIDS";
else cmd_str = "METADATA";

```

343. Check whether *filename* includes a slash. If it doesn't, insert *irods_current_dir* at the front of it.
[LDF 2012.12.12.]

Log

[LDF 2012.12.12.] Added this section.

```
⟨Scan_Parse_Parameter_Type::get_metadata definition 341⟩ +≡
    size_t pos = filename.find('/');
    if (pos ≡ string::npos) {
        filename.insert(0, "/");
        filename.insert(0, irods_current_dir);
    }
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ "filename" ≪ filename ≪ endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

344.

Log

[LDF 2012.12.07.] Added this section.

```

⟨ Scan_Parse_Parameter_Type::get_metadata definition 341 ⟩ +≡
    stringstream sql_strm;
    MYSQL_RES * result = 0;
    unsigned int row_ctr;
    unsigned int field_ctr;
    MYSQL_ROW curr_row;
    stringstream sql_strm_1;
    MYSQL_RES * result_1 = 0;
    unsigned int row_ctr_1;
    unsigned int field_ctr_1;
    MYSQL_ROW curr_row_1;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "user_id=" << user_id << endl;
        cerr << "irods_current_dir=" << irods_current_dir << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */ /* !! TODO: LDF 2012.12.12. Adjust to account for different
    "where" clauses. A query may return data for more than one user, irods_object_path value, etc. */
    sql_strm << "select_dublin_core_metadata_id, user_id, handle_id, irods_object_path"
    "from_Dublin_Core_Metadata where"
    "user_id=" << user_id << endl;
    "and irods_object_path=" << filename << ',' << "order_by_dublin_core_metadata_id";
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "sql_strm.str()=" << sql_strm.str() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    status = submit_mysql_query(sql_strm.str(), result, &row_ctr, &field_ctr);
    if (status != 0) {
        lock_cerr_mutex();
        cerr << "[Thread]" << thread_ctr << "] "
        "ERROR! In 'Scan_Parse_Parameter_Type::get_metadata':"
        "Scan_Parse_Parameter_Type::submit_mysql_query' failed, returning"
        "status << status << endl << mysql_error(mysql_ptr) << endl << "Failed to retrieve metadata."
        "Exiting function unsuccessfully with return value 1."
        "with return value 1." << endl;
        unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    ++errors_occurred;
    if (buffer_ptr != 0 & buffer_size > 0) {
        temp_strm.str("");
        temp_strm << "GET "
        cmd_str << "RESPONSE\""
        << filename << "\" "
        << "10" << options <<
        "U" << "\Error: Failed to retrieve XML metadata"
        << "for_iRODSobject"
        << filename << '"';

```

```
    strcpy(buffer_ptr,temp_strm.str().c_str());
} /* if (buffer_ptr != 0 & buffer_size > 0) */
++errors_occurred;
return 1;
} /* if (status != 0) */
#endif DEBUG_COMPILE
else
if (DEBUG) {
lock_cerr_mutex();
cerr << "[Thread" << thread_ctr << "]"
<< "In 'Scan_Parse_Parameter_Type::get_metadata':"
<< endl <<
"'Scan_Parse_Parameter_Type::submit_mysql_query' succeeded, returning 0."
<< endl;
unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

345.

```

⟨ Scan_Parse_Parameter_Type ::get_metadata definition 341 ⟩ +≡
  if (row_ctr == 0) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread]" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::get_meta"
      data':" << endl << "'row_ctr'==0, MySQL query returned no rows." << endl <<
      "No XML metadata present in database." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  mysql_free_result(result);
  result = 0;
  if (buffer_ptr != 0 & buffer_size > 0) {
    temp_strm.str("");
    temp_strm << "GET " << cmd_str << "_RESPONSE\" " << filename << "\\" << "0_0" << options <<
      "\\" << "\No XML metadata for iRODS object" << filename << "\\";
    strcpy(buffer_ptr, temp_strm.str().c_str());
    temp_strm.str("");
  }
#endif DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread]" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::get_meta"
      data':" << endl << "Skipping to 'GET_METADATA_IRODS_USER_METADATA'." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  goto GET_METADATA_IRODS_USER_METADATA;
} /* if (row_ctr == 0) */
#ifndef DEBUG_COMPILE
else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread]" << thread_ctr << "] " <<
      "In 'Scan_Parse_Parameter_Type::get_metadata':" << endl << "'row_ctr'=="
      row_ctr << ", " << endl << "'field_ctr'=="
      field_ctr << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */ /* ***** (7) */
for (int i = 0; i < row_ctr; ++i) {
  if ((curr_row = mysql_fetch_row(result)) == 0) {
    lock_cerr_mutex();
    cerr << "[Thread]" << thread_ctr << "] ERROR! " <<
      "In 'Scan_Parse_Parameter_Type::get_metadata':" << endl <<
      "'mysql_fetch_row' failed: " << endl << mysql_error(mysql_ptr) << endl <<
      "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
  mysql_free_result(result);
  if (buffer_ptr != 0 & buffer_size > 0) {

```

```
temp_strm.str("");
temp_strm << "GET" << cmd_str << "RESPONSE\" " << filename << "\" << "10" << options <<
    "U" << "\\" Error:\\Failed\\to\\retrieve\\XML\\metadata" << "for\\ir0DS\\object" " <<
    filename << ',' \\from\\database\" ";
strcpy(buffer_ptr, temp_strm.str().c_str());
} /* if (buffer_ptr != 0 & buffer_size > 0) */
++errors_occurred;
return 1;
} /* if (curr_row = mysql_fetch_row(result) == 0) */
else if (DEBUG) {
lock_cerr_mutex();
cerr << "[Thread" << thread_ctr << "] In 'Scan_Parse_Parameter_Type::get_metadata':"
    endl << 'mysql_fetch_row'\\succeeded." << endl;
unlock_cerr_mutex();
} /* else if (DEBUG) */
```

346. dublin_core_metadata_id.

```

⟨ Scan_Parse_Parameter_Type ::get_metadata definition 341 ⟩ +≡
  if (curr_row[0] ∧ strlen(curr_row[0]) > 0) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "curr_row[0]" << setw(24) << std::left << "dublin_core_metadata_id" <<
      curr_row[0] << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  errno = 0;
  temp_val = strtoul(curr_row[0], 0, 10);
  if (temp_val ≡ ULONG_MAX) {
    lock_cerr_mutex();
    cerr << "[Thread]" << thread_ctr << "]ERROR!" <<
      "In 'Scan_Parse_Parameter_Type::get_metadata':" << endl <<
      "'strtoul' failed, returning 'ULONG_MAX':" << endl << strerror(errno) <<
      endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    ++errors_occurred;
    if (buffer_ptr ≠ 0 ∧ buffer_size > 0) {
      temp_strm.str("");
      temp_strm << "GET" << cmd_str << "RESPONSE\" " << filename << "\" " <<
        "10" << options << "U" << "\Error: Failed to retrieve XML metadata" <<
        "for iRODS object" << filename << "from database\"";
      strcpy(buffer_ptr, temp_strm.str().c_str());
    } /* if (buffer_ptr ≠ 0 ∧ buffer_size > 0) */
    ++errors_occurred;
    return 1;
  } /* if (temp_val ≡ ULONG_MAX) */
#endif DEBUG_COMPILE
  else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread]" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::get_meta" <<
      "data':" << endl << "'strtoul' succeeded. " << "temp_val'" << temp_val << "." << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  dcm.id = temp_val;
} /* if (curr_row[0] ∧ strlen(curr_row[0]) > 0) */

```

347. user_id.

Log

[LDF 2012.12.12.] Added this section.

```

⟨ Scan_Parse_Parameter_Type ::get_metadata definition 341 ⟩ +≡
  if (curr_row[1] ∧ strlen(curr_row[1]) > 0) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "curr_row[1]" << setw(24) << std::left << "user_id" << curr_row[1] << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  errno = 0;
  temp_val = strtoul(curr_row[1], 0, 10);
  if (temp_val == ULONG_MAX) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] ERROR!" <<
      "In 'Scan_Parse_Parameter_Type::get_metadata':" << endl <<
      "'strtoul' failed, returning 'ULONG_MAX':" << endl << strerror(errno) <<
      endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    ++errors_occurred;
  if (buffer_ptr ≠ 0 ∧ buffer_size > 0) {
    temp_strm.str("");
    temp_strm << "GET" << cmd_str << "RESPONSE\\"" << filename << "\" " <<
      "10" << options << "U" << "\"Error: Failed to read XML metadata\" " <<
      "for iRODS object'" << filename << "'\"";
    strcpy(buffer_ptr, temp_strm.str().c_str());
  } /* if (buffer_ptr ≠ 0 ∧ buffer_size > 0) */
    ++errors_occurred;
  return 1;
} /* if (temp_val == ULONG_MAX) */
#endif DEBUG_COMPILE
else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::get_meta\
      data':" << endl << "'strtoul'succeeded.'temp_val'" << temp_val << "." << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  dcm.user_id = temp_val;
} /* if (curr_row[1] ∧ strlen(curr_row[1]) > 0) */

```

348. handle_id.

Log

[LDF 2012.12.12.] Added this section.

```

⟨ Scan_Parse_Parameter_Type::get_metadata definition 341 ⟩ +≡
    if (curr_row[2] ∧ strlen(curr_row[2]) > 0) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "curr_row[2]" << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    errno = 0;
    temp_val = strtoul(curr_row[2], 0, 10);
    if (temp_val ≡ ULONG_MAX) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "]ERROR!" <<
            "In 'Scan_Parse_Parameter_Type::get_metadata':" << endl <<
            "'strtoul' failed, returning ULONG_MAX'" << endl << strerror(errno) <<
            endl << "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        mysql_free_result(result);
        ++errors_occurred;
        if (buffer_ptr ≠ 0 ∧ buffer_size > 0) {
            temp_strm.str("");
            temp_strm << "GET" << cmd_str << "RESPONSE\\"" << filename << "\""
                << "10" << options << "U" << "\"Error: Failed to read XML metadata\""
                << "for iRODS_object\"" << filename << "\"";
            strcpy(buffer_ptr, temp_strm.str().c_str());
        } /* if (buffer_ptr ≠ 0 ∧ buffer_size > 0) */
        ++errors_occurred;
        return 1;
    } /* if (temp_val ≡ ULONG_MAX) */
#endif DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "[Thread" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::get_meta"
                "data':" << endl << "'strtoul' succeeded. " << "'temp_val'" << temp_val << "."
                << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
        dcm.handle_id = temp_val;
    } /* if (curr_row[2] ∧ strlen(curr_row[2]) > 0) */

```

349. irods_object_path.

Log

[LDF 2012.12.12.] Added this section.

```
<Scan_Parse_Parameter_Type::get_metadata definition 341> +≡
    if (curr_row[3] ∧ strlen(curr_row[3]) > 0) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ "curr_row[3]" ≪ endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    dcm.irods_object_path = curr_row[3];
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ "dcm.irods_object_path" ≪ dcm.irods_object_path ≪ endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (curr_row[3] ∧ strlen(curr_row[3]) > 0) */
dc_metadata_type_vector.push_back(dcm);
dcm.clear(); } /* for */
mysql_free_result(result);
result = 0;
```

350.

```

⟨ Scan_Parse_Parameter_Type ::get_metadata definition 341 ⟩ +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "dc_metadata_type_vector.size() == " << dc_metadata_type_vector.size() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    for (vector<Dublin_Core_Metadata_Type>::iterator iter = dc_metadata_type_vector.begin();
         iter != dc_metadata_type_vector.end(); ++iter) /* outer for */
    { sql_strm.str("");
      sql_strm << "select S.dublin_core_metadata_sub_id, "
              << "S.dublin_core_element_id, E.element_name, "
              << "S.dublin_core_term_"
              << id, T.term_name, S.value "
              << "from Dublin_Core_Metadata_Sub as S, "
              << "Dublin_Core_Elements as E, Dublin_Core_Terms as T"
              << "where S.dublin_core_metadata_id = "
              << iter->id << " "
              << "and S.dublin_core_element_id = E.dublin_core_element_id "
              << "and S.dublin_core_term_id = T.dublin_core_term_id "
              << "order by S.dublin_core_metadata_sub_id, S.dublin_core_element_id "
              << "S.dublin_core_term_id";
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "sql_strm.str() == " << sql_strm.str() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    status = submit_mysql_query(sql_strm.str(), result, &row_ctr, &field_ctr);
    if (status != 0) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] "
              << "ERROR! In 'Scan_Parse_Parameter_Type::get_metadata':"
              << endl <<
              "'Scan_Parse_Parameter_Type::submit_mysql_query' failed, returning"
              << status <<
              ". " << endl << mysql_error(mysql_ptr) << endl <<
              "Failed to retrieve metadata. " << endl <<
              "Exiting function unsuccessfully with return value 1. " << endl;
        unlock_cerr_mutex();
        ++errors_occurred;
    }
    if (buffer_ptr != 0 & buffer_size > 0) {
        temp_strm.str("");
        temp_strm << "GET "
              << cmd_str << " RESPONSE "
              << filename << "\n"
              << "10 "
              << options <<
              "U "
              << "\Error: Failed to retrieve XML metadata"
              << "for iRODS object"
              << filename << "\n"
              << ' '
              << strcpy(buffer_ptr, temp_strm.str().c_str());
    } /* if (buffer_ptr != 0 & buffer_size > 0) */
    if (result) mysql_free_result(result);
    ++errors_occurred;
    return 1;
} /* if (status != 0) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {

```

```

lock_cerr_mutex();
cerr << "[Thread" << thread_ctr << "] " <<
    "In 'Scan_Parse_Parameter_Type::get_metadata':" << endl <<
    "'Scan_Parse_Parameter_Type::submit_mysql_query' succeeded, returning 0." << endl;
unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

351.

```

⟨ Scan_Parse_Parameter_Type::get_metadata definition 341 ⟩ +≡
if (row_ctr ≡ 0) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] " << "WARNING! In 'Scan_Parse_Parameter_Type::"
        :get_metadata':" << endl << "'row_ctr' == 0, MySQL query returned no rows." <<
        endl << "Failed to retrieve metadata." << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    ++warnings_occurred;
    mysql_free_result(result);
    if (buffer_ptr ≠ 0 ∧ buffer_size > 0) {
        temp_strm.str("");
        temp_strm << "GET " << cmd_str << " RESPONSE\ " << filename << "\" " << "10" << options <<
            "\Warning: No XML metadata" << "for iRODS object" << filename << "\";
        strcpy(buffer_ptr, temp_strm.str().c_str());
    } /* if (buffer_ptr ≠ 0 ∧ buffer_size > 0) */
    ++errors_occurred;
    return 1;
} /* if (row_ctr ≡ 0) */
#endif DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] " <<
        "In 'Scan_Parse_Parameter_Type::get_metadata':" << endl << "'row_ctr' == "
        row_ctr << ", " << endl << "'field_ctr' == " << field_ctr << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

352.

```

⟨ Scan_Parse_Parameter_Type ::get_metadata definition 341 ⟩ +≡
  string curr_element_name;
  string curr_term_name; for (int i = 0; i < row_ctr; ++i)      /* inner for 1 */
  { dem_sub.metadata_id = iter→id;
    if ((curr_row = mysql_fetch_row(result)) ≡ 0) {
      lock_cerr_mutex();
      cerr << "[Thread]" << thread_ctr << "] ERROR!" <<
        "In 'Scan_Parse_Parameter_Type::get_metadata':" << endl <<
        "'mysql_fetch_row' failed:" << endl << mysql_error(mysql_ptr) << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
      unlock_cerr_mutex();
      mysql_free_result(result);
      ++errors_occurred;
      if (buffer_ptr ≠ 0 ∧ buffer_size > 0) {
        temp_strm.str("");
        temp_strm << "GET" << cmd_str << " RESPONSE\" " << filename << "\" " << "10" << options <<
          "U" << "\\" "Error:\\Failed\\to\\retrieve\\XML\\metadata" << "for\\iRODS\\object" <<
          filename << "\\" "from\\database\\";
        strcpy(buffer_ptr, temp_strm.str().c_str());
      } /* if (buffer_ptr ≠ 0 ∧ buffer_size > 0) */
      ++errors_occurred;
      return 1;
    } /* if (curr_row = mysql_fetch_row(result) ≡ 0) */
  else if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread]" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::get_metadata':" <<
      endl << "'mysql_fetch_row' succeeded." << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
}

```

353. Extract data from fields. [LDF 2012.12.12.]

```

⟨ Scan_Parse_Parameter_Type ::get_metadata definition 341 ⟩ +≡

```

```

354. dublin_core_metadata_sub_id.

⟨ Scan_Parse_Parameter_Type ::get_metadata definition 341 ⟩ +≡
  if (curr_row[0] ∧ strlen(curr_row[0]) > 0) {
#if DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ “curr_row[0]” ≪ setw(24) ≪ std::left ≪
      “dublin_core_metadata_sub_id” ≪ curr_row[0] ≪ endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  errno = 0;
  temp_val = strtoul(curr_row[0], 0, 10);
  if (temp_val ≡ ULONG_MAX) {
    lock_cerr_mutex();
    cerr ≪ “[Thread” ≪ thread_ctr ≪ ”]ERROR!” ≪
      “In ‘Scan_Parse_Parameter_Type::get_metadata’:” ≪ endl ≪
      “‘strtoul’ failed, returning ‘ULONG_MAX’:” ≪ endl ≪ strerror(errno) ≪
      endl ≪ “Exiting function unsuccessfully with return value 1.” ≪ endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    ++errors_occurred;
    if (buffer_ptr ≠ 0 ∧ buffer_size > 0) {
      temp_strm.str("");
      temp_strm ≪ “GET” ≪ cmd_str ≪ “RESPONSE\” ≪ filename ≪ “\” ≪
        “10” ≪ options ≪ “U” ≪ “\Error:\Failed to read XML metadata\” ≪
        “for iRODS object\” ≪ filename ≪ “\”;
      strcpy(buffer_ptr, temp_strm.str().c_str());
    } /* if (buffer_ptr ≠ 0 ∧ buffer_size > 0) */
    ++errors_occurred;
    return 1;
  } /* if (temp_val ≡ ULONG_MAX) */
#if DEBUG_COMPILE
  else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ “[Thread” ≪ thread_ctr ≪ ”]” ≪ “In ‘Scan_Parse_Parameter_Type::get_meta\”
      data’:” ≪ endl ≪ “‘strtoul’ succeeded.\” ≪ temp_val” ≪ temp_val ≪ “.” ≪ endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  dcm_sub.id = temp_val;
} /* if (curr_row[0] ∧ strlen(curr_row[0]) > 0) */

```

355. dublin_core_element_id.

Log

[LDF 2012.12.12.] Added this section.

```

⟨ Scan_Parse_Parameter_Type::get_metadata definition 341 ⟩ +≡
    if (curr_row[1] ∧ strlen(curr_row[1]) > 0) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "curr_row[1]" << setw(24) << std::left << "dublin_core_element_id" <<
            curr_row[1] << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    errno = 0;
    temp_val = strtoul(curr_row[1], 0, 10);
    if (temp_val == ULONG_MAX) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] ERROR!" <<
            "In 'Scan_Parse_Parameter_Type::get_metadata':" << endl <<
            "'strtoul' failed, returning 'ULONG_MAX':" << endl << strerror(errno) <<
            endl << "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        mysql_free_result(result);
        ++errors_occurred;
        if (buffer_ptr != 0 ∧ buffer_size > 0) {
            temp_strm.str("");
            temp_strm << "GET" << cmd_str << "RESPONSE\\"" << filename << "\""
                << "10" << options << "U" << "\"Error: Failed to read XML metadata\""
                << "for iRODS object" << filename << "\"";
            strcpy(buffer_ptr, temp_strm.str().c_str());
        } /* if (buffer_ptr != 0 ∧ buffer_size > 0) */
        ++errors_occurred;
        return 1;
    } /* if (temp_val == ULONG_MAX) */
#endif DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "[Thread" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::get_meta"
                "data':" << endl << "'strtoul' succeeded. " << "'temp_val'" << temp_val << "."
                << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    dcm_sub.element_id = temp_val;
} /* if (curr_row[1] ∧ strlen(curr_row[1]) > 0) */

```

356. dublin_core_element_name.

Log

[LDF 2012.12.13.] Added this section.

```
< Scan_Parse_Parameter_Type ::get_metadata definition 341 > +≡
  if (curr_row[2] ∧ strlen(curr_row[2]) > 0) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "curr_row[2]" << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  curr_element_name = curr_row[2];
} /* if (curr_row[2] ∧ strlen(curr_row[2]) > 0) */
```

357. dublin_core_term_id.

Log

[LDF 2012.12.13.] Added this section.

```

⟨ Scan_Parse_Parameter_Type::get_metadata definition 341 ⟩ +≡
  if (curr_row[3] ∧ strlen(curr_row[3]) > 0) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "curr_row[3]" << setw(24) << std::left << "dublin_core_term_id" <<
      curr_row[3] << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  errno = 0;
  temp_val = strtoul(curr_row[3], 0, 10);
  if (temp_val == ULONG_MAX) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "]ERROR!" <<
      "In 'Scan_Parse_Parameter_Type::get_metadata':" << endl <<
      "'strtoul' failed, returning 'ULONG_MAX':" << endl << strerror(errno) <<
      endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    ++errors_occurred;
    if (buffer_ptr != 0 ∧ buffer_size > 0) {
      temp_strm.str("");
      temp_strm << "GET" << cmd_str << "RESPONSE\\"" << filename << "\""
        << "10" << options << "U" << "\"Error: Failed to read XML metadata\""
        << "for iRODS_object" << filename << "\"";
      strcpy(buffer_ptr, temp_strm.str().c_str());
    } /* if (buffer_ptr != 0 ∧ buffer_size > 0) */
    ++errors_occurred;
    return 1;
  } /* if (temp_val == ULONG_MAX) */
#endif DEBUG_COMPILE
  else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::get_meta"
      "data':" << endl << "'strtoul' succeeded. " << "'temp_val'" << temp_val << "."
      << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  dcm_sub.qualifier_id = temp_val;
} /* if (curr_row[3] ∧ strlen(curr_row[3]) > 0) */

```

358. dublin_core_term_name.**Log**

[LDF 2012.12.13.] Added this section.

```

⟨ Scan_Parse_Parameter_Type ::get_metadata definition 341 ⟩ +≡
  if (curr_row[4] ∧ strlen(curr_row[4]) > 0) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "curr_row[4]" << setw(24) << std::left << "dublin_core_term_name" <<
      curr_row[4] << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  curr_term_name = curr_row[4];
} /* if (curr_row[4] ∧ strlen(curr_row[4]) > 0) */

```

359. value.**Log**

[LDF 2012.12.13.] Added this section.

```

⟨ Scan_Parse_Parameter_Type ::get_metadata definition 341 ⟩ +≡
  if (curr_row[5] ∧ strlen(curr_row[5]) > 0) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "curr_row[5]" << setw(24) << std::left << "value" << curr_row[5] << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  dcm_sub.value = curr_row[5];
} /* if (curr_row[5] ∧ strlen(curr_row[5]) > 0) */
iter→metadata_sub_map.insert(make_pair(dcm_sub.id, dcm_sub));
dcm_sub.clear(); /* inner for 1 */
mysql_free_result(result);
result = 0;

```

360. Get attributes. [LDF 2012.12.13.]

Log

[LDF 2012.12.13.] Added this section.

```

⟨ Scan_Parse_Parameter_Type::get_metadata definition 341 ⟩ +≡
    sql_strm.str("");
    sql_strm << "select_dublin_core_metadata_sub_id,attribute,value"
    << "from_Dublin_Core_Attributes where"
    << "dublin_core_metadata_id=_" << iter_id <<
    "order_by_dublin_core_metadata_sub_id,attribute,value";
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "sql_strm.str() == " << sql_strm.str() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    status = submit_mysql_query(sql_strm.str(), result, &row_ctr, &field_ctr);
    if (status != 0) {
        lock_cerr_mutex();
        cerr << "[Thread_"
        << thread_ctr << "]_"
        << "ERROR!_In_`Scan_Parse_Parameter_Type::get_metadata':"
        << endl <<
        "'Scan_Parse_Parameter_Type::submit_mysql_query'_failed,_returning_"
        << status <<
        "._" << endl << mysql_error(mysql_ptr) << endl <<
        "Failed_to_retrieve_metadata."
        << endl <<
        "Exiting_function_unsuccessfully_with_return_value_1."
        << endl;
        unlock_cerr_mutex();
        ++errors_occurred;
    }
    if (result) mysql_free_result(result);
    if (buffer_ptr != 0 & buffer_size > 0) {
        temp_strm.str("");
        temp_strm << "GET_"
        << cmd_str << "_RESPONSE_\""
        << filename << "\_"
        << "1_0_"
        << options <<
        "U_"
        << "\Error:_Failed_to_retrieve_XML_metadata_"
        << "for_uir0DS_object_"
        << filename << "\_from_database\";
        strcpy(buffer_ptr, temp_strm.str().c_str());
    } /* if (buffer_ptr != 0 & buffer_size > 0) */
    ++errors_occurred;
    return 1;
} /* if (status != 0) */
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "[Thread_"
            << thread_ctr << "]_"
            << "In_`Scan_Parse_Parameter_Type::get_metadata':"
            << endl <<
            "'Scan_Parse_Parameter_Type::submit_mysql_query'_succeeded,_returning_0."
            << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

361.

```

⟨ Scan_Parse_Parameter_Type :: get_metadata definition 341 ⟩ +≡
  if (row_ctr ≡ 0) {
#if DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread_"
    << thread_ctr << "]"
    << "In_`Scan_Parse_Parameter_Type::get_meta"
    << "data':"
    << endl << "'row_ctr'==0, MySQL query returned no rows."
    << endl <<
    "No_attributes. Continuing."
    << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  mysql_free_result(result);
  result = 0;
  continue;
} /* if (row_ctr ≡ 0) */

```

362. Attributes present. [LDF 2012.12.13.]

```

⟨ Scan_Parse_Parameter_Type :: get_metadata definition 341 ⟩ +≡
  else /* row_ctr > 0 */
  {
#if DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread_"
    << thread_ctr << "]"
    << "In_`Scan_Parse_Parameter_Type::get_metadata':"
    << endl << "Attributes are present:"
    << endl << "'row_ctr'=="
    << row_ctr << ","
    << endl <<
    "'field_ctr'=="
    << field_ctr << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

363.

```

⟨ Scan_Parse_Parameter_Type::get_metadata definition 341 ⟩ +≡
  unsigned int curr_dc_metadata_sub_id;
  string curr_attribute;
  string curr_value; for (int i = 0; i < row_ctr; ++i)      /* inner for 2 */
  {
    if ((curr_row = mysql_fetch_row(result)) == 0) {
      lock_cerr_mutex();
      cerr << "[Thread]" << thread_ctr << "] ERROR!" <<
        "In 'Scan_Parse_Parameter_Type::get_metadata':" << endl <<
        "'mysql_fetch_row' failed:" << endl << mysql_error(mysql_ptr) << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
      unlock_cerr_mutex();
      ++errors_occurred;
      mysql_free_result(result);
      if (buffer_ptr != 0 & buffer_size > 0) {
        temp_strm.str("");
        temp_strm << "GET" << cmd_str << "RESPONSE\" " << filename << "\" << "10" << options <<
          "U" << "\"Error: Failed to retrieve XML metadata" << "for iRODS object" <<
          filename << " from database\"";
        strcpy(buffer_ptr, temp_strm.str().c_str());
      }      /* if (buffer_ptr != 0 & buffer_size > 0) */
      ++errors_occurred;
      return 1;
    }      /* if (curr_row = mysql_fetch_row(result) == 0) */
  else if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread]" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::get_metadata':" <<
      endl << "'mysql_fetch_row' succeeded." << endl;
    unlock_cerr_mutex();
  }      /* else if (DEBUG) */
}

```

364. dublin_core_metadata_sub_id. [LDF 2012.12.13.]

```

< Scan_Parse_Parameter_Type ::get_metadata definition 341 > +≡
  if (curr_row[0] ∧ strlen(curr_row[0]) > 0) {
#if DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "'curr_row[0]'<==uu" << setw(24) << std::left <<
      "'dublin_core_metadata_sub_id'<==u" << curr_row[0] << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  errno = 0;
  temp_val = strtoul(curr_row[0], 0, 10);
  if (temp_val ≡ ULONG_MAX) {
    lock_cerr_mutex();
    cerr << "[Thread]" << thread_ctr << "]ERROR!uu" <<
      "In 'Scan_Parse_Parameter_Type::get_metadata':" << endl <<
      "'strtoul' failed, returning 'ULONG_MAX':'" << endl << strerror(errno) <<
      endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    ++errors_occurred;
    if (buffer_ptr ≠ 0 ∧ buffer_size > 0) {
      temp_strm.str("");
      temp_strm << "GET" << cmd_str << "RESPONSE\"'" << filename << "\"u" <<
        "1u0u" << options << "Uu" << "\Error: uuFailed to read XML metadatau" <<
        "for uiRODSObject" << filename << "\u";
      strcpy(buffer_ptr, temp_strm.str().c_str());
    } /* if (buffer_ptr ≠ 0 ∧ buffer_size > 0) */
    ++errors_occurred;
    return 1;
  } /* if (temp_val ≡ ULONG_MAX) */
#endif DEBUG_COMPILE
  else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread]" << thread_ctr << "]u" << "In 'Scan_Parse_Parameter_Type::get_meta" <<
      "data':" << endl << "'strtoul' succeeded. uu'temp_val'<==u" << temp_val << "." << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  curr_dc_metadata_sub_id = temp_val;
} /* if (curr_row[0] ∧ strlen(curr_row[0]) > 0) */

```

365. attribute. [LDF 2012.12.13.]

```
<Scan_Parse_Parameter_Type::get_metadata definition 341> +≡
  if (curr_row[1] ∧ strlen(curr_row[1]) > 0) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "curr_row[1]" ≪ setw(24) ≪ std::left ≪ "attribute" ≪ curr_row[1] ≪
      endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  curr_attribute = curr_row[1];
} /* if (curr_row[1] ∧ strlen(curr_row[1]) > 0) */
```

366. value. [LDF 2012.12.13.]

```
<Scan_Parse_Parameter_Type::get_metadata definition 341> +≡
  if (curr_row[2] ∧ strlen(curr_row[2]) > 0) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "curr_row[2]" ≪ setw(24) ≪ std::left ≪ "value" ≪ curr_row[2] ≪ endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  curr_value = curr_row[2];
} /* if (curr_row[2] ∧ strlen(curr_row[2]) > 0) */
```

367.

```

⟨ Scan_Parse_Parameter_Type ::get_metadata definition 341 ⟩ +≡
dc_metadata_sub_iter = iter->metadata_sub_map.find(curr_dc_metadata_sub_id);
if (dc_metadata_sub_iter == iter->metadata_sub_map.end()) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "]WARNING!" <<
        "In 'Scan_Parse_Parameter_Type::get_metadata':" << endl <<
        "'map<unsigned_int,Dublin_Core_Metadata_Sub_Type>::find' failed," <<
        "returning 'iter->metadata_sub_map.end()'." << endl <<
        "Failed to find 'Dublin_Core_Metadata_Sub_Type' in map for"
        "the current attribute." << endl << "Continuing." << endl;
    unlock_cerr_mutex();
    ++warnings_occurred;
}
else /* Found */
{
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Found 'Dublin_Core_Metadata_Sub_Type' in map for"
            "the current attribute." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    dc_metadata_sub_iter->second.attribute_map.insert(make_pair(curr_attribute, curr_value));
} /* else (Found) */
} /* inner for 2 */
mysql_free_result(result);
result = 0; } /* else (row_ctr > 0) */
} /* outer for */

```

368.

```

⟨ Scan_Parse_Parameter_Type ::get_metadata definition 341 ⟩ +≡
#ifndef 0
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        for (vector<Dublin_Core_Metadata_Type>::const_iterator iter =
            dc_metadata_type_vector.begin();
            iter != dc_metadata_type_vector.end(); ++iter) iter->show();
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
#endif

```

369. If *do_output* ≡ *true*: Output *dc_metadata_type_vector* (XML). [LDF 2012.12.13.]

The amount of XML code generated is likely to be fairly large, so a temporary file is always created to hold it. The gain in efficiency that could be achieved by using a **char** buffer or a **stringstream** is outweighed by the inconvenience of writing code for dumping the contents to a file, if the size of the buffer is insufficient. [LDF 2012.12.13.]

```

⟨ Scan_Parse_Parameter_Type ::get_metadata definition 341 ⟩ +≡
if (do_output) { ofstream out_strm;
char temp_filename[] = "/tmp/gwirdsif.XXXXXX";
errno = 0;
int fd = mkstemp(temp_filename);
if (fd ≡ -1) {
    lock_cerr_mutex();
    cerr ≪ "[Thread" ≪ thread_ctr ≪ "] ERROR!" ≪
        "In 'Scan_Parse_Parameter_Type::get_metadata':" ≪ endl ≪
        "'mkstemp' failed, returning -1:" ≪ endl ≪ strerror(errno) ≪ endl ≪
        "Exiting function unsuccessfully with return value 1." ≪ endl;
    unlock_cerr_mutex();
    ++errors_occurred;
if (buffer_ptr ≠ 0 ∧ buffer_size > 0) {
    temp_strm.str("");
    temp_strm ≪ "GET" ≪ cmd_str ≪ " RESPONSE" ≪ filename ≪ "\n" ≪ "10" ≪
        options ≪ "U" ≪ "\"Error: Failed to create server-side temporary file" ≪
        "for XML metadata for iRODS object" ≪ filename ≪ "\n";
    strcpy(buffer_ptr, temp_strm.str().c_str());
} /* if (buffer_ptr ≠ 0 ∧ buffer_size > 0) */
++errors_occurred;
return 1;
} /* if (fd ≡ -1) */
#endif DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "[Thread" ≪ thread_ctr ≪ "] " ≪ "In 'Scan_Parse_Parameter_Type::get_meta" ≪
        "data':" ≪ endl ≪ "'mkstemp' succeeded. temp_filename'" ≪ temp_filename ≪
        "\n";
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
temp_file_vector.push_back(temp_filename);
close(fd); /* We just need the name. [LDF 2012.12.13.] */

```

370.

```
<Scan_Parse_Parameter_Type::get_metadata definition 341> +≡
  out_strm.open(temp_filename);
  if (¬(out_strm ∧ out_strm.is_open() ∧ out_strm.good())) {
    lock_cerr_mutex();
    cerr << "[Thread]" << thread_ctr << "] ERROR! "
    "In 'Scan_Parse_Parameter_Type::get_metadata':" << endl << "'ofstream::open' failed.\n"
    "Failed to open 'ofstream' " << "(" << temp_filename << ') . " << endl <<
    "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
  if (buffer_ptr ≠ 0 ∧ buffer_size > 0) {
    temp_strm.str("");
    temp_strm << "GET" << cmd_str << " RESPONSE" << filename << "\n" << "10"
    options << "U" << "\" Error: Failed to write to server-side temporary file "
    "for XML metadata" << "for iRODS object" << filename << "\n";
    strcpy(buffer_ptr, temp_strm.str().c_str());
  } /* if (buffer_ptr ≠ 0 ∧ buffer_size > 0) */
  ++errors_occurred;
  return 1;
} /* if (¬(out_strm ∧ out_strm.is_open() ∧ out_strm.good())) */
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "[Thread]" << thread_ctr << "] "
      "In 'Scan_Parse_Parameter_Type::get_metadata': "
      "'ofstream::open' succeeded.\n"
      "Opened 'ofstream' (" << temp_filename << ') "
      "successfully.\n"
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

371.

```

⟨ Scan_Parse_Parameter_Type ::get_metadata definition 341 ⟩ +≡
out_strm << "<?xml_version=\"1.0\"?>" << endl << endl;
for (vector<Dublin_Core_Metadata_Type>::const_iterator iter = dc_metadata_type_vector.begin();
     iter != dc_metadata_type_vector.end(); ++iter) {
    status = iter->output(out_strm);
    if (status != 0) {
        lock_cerr_mutex();
        cerr << "[Thread]" << thread_ctr << "] ERROR!" <<
            "In 'Scan_Parse_Parameter_Type::get_metadata':" << endl <<
            "'Dublin_Core_Metadata_Type' failed, returning" << status << "." <<
            endl << "Will try to continue." << endl;
        unlock_cerr_mutex();
        ++errors_occurred;
    if (buffer_ptr != 0 & buffer_size > 0) {
        temp_strm.str("");
        temp_strm << "GET" << cmd_str << "RESPONSE" << filename << "\0" << "10" <<
            options << "U" << "\"Error(server-side): Failed to output XML metadata" <<
            "for iRODS object" << filename << '"';
        strcpy(buffer_ptr, temp_strm.str().c_str());
    } /* if (buffer_ptr != 0 & buffer_size > 0) */
    } /* if */
}
#endif DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread]" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::get_meta-
        data':" << endl << "'Dublin_Core_Metadata_Type' succeeded, returning 0." << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* for */

```

372.

```
<Scan_Parse_Parameter_Type::get_metadata definition 341> +≡
    out_strm.close();
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        temp_strm.str("");
        temp_strm << "cat" << temp_filename;
        lock_cerr_mutex();
        cerr << "Outputting" << temp_filename << ":" << endl << endl;
        status = system(temp_strm.str().c_str());
        cerr << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    if (buffer_ptr != 0 & buffer_size > 0) {
        response.type = Response_Type::SEND_METADATA_TYPE;
        temp_strm.str("");
        temp_strm << "SERVER_SENDING_METADATAFILE" << "\" << filename << "\"" << options << "U";
        response.command = temp_strm.str();
        temp_strm.str("");
        response.local_filename = temp_filename;
        response_deque.push_back(response);
    } /* if (buffer_ptr != 0 & buffer_size > 0) */
} /* if (do_output) */
```

373.

```
<Scan_Parse_Parameter_Type::get_metadata definition 341> +≡
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "'do_output' == 'false'. Not writing XML data to a temporary file." << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

374.

```

⟨ Scan_Parse_Parameter_Type :: get_metadata definition 341 ⟩ +≡
  if (dc_metadata_type_vector_ptr ≠ 0) {
#if DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "{'dc_metadata_type_vector'} != 0. Will copy" <<
      "'dc_metadata_type_vector' to '*dc_metadata_type_vector'." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  dc_metadata_type_vector_ptr->clear();
  *dc_metadata_type_vector_ptr = dc_metadata_type_vector;
#endif /* DEBUG_COMPILE */
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "'dc_metadata_type_vector'->size()' == " << dc_metadata_type_vector_ptr->size() <<
      ". " << endl;
    for (vector<Dublin_Core_Metadata_Type>::const_iterator iter =
        dc_metadata_type_vector_ptr->begin(); iter ≠ dc_metadata_type_vector_ptr->end();
        ++iter) {
      iter->show();
    }
    cerr << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (dc_metadata_type_vector_ptr ≠ 0) */

```

375.

```

⟨ Scan_Parse_Parameter_Type :: get_metadata definition 341 ⟩ +≡
GET_METADATA_IRODS_USER_METADATA:
  if (!do_irods_user_metadata) {
#endif /* DEBUG_COMPILE */
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] In 'Scan_Parse_Parameter_Type::get_metadata':" <<
      endl << "'do_irods_user_metadata'" == "false". N
      ot_retrieving_iRODS_user_metadata" << "(AVU triplets)." << endl <<
      "Exiting function successfully with return value 0." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  return 0;
} /* if (!do_irods_user_metadata) */

```

376.

```
<Scan_Parse_Parameter_Type::get_metadata definition 341> +≡
  if (icommands) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread_"
    << thread_ctr << "]_In_`Scan_Parse_Parameter_Type::get_metadata':"
    << endl << "icommands'_==_`true'." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

377.

```
<Scan_Parse_Parameter_Type::get_metadata definition 341> +≡
  response.clear();
  response.type = Response_Type::COMMAND_ONLY_TYPE;
  temp_strm.str("");
  temp_strm << "env_irodsEnvFile=" << irods_env_filename << "_" << "imeta_ls-ld"
  << filename;
  if (flags & 1U) /* (PID attribute-value-units triplets only) */
  {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread_"
    << thread_ctr << "]_"
    << "In_`Scan_Parse_Parameter_Type::get_metadata':"
    << endl << "flags_&_1U_==_"
    << (flags & 1U) << endl << "Only_retrieving_\"PID\\"_attribute-value_pairs."
    << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  temp_strm << "_PID";
  cmd_str = "PIDS";
} /* if (flags & 1U) (PID attribute-value-units triplets only) */
else cmd_str = "METADATA";
#ifndef DEBUG_COMPILE
if (DEBUG) {
  lock_cerr_mutex();
  cerr << "temp_strm.str()_==_"
  << temp_strm.str() << endl;
  unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

378.

```
< Scan_Parse_Parameter_Type::get_metadata definition 341 > +≡
Irods_Object_Type irods_object;
irods_object.path = filename;
status = irods_object.get_avus_from_irods_system(temp_strm.str(), filename, *this, ctr);
if (status ≠ 0) {
    cerr ≪ "[Thread]" ≪ thread_ctr ≪ "]"
    ≪ "ERROR! In 'Scan_Parse_Parameter_Type::get_metadata':"
    ≪ endl ≪
    " 'Irods_Object_Type::get_avus_from_irods_system' failed, returning "
    ≪ status ≪
    "."
    ≪ endl ≪ "Failed to read iRODS user-defined metadata for iRODS object"
    ≪ " "
    ≪ filename ≪ "."
    ≪ endl ≪ "Exiting function unsuccessfully with return value 1."
    ≪ endl;
    ++errors_occurred;
if (buffer_ptr ≠ 0 ∧ buffer_size > 0) {
    temp_strm.str("");
    temp_strm ≪ "GET"
    ≪ cmd_str ≪ "\RESPONSE"
    ≪ filename ≪ "\"
    ≪ "10"
    ≪ options ≪ "U"
    ≪ "\"Error: Failed to read iRODS user-defined metadata\""
    ≪ "for iRODS object"
    ≪ filename ≪ "\";
    strcpy(buffer_ptr, temp_strm.str().c_str());
}
/* if (buffer_ptr ≠ 0 ∧ buffer_size > 0) */
++errors_occurred;
return 1;
} /* if (status ≠ 0) */
```

379.

```
< Scan_Parse_Parameter_Type::get_metadata definition 341 > +≡
/* !! TODO: LDF 2013.03.22. Try getting the AVUs out of the database; probably at an earlier point
   than here, i.e., when the Irods_Object_Type object is created. */
#if DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr ≪ "[Thread]"
            ≪ thread_ctr ≪ "]"
            ≪ "In 'Scan_Parse_Parameter_Type::get_metadata':"
            ≪ endl ≪
            " 'Irods_Object_Type::get_avus_from_irods_system' succeeded, returning 0."
            ≪ endl;
            irods_object.show("irods_object:");
            unlock_cerr_mutex();
        }
        /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
temp_strm.str("");
string blank_str;
```

380.

```
<Scan_Parse_Parameter_Type::get_metadata definition 341> +≡
  if (flags & 1U) /* (PID attribute-value-units triplets only) */
  {
    temp_strm << "GET_PIDS_RESPONSE\"" << filename << "\"0\" << irods_object.avu_vector.size() <<
      "\" << options << "U\"";
    for (vector<Irods_AVU_Type>::const_iterator iter = irods_object.avu_vector.begin();
         iter != irods_object.avu_vector.end(); ++iter) {
      temp_strm << blank_str << iter->value;
      blank_str = "\"";
    }
    temp_strm << "\"";
  } /* if (flags & 1U) (PID attribute-value-units triplets only) */
```

381.

```
<Scan_Parse_Parameter_Type::get_metadata definition 341> +≡
  else /* (all attribute-value-units triplets) */
  {
    temp_strm << "GET_METADATA_RESPONSE\"" << filename << "\"0\" <<
      irods_object.avu_vector.size() << "\" << options << "U";
    for (vector<Irods_AVU_Type>::const_iterator iter = irods_object.avu_vector.begin();
         iter != irods_object.avu_vector.end(); ++iter) {
      temp_strm << blank_str << "ATTRIBUTE\"" << iter->attribute << "\" << "VALUE\"" << iter->value <<
        "\" << "UNITS\"" << iter->units << "\" << "TIME_SET\"" << iter->time_set << "U";
      blank_str = "\"";
    }
  } /* else (all attribute-value-units triplets) */
#endif DEBUG_COMPILE
if (DEBUG) {
  lock_cerr_mutex();
  cerr << "temp_strm.str() == " << temp_strm.str() << endl;
  unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
if (buffer_ptr != 0 & buffer_size > 0) {
  strcpy(buffer_ptr, temp_strm.str().c_str());
} /* if (buffer_ptr != 0 & buffer_size > 0) */
} /* if (icommands) */
else /* !icommands */
{
  lock_cerr_mutex();
  cerr << "[Thread]" << thread_ctr << "]" <<
    "WARNING! In 'Scan_Parse_Parameter_Type::get_metadata':"
    "'icommands' == 'false'. This case hasn't been programmed yet."
    endl << "Continuing." << endl;
  unlock_cerr_mutex();
} /* else (!icommands) */
```

382.

```
<Scan_Parse_Parameter_Type::get_metadata definition 341> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] " << "Exiting 'Scan_Parse_Parameter_Type::get"
            _metadata'" << "successfully with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; /* End of Scan_Parse_Parameter_Type::get_metadata definition */

```

383. Add metadata.. [LDF 2012.12.14.]

Log

[LDF 2012.12.14.] Added this function.

[LDF 2013.02.13.] Moved the definition of this function from `scprpmtp.web` to this file (`spptfnc1.web`).

384.

```
<Scan_Parse_Parameter_Type::add_metadata definition 384> ≡
int Scan_Parse_Parameter_Type::add_metadata(Response_Type &response){ bool DEBUG = false;
    /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    size_t pos;
    stringstream temp_strm;
    Response_Type new_response;
    new_response.type = Response_Type::COMMAND_ONLY_TYPE;
    int items_written = 0;
    vector<Handle_Type> handle_vector;
    vector<Handle_Type> irods_object_handle_vector;
    bool store = false;
    bool force_add = false;
    bool force_store = false;
    string irods_object_path;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] Entering 'Scan_Parse_Parameter_Type::"
            "add_metadata'." << endl;
        response.show("response:");
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

See also sections 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, and 416.

This code is used in section 544.

385.

```
<Scan_Parse_Parameter_Type::add_metadata definition 384> +≡
  if (response.metadata_options & 1U) force_add = true;
  if (response.metadata_options & 2U) force_store = true;
  if (response.metadata_options & 4U) store = true;
```

386.

```
<Scan_Parse_Parameter_Type::add_metadata definition 384> +≡
  pos = response.local_filename.find("/");
  if (pos ≡ string::npos) {
    response.local_filename.insert(0, "/");
    response.local_filename.insert(0, irods_current_dir);
  }
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "response.local_filename=" ≪ response.local_filename ≪ endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

387. Test for existence of iRODS object. [LDF 2012.12.31.]

Log

[LDF 2012.12.31.] Added this section.

```
<Scan_Parse_Parameter_Type::add_metadata definition 384> +≡
  bool irods_object_exists;
  Irods_AVU_Type curr_avu;
  Irods_Object_Type curr_irods_object(user_id, response.local_filename);
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    curr_irods_object.show("curr_irods_object:");
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

388. Calling `Irods_Object_Type::get_from_database` with the `bool id_only` argument = *false*. This way, the `curr_irods_object.handle_id_vector` will be filled, if there are entries for this iRODS object in the `gwiridsif.Irods_Objects_Handles`. [LDF 2013.01.31.]

```

⟨ Scan_Parse_Parameter_Type::add_metadata definition 384 ⟩ +≡
    status = curr_irods_object.get_from_database(mysql_ptr, false);
    if (status < 0) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "]"
        << "ERROR! In 'Scan_Parse_Parameter_Type::add_metadata':"
        << endl <<
        " 'Irods_Object_Type::get_from_database' failed, returning"
        << status << "."
        << endl <<
        "Exiting function unsuccessfully with return value 1."
        << endl;
        unlock_cerr_mutex();
        ++errors_occurred;
        temp_strm.str("");
        temp_strm << "ADD_METADATA_RESPONSE" << response.remote_filename << "\"
        << "\"
        << response.local_filename << "\"
        << "\"Checking for existence of iRODS object\""
        << '\"'
        << response.local_filename << ", failed.\"";
#if DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "temp_strm.str() == "
        << temp_strm.str()
        << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    new_response.command = temp_strm.str();
    response_deque.push_back(new_response);
    ++errors_occurred;
    return 1;
} /* if (status < 0) */

```

389. Under normal circumstances, this should never happen. It could happen, if iRODS objects are created without using `gwrdifpk`, i.e., using `input` directly. This shouldn't be possible, except for the administrator of `gwrdifpk`. However, as of this date, the database tables in the `gwirdsif` database (`Irods_Objects`, `Irods_AVUs`, etc.) are not needed for the proper operation of `gwrdifpk`. [LDF 2013.06.06.]

```

< Scan_Parse_Parameter_Type :: add_metadata definition 384 > +≡
  else if (status == 0) { irods_object_exists = false;
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread_" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::add_metadata':" <<
      endl << "'Irods_Object_Type::get_from_database' returned 0." << endl <<
      "Will use 'ils' to query iRODS server." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  char buffer[512];
  memset(buffer, 0, 512);
  string temp_filename;
  status = ls(buffer, 512, &temp_filename, 0, response.local_filename, false);
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "ls returned status = " << status << endl << "buffer = " << buffer << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  temp_strm.str();
  temp_strm << buffer;
  temp_strm.clear();
  memset(buffer, 0, 512);
  temp_strm.getline(buffer, 512);
#endif /* DEBUG_COMPILE */
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "buffer after first getline: " << buffer << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  memset(buffer, 0, 512);
  temp_strm.getline(buffer, 512);
#endif /* DEBUG_COMPILE */
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "buffer after second getline: " << buffer << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  errno = 0;
  long int temp_val = strtol(buffer, 0, 10);

```

390.

```

⟨ Scan_Parse_Parameter_Type :: add_metadata definition 384 ⟩ +≡
  if (temp_val == LONG_MAX ∨ temp_val == LONG_MIN) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] " <<
      "ERROR! In 'Scan_Parse_Parameter_Type::add_metadata'" << endl <<
      "'strtol' failed, returning 'temp_val' ('LONG_MAX' or 'LONG_MIN')." <<
      endl << "Failed to read return value of 'ils' command." << endl <<
      "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    temp_strm.str("");
    temp_strm << "ADD_METADATA_RESPONSE_1 \" " << response.remote_filename << "\" " <<
      response.local_filename << "\" " << "\"Checking for existence of iRODS object" << " "
      response.local_filename << ", failed.\" ";
#define DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "temp_strm.str() == " << temp_strm.str() << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  new_response.command = temp_strm.str();
  response_deque.push_back(new_response);
  ++errors_occurred;
  return 1;
} /* if (temp_val == LONG_MAX ∨ temp_val == LONG_MIN) */
else if (temp_val != 0) {
#define DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "iRODS object " << response.local_filename << ", doesn't exist." << endl <<
      "Setting 'irods_object_exists' to 'false'." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  irods_object_exists = false;
} /* else if (temp_val != 0) */

```

391.

Log

[LDF 2013.01.11.] Now sending response with a warning to client.

```
< Scan_Parse_Parameter_Type :: add_metadata definition 384 > +≡
else /* temp_val ≡ 0 */
{
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ "iRODS\object" ≪ response.local_filename ≪ "'\exists." ≪ endl ≪
            "Setting\i rods_object_exists'to'true'." ≪ endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
irods_object_exists = true;
temp_strm.str("");
temp_strm ≪ "ADD\METADATA\RESPONSE\3\" ≪ response.remote_filename ≪ "\" ≪
    "\" ≪ response.local_filename ≪ "\"\WARNING!\i rods_object\exists," ≪
    "but\there\no\entry\for\it\in\the" ≪ "'gwirdsif.Irods_Objects'database\table\
    .\"";
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ "temp_strm.str() == " ≪ temp_strm.str() ≪ endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
new_response.command = temp_strm.str();
response_deque.push_back(new_response);
} /* else (temp_val ≡ 0) */
} /* else if (status ≡ 0) */
```

392. iRODS object found in database. [LDF 2013.06.06.]

This is what should normally happen. [LDF 2013.06.06.]

```
< Scan_Parse_Parameter_Type :: add_metadata definition 384 > +≡
  else
    if (status ≥ 1) {
      irods_object_exists = true;
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread_" << thread_ctr << "] " << "In_`Scan_Parse_Parameter_Type::add_meta\
      data': " << endl << `Irods_Object_Type::get_from_database'`_returned" << status <<
      "(_(>=_1)." << endl << "Set_`irods_object_exists'`_to`'true'." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* else if (status ≥ 1) */
#ifndef DEBUG_COMPILE
if (DEBUG) {
  lock_cerr_mutex();
  curr_irods_object.show("curr_irods_object:");
  unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

393.

Log

[LDF 2012.12.31.] Added this section.

```

⟨ Scan_Parse_Parameter_Type :: add_metadata definition 384 ⟩ +≡
  if (force_add) { /* --force-add, --force or --force-all option used. [LDF 2012.12.31.] */
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] " <<
      "In 'Scan_Parse_Parameter_Type::add_metadata':" << endl <<
      "'--force-add' option used. Will add metadata even if iRODS object "
      "doesn't exist." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (force_add) */
else if (!irods_object_exists) /* response.metadata_options & 1U == false */
{ /* --force-add, --force or --force-all option not used. [LDF 2012.12.31.] */
  lock_cerr_mutex();
  cerr << "[Thread" << thread_ctr << "] " <<
    "ERROR! In 'Scan_Parse_Parameter_Type::add_metadata':" << endl <<
    "iRODS object " << response.local_filename << "' doesn't exist" <<
    "and '--force' option not used." << endl << "Not adding metadata." << endl <<
    "Exiting function unsuccessfully with return value 1." << endl;
  unlock_cerr_mutex();
  ++errors_occurred;
  temp_strm.str("");
  temp_strm << "ADD_METADATA_RESPONSE_1 \" " << response.remote_filename << "\" " << "\" " <<
    response.local_filename << "\" " << "\" iRODS object " << response.local_filename << "\" " <<
    "doesn't exist and --force-add option not used. Metadata " << "not added.\" ";
#endif DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "temp_strm.str() == " << temp_strm.str() << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  new_response.command = temp_strm.str();
  response_deque.push_back(new_response);
  ++errors_occurred;
  return 1;
} /* else if (!irods_object_exists) (force_add == false) */

```

394.

```

⟨ Scan_Parse_Parameter_Type :: add_metadata definition 384 ⟩ +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        curr_irods_object.show("curr_irods_object:");
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
vector<Dublin_Core_Metadata_Type> dc_metadata_vector;
status = parse_metadata(dc_metadata_vector, response);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] ERROR!" <<
        "In 'Scan_Parse_Parameter_Type::add_metadata':" << endl <<
        "'Scan_Parse_Parameter_Type::parse_metadata' failed, returning" << status << "."
        << endl << "Failed to parse metadata in file" << response.temporary_filename << "."
        << endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    temp_strm.str("");
    temp_strm << "ADD_METADATA_RESPONSE_1\""
    << response.remote_filename << "\""
    << response.local_filename << "\" Failed to parse metadata\"";
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "temp_strm.str() == " << temp_strm.str() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    new_response.command = temp_strm.str();
    response_deque.push_back(new_response);
    ++errors_occurred;
    return 1;
} /* if (status ≠ 0) */

```

395.

```

⟨ Scan_Parse_Parameter_Type :: add_metadata definition 384 ⟩ +≡
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "[Thread" << thread_ctr << "] In 'Scan_Parse_Parameter_Type::add_metadata':" <<
                endl << "'Scan_Parse_Parameter_Type::parse_metadata' succeeded, returning 0." <<
                endl << "Parsed metadata in file " << response.temporary_filename <<
                "' successfully." << endl << "'dc_metadata_vector.size()' == " <<
                dc_metadata_vector.size() << endl;
#endif 0
    for (vector<Dublin_Core_Metadata_Type>::const_iterator iter = dc_metadata_vector.begin();
        iter != dc_metadata_vector.end(); ++iter) {
        iter->show();
    }
#endif
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

vector<Dublin_Core_Metadata_Type> existing_metadata_vector;
status = get_metadata(response.local_filename, /* filename */
0, /* flags */
0, /* int *ctr */
0, /* unsigned int options */
0, /* char *buffer_ptr */
0, /* size_t buffer_size */
false, /* do_output */
false, /* do_irods_user_metadata */
&existing_metadata_vector);
if (status ≡ 2) {
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] In 'Scan_Parse_Parameter_Type::add_metadata':" <<
            endl <<
            "'Scan_Parse_Parameter_Type::get_metadata' returned 2: No existing " <<
            "metadata for user " << user_id << " and iRODS object " << response.local_filename <<
            "' in database." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    existing_metadata_vector.clear();
    /* Just to be sure. It should be empty in this case. [LDF 2012.12.21.] */
} /* if (status ≡ 2) */
else if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] ERROR!" <<
        endl << "'Scan_Parse_Parameter_Type::add_metadata':" << endl <<
        "'Scan_Parse_Parameter_Type::get_metadata' failed, returning " << status << "." <<
        endl << "Failed to check for existing metadata in database for file" <<

```

```

    " " << response.temporary_filename << "." << endl <<
    "Exiting function unsuccessfully with return value 1." << endl;
unlock_cerr_mutex();
++errors_occurred;
temp_strm.str("");
temp_strm << "ADD_METADATA_RESPONSE_1\" " << response.remote_filename << "\" << "\" <<
    response.local_filename << "\" << "\"Checking for existing metadata failed\"";
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "temp_strm.str() == " << temp_strm.str() << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
new_response.command = temp_strm.str();
response_deque.push_back(new_response);
++errors_occurred;
return 1;
} /* else if (status != 0) */
#endif DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "existing_metadata_vector.size() == " << existing_metadata_vector.size() << endl;
#endif 0
for (vector<Dublin_Core_Metadata_Type>::const_iterator iter =
    existing_metadata_vector.begin(); iter != existing_metadata_vector.end();
    ++iter) {
    iter->show();
}
cerr << endl;
#endif
unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

396.

```

⟨ Scan_Parse_Parameter_Type :: add_metadata definition 384 ⟩ +≡
  bool matched;
  vector<Dublin_Core_Metadata_Type>::iterator iter;
  vector<Dublin_Core_Metadata_Type>::iterator iter_1;
  string curr_pid; for (iter = dc_metadata_vector.begin(); iter ≠ dc_metadata_vector.end(); ++iter) {
    matched = false;
    for (iter_1 = existing_metadata_vector.begin(); iter_1 ≠ existing_metadata_vector.end(); ++iter_1) {
      if (*iter ≡ *iter_1) {
#if DEBUG_COMPILE
        if (DEBUG) {
          lock_cerr_mutex();
          cerr ≡ "In 'Scan_Parse_Parameter_Type::add_metadata':\n"
          "  *iter == *iter_1. Matching metadata found in database.\n"
          "  Breaking out of inner loop.\n";
          unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        matched = true;
        break;
      } /* if */
    } /* inner for */
    if (matched) {
#if DEBUG_COMPILE
        if (DEBUG) {
          lock_cerr_mutex();
          cerr ≡ "In outer loop:\n"
          "  Matching metadata found in database.\n"
          "  Not calling 'Dublin_Core_Metadata_Type::write_to_database'.\n"
          "Continuing.\n";
          unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        continue;
      } /* if (matched) */
    else /* matched ≡ false */
    {
#if DEBUG_COMPILE
        if (DEBUG) {
          lock_cerr_mutex();
          cerr ≡ "In outer loop:\n"
          "  No matching metadata found in database.\n"
          "  Will call 'Dublin_Core_Metadata_Type::write_to_database'.\n";
          unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        status = iter->write_to_database(mysql_ptr);
        if (status ≠ 0) /* Dublin_Core_Metadata_Type::write_to_database failed */
    {
      lock_cerr_mutex();
      cerr ≡ "[Thread ] ERROR!\n"
      "In 'Scan_Parse_Parameter_Type::add_metadata':\n"
      "'Dublin_Core_Metadata_Type::write_to_database' failed, returning\n";
    }
  }
}

```

```

status << "." << endl << "Failed_to_write_XML_metadata_to_database_for_file" << "\"" <
response.temporary_filename << '.' << endl << "Will_try_to_continue." << endl;
unlock_cerr_mutex();
++errors_occurred;
temp_strm.str("");
temp_strm << "ADD_METADATA_RESPONSE_1\"\" << response.remote_filename << "\" << "\"\" <
response.local_filename << "\" << "\"Failed_to_write_Metadata_to_database\"";
#endif /* DEBUG_COMPILE */
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "temp_strm.str() == " << temp_strm.str() << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
new_response.command = temp_strm.str();
response_deque.push_back(new_response);
continue;
} /* if (status != 0) (Dublin_Core_Metadata_Type::write_to_database failed) */
else /* Dublin_Core_Metadata_Type::write_to_database succeeded */
{ ++items_written;
#endif /* DEBUG_COMPILE */
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread]" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::add_metadata':" <<
        endl << "'Dublin_Core_Metadata_Type::write_to_database' succeeded, " <<
        "returning 0." << endl << "Wrote XML metadata to database for file" << "\"" <<
        response.temporary_filename << "' successfully." << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
curr_pid = "";
status = generate_pids(mysql_ptr, default_handle_prefix, curr_pid, 0,
/* PID vector pointer (vector<string> * */
1, /* Number of PIDS */
0, /* Handle ID vector pointer */
0, /* Handle value ID vector pointer */
true, /* standalone handle server */
"", /* Institute string */
"", /* Suffix string */
&handle_vector, "", /* fifo_pathname */
user_id, username);
if (status != 0) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] ERROR!" <<
        "In 'Scan_Parse_Parameter_Type::add_metadata':" << endl <<
        "'generate_pids' failed, returning" << status << "." << endl <<
        "Failed_to_generate_PID." << endl << "Will_try_to_continue." << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    continue;
} /* if (status != 0) */

```

397.

```
< Scan_Parse_Parameter_Type :: add_metadata definition 384 > +≡
else /* status ≡ 0 */
{
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] " << "In " 'Scan_Parse_Parameter_Type::add_meta\
        data' :" << endl << "'generate_pids' succeeded, returning 0:" << endl <<
        "'curr_pid'" <= curr_pid << endl;
    if (handle_vector.size() ≠ 0) {
        cerr << "Showing " 'handle_vector.back()' ":" << endl;
        handle_vector.back().show("*handle_value.back():");
    } /* if */
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
temp_strm.str("");
temp_strm << "ADD_METADATA_RESPONSE" << response.remote_filename << "\n" << "\n" <<
response.local_filename << "\n" << "\nGenerated handle for metadata: " << curr_pid <<
".\n";
new_response.command = temp_strm.str();
response_deque.push_back(new_response);
} /* else (status ≡ 0) */
```

398.

Log

[LDF 2013.02.28.] Added this section.

```

⟨ Scan_Parse_Parameter_Type :: add_metadata definition 384 ⟩ +≡
status = iter->set_handle_id(mysql_ptr, handle_vector.back().handle_id);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << "[Thread]" << thread_ctr << "] ERROR!" <<
        "In 'Scan_Parse_Parameter_Type::add_metadata':" << endl <<
        "'Dublin_Core_Metadata_Type::set_handle_id' failed, returning" << status << "."
        endl << "Failed to update IDs in 'Dublin_Core_Metadata_Type' object."
        endl << "Will try to continue."
        endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    continue;
} /* if (status ≠ 0) */
#if DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread]" << thread_ctr << "] "
        "In 'Scan_Parse_Parameter_Type::add_metadata':" << endl <<
        "'Dublin_Core_Metadata_Type::set_handle_id' succeeded, returning 0."
        endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

399.

```

⟨ Scan_Parse_Parameter_Type :: add_metadata definition 384 ⟩ +≡
} /* else (Dublin_Core_Metadata_Type::write_to_database succeeded) */
} /* else (matched ≡ false) */
} /* Outer for */

```

400. Get handles for *curr_irods_object*. [LDF 2013.02.08.]

[LDF 2013.03.01.] !! TODO: If there was no **Irods-Object-Type** object, but there was an iRODS object, I'll have to use 'imeta' to get its PID.

Log

[LDF 2013.02.08.] Added this section.

```

⟨ Scan_Parse_Parameter_Type::add_metadata definition 384 ⟩ +≡
if (curr_irods_object.handle_id_vector.size() > 0) {
    status = fetch_handles_from_database(curr_irods_object.handle_id_vector, irods_object.handle_vector,
                                         "IRODS_OBJECT");
    if (status ≠ 0) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] ERROR!" <<
            "In 'Scan_Parse_Parameter_Type::add_metadata':" << endl <<
            "'fetch_handles_from_database' failed, returning" << status << "."
            endl << "Failed to fetch handles for iRODS object" << curr_irods_object.path <<
            "' from 'handlesystem_standalone.handles' database." << endl <<
            "Will try to continue." << endl;
        unlock_cerr_mutex();
        ++errors_occurred;
        temp_strm.str("");
        temp_strm << "ADD_METADATA_RESPONSE_1\""
        << response.remote_filename << "\""
        << response.local_filename << "\""
        << "Failed to fetch handles for iRODS object" <<
            "'"
            << curr_irods_object.path << "\";
        new_response.command = temp_strm.str();
        response_deque.push_back(new_response);
    } /* if (status ≠ 0) */
#endif DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] "
    "In 'Scan_Parse_Parameter_Type::add_metadata':" << endl <<
    "'fetch_handles_from_database' succeeded, returning 0."
    << endl <<
    "'irods_object_handle_vector.size()' == "
    << irods_object.handle_vector.size() << endl;
    if (irods_object.handle_vector.size() > 0) {
        for (vector<Handle_Type>::const_iterator iter = irods_object.handle_vector.begin();
             iter ≠ irods_object.handle_vector.end(); ++iter) {
            iter->show();
        } /* for */
        cerr << endl;
    } /* if (irods_object.handle_vector.size() > 0) */
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (curr_irods_object.handle_id_vector.size() > 0) */

```

401. Add handle values to the handles for the metadata. [LDF 2013.01.11.]

Log

[LDF 2013.01.11.] Added this section.

```

⟨ Scan_Parse_Parameter_Type :: add_metadata definition 384 ⟩ +≡
for (vector<Handle_Type>::iterator iter = handle_vector.begin(); iter != handle_vector.end();
    ++iter) { temp_strm.str(""); temp_strm << "Qualified_Dublin_Core_XML_Metadata_for_iRODS_object"
    response.local_filename << ".";
status = iter->add_value(mysql_ptr, Handle_Value_Type::DC_METADATA_INDEX, "DC_METADATA",
    temp_strm.str(), user_id);
if (status != 0) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] ERROR!" <<
        "In 'Scan_Parse_Parameter_Type::add_metadata':" << endl <<
        "'Handle_Value_Type::add_value' failed, returning" << status << "." <<
        endl << "Failed to add handle value for handle" << iter->handle << "." << endl <<
        "Will try to continue." << endl;
unlock_cerr_mutex();
++errors_occurred;
continue;
} /* if (status != 0) */
#endif DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::add_meta"
        "data':" << endl << "'Handle_Value_Type::add_value' succeeded, returning 0." << endl;
unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

- 402.

!! TODO: If there's no **Irods_Object_Type** object, then I could try to get the PID using 'imeta' after calling 'ils'. This shouldn't be necessary but would be good as a backup.

ils is not currently called in **Irods_Object_Type::get_from_database**. I could implement this and call 'imeta' there to try to extract the PIDs. I should then check to see if they exist.

Or perhaps I should just check the 'handlesystem_standalone' database in the first place. Or maybe both and synchronize it with the iRODS user metadata, if necessary. [LDF 2013.03.01.]

Log

[LDF 2013.01.31.] Added this section.

```

⟨ Scan_Parse_Parameter_Type :: add_metadata definition 384 ⟩ +≡

```

403.

Log

[LDF 2013.02.08.] Added this section.

```

⟨ Scan_Parse_Parameter_Type :: add_metadata definition 384 ⟩ +≡
if (irods_object_handle_vector.size() ≡ 0) {
    status = iter-add_value(mysql_ptr, Handle_Value_Type :: IRODS_OBJECT_REF_INDEX,
                           "IRODS_OBJECT_REF", response.local_filename, user_id);
    if (status ≠ 0) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] ERROR!" <<
            "In 'Scan_Parse_Parameter_Type::add_metadata':" << endl <<
            "'Handle_Value_Type::add_value' failed, returning" << status << "." <<
            endl << "Failed to add handle value for (non-existent)" << "iRODS object" <<
            response.local_filename << "' " << "with type 'IRODS_OBJECT' to handle" << "' " <<
            iter-handle << "' ." << endl << "Will try to continue." << endl;
        unlock_cerr_mutex();
        ++errors_occurred;
        temp_strm.str("");
        temp_strm << "ADD_METADATA_RESPONSE_2" << "\n" << response.remote_filename <<
            "\n" << "\n" << response.local_filename << "\n" <<
            "\nFailed to add handle value for (non-existent)" << "iRODS object" <<
            response.local_filename << "' " << "with type 'IRODS_OBJECT' to handle" << "' " <<
            iter-handle << "' .\n";
        new_response.command = temp_strm.str();
        response_deque.push_back(new_response);
    } /* if (status ≠ 0) */
    else {
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "[Thread" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::add_meta" <<
                "data':" << endl << "'Handle_Value_Type::add_value' succeeded, returning 0." <<
                endl << "Added handle value for (non-existent)" << "iRODS object" <<
                response.local_filename << "' " << "with type 'IRODS_OBJECT' to handle" << "' " <<
                iter-handle << "' successfully." << endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        temp_strm.str("");
        temp_strm << "ADD_METADATA_RESPONSE_0\n" << response.remote_filename << "\n" << "\n" <<
            response.local_filename << "\n" << "\nAdded handle value for (non-existent)" <<
            "iRODS object" << response.local_filename << "' " <<
            "with type 'IRODS_OBJECT' to handle" << "' " << iter-handle << "' successfully.\n";
        new_response.command = temp_strm.str();
        response_deque.push_back(new_response);
    } /* else */
} /* if (irods_object_handle_vector.size() ≡ 0) */

```

404.

```

< Scan_Parse_Parameter_Type :: add_metadata definition 384 > +≡
  for (vector<Handle_Type>::iterator iter_2 = irods_object_handle_vector.begin();
       iter_2 != irods_object_handle_vector.end(); ++iter_2) { status = iter->add_value(mysql_ptr,
         Handle_Value_Type::IRODS_OBJECT_PID_INDEX, "IRODS_OBJECT_PID", iter_2->handle, user_id);
  if (status != 0) {
    lock_cerr_mutex();
    cerr << "[Thread_" << thread_ctr << "] ERROR!" <<
      "In 'Scan_Parse_Parameter_Type::add_metadata':" << endl <<
      "'Handle_Value_Type::add_value' failed, returning" << status << "." <<
      endl << "Failed to add handle value for handle" << iter_2->handle <<
      " with type 'IRODS_OBJECT_PID' to handle" << iter_2->handle << endl <<
      "Will try to continue." << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    temp_strm.str("");
    temp_strm << "ADD_METADATA_RESPONSE_2" << "\\" << response.remote_filename << "\\" <<
      "\\" << response.local_filename << "\\" << "\Failed to add handle value for handle" <<
      "\\" << iter_2->handle << "\\" << "with type 'IRODS_OBJECT_PID' to handle" << "\\" <<
      iter->handle << "\\";
    new_response.command = temp_strm.str();
    response_deque.push_back(new_response);
  } /* if (status != 0) */
  else {
#if DEBUG_COMPILE
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "[Thread_" << thread_ctr << "] In 'Scan_Parse_Parameter_Type::add_meta\"
        data':" << endl << "'Handle_Value_Type::add_value' succeeded, returning 0." <<
        endl << "Added handle value for handle" << iter_2->handle <<
        "' with type 'IRODS_OBJECT_PID' to handle" << iter->handle <<
        "' successfully." << endl;
      unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    temp_strm.str("");
    temp_strm << "ADD_METADATA_RESPONSE_0\"\\\" << response.remote_filename << "\\" << "\\" <<
      response.local_filename << "\\" << "\Added handle value for handle" << iter_2->handle <<
      "\\" << "with type 'IRODS_OBJECT_PID' to handle" << iter->handle <<
      "\\" successfully\"";
    new_response.command = temp_strm.str();
    response_deque.push_back(new_response);
  } /* else */
}

```

405. Cross-reference with *type* ≡ "IRODS_OBJECT_REF". [LDF 2013.02.08.]

Log

[LDF 2013.02.08.] Added this section.

```

⟨ Scan_Parse_Parameter_Type :: add_metadata definition 384 ⟩ +≡
status = iter->add_value(mysql_ptr, Handle_Value_Type :: IRODS_OBJECT_REF_INDEX,
    "IRODS_OBJECT_REF", curr_irods_object.path, user_id);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] ERROR!" <<
        "In‘Scan_Parse_Parameter_Type::add_metadata’::" << endl <<
        "‘Handle_Value_Type::add_value’_failed,_returning" << status << "."
        << endl << "Failed_to_add_handle_value_for_iRODS_object" << "'"
        << curr_irods_object.path << ",_with_type‘IRODS_OBJECT_REF’_to_handle"
        << "'." << endl <<
        "Will_try_to_continue." << endl;
unlock_cerr_mutex();
++errors_occurred;
temp_strm.str("");
temp_strm << "ADD_METADATA_RESPONSE_2" << "\"
    << response.remote_filename << "\"
    << response.local_filename << "\"
    << "\Failed_to_add_handle_value_for_iRODS_object" <<
    "'"
    << curr_irods_object.path << ",_with_type‘IRODS_OBJECT_REF’_to_handle"
    << "'"
    << iter->handle << '\';
new_response.command = temp_strm.str();
response_deque.push_back(new_response);
} /* if (status ≠ 0) */
else {
#if DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] "
            << "In‘Scan_Parse_Parameter_Type::add_meta"
            "data’::" << endl <<
            "‘Handle_Value_Type::add_value’_succeeded,_returning_0."
            << endl <<
            "Added_handle_value_for_iRODS_object" << "'"
            << curr_irods_object.path <<
            ",_with_type‘IRODS_OBJECT_REF’_to_handle"
            << "'"
            << iter->handle <<
            ",_successfully." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
temp_strm.str("");
temp_strm << "ADD_METADATA_RESPONSE_0\"
    << response.remote_filename << "\"
    << response.local_filename << "\"
    << "\Added_handle_value_for_iRODS_object" << "'"
    << curr_irods_object.path << ",_with_type‘IRODS_OBJECT_REF’_to_handle"
    << "'"
    << iter->handle << ",_successfully\";
new_response.command = temp_strm.str();
response_deque.push_back(new_response);
} /* else */
}

```

406.

Log

[LDF 2013.02.08.] Added this section.

```

⟨ Scan_Parse_Parameter_Type::add_metadata definition 384 ⟩ +≡
status = iter_2→add_value(mysql_ptr, Handle_Value_Type::DC_METADATA_PID_INDEX,
    "DC_METADATA_PID", iter→handle, user_id);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << "[Thread_" << thread_ctr << "] ERROR! " <<
        "In 'Scan_Parse_Parameter_Type::add_metadata': " << endl <<
        "'Handle_Value_Type::add_value' failed, returning " << status << ". " <<
        endl << "Failed to add handle value for handle " << iter→handle <<
        " with type 'DC_METADATA_PID' to handle " << '"' << iter_2→handle << '"' << endl <<
        "Will try to continue." << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    temp_strm.str("");
    temp_strm << "ADD_METADATA_RESPONSE_1\\" " << response.remote_filename << "\" " << "\" " <<
        response.local_filename << "\" " << "\" Failed to add handle value for handle " <<
        '"' << iter→handle << '"' << " with type 'DC_METADATA_PID' to handle " << '"' <<
        iter_2→handle << '"';
    new_response.command = temp_strm.str();
    response_deque.push_back(new_response);
} /* if (status ≠ 0) */
else {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread_" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::add_meta-
            data': " << endl << "'Handle_Value_Type::add_value' succeeded, returning 0. " <<
            endl << "Added handle value for handle " << '"' << iter→handle <<
            " with type 'IRODS_OBJECT_PID' to handle " << '"' << iter_2→handle <<
            " successfully." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    temp_strm.str("");
    temp_strm << "ADD_METADATA_RESPONSE_0\\" " << response.remote_filename << "\" " << "\" " <<
        response.local_filename << "\" " << "\" Added handle value for handle " << iter→handle <<
        " " << " with type 'DC_METADATA_PID' to handle " << '"' << iter_2→handle <<
        " successfully\"";
    new_response.command = temp_strm.str();
    response_deque.push_back(new_response);
} /* else */

```

407.

Log

[LDF 2013.02.08.] Added this section.

[LDF 2013.06.06.] BUG FIX: Now calling imeta rm to remove the AVU, if it already exists.

```
< Scan_Parse_Parameter_Type :: add_metadata definition 384 > +≡
temp_strm.str("");
temp_strm << "env_irodsEnvFile=" << irods_env_filename << "imeta_rm_d\""
<< curr_irods_object.path << "\"DC_METADATA_PID\""
<< iter->handle << "\">/dev/null>&1;\""
<< "env_irodsEnvFile=" << irods_env_filename << "imeta_add_d\""
<< curr_irods_object.path <<
"\"DC_METADATA_PID\""
<< iter->handle << "\"";
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "temp_strm.str() == "
        << temp_strm.str() << endl;
    unlock_cerr_mutex();
}
#endif /* DEBUG_COMPILE */
```

408.

```

⟨ Scan_Parse_Parameter_Type :: add_metadata definition 384 ⟩ +≡
status = system(temp_strm.str().c_str());
if (status == -1 || !WIFEXITED(status)) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] " << "ERROR! In 'Scan_Parse_Parameter_Type::"
        add_metadata': " << "'system' failed, returning" << status << ". " << endl;
    if (WIFEXITED(status)) cerr << "WEXITSTATUS(status)==" << WEXITSTATUS(status) << endl;
    else cerr << "Process failed to exit." << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    temp_strm.str("");
    temp_strm << "ADD_METADATA_RESPONSE_1\\" << response.remote_filename <<
        "\\" << "\\" << response.local_filename << "\\" <<
        "\\"Call_to_‘imeta’_failed_to_exit..Failed_to_add_AVU\\" <<
        "with_type_‘DC_METADATA_PID’_and_value" << "\\" << iter_handle <<
        "\_to_iRODS_object" << "\\" << curr_irods_object.path << ".\\"";
    new_response.command = temp_strm.str();
    response_deque.push_back(new_response);
} /* (status == -1 || !WIFEXITED(status)) */
else if (WEXITSTATUS(status) != 0) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] " << "ERROR! In 'Scan_Parse_Parameter_Type::"
        add_metadata': " << "'imeta' command_(called_via_‘system’) failed, "
        "returning" << WEXITSTATUS(status) << ". " << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    temp_strm.str("");
    temp_strm << "ADD_METADATA_RESPONSE_1\\" << response.remote_filename <<
        "\\" << "\\" << response.local_filename << "\\" <<
        "\\"Call_to_‘imeta’_failed_with_exit_status\\" << WEXITSTATUS(status) <<
        "\_Failed_to_add_AVU\\" << "with_type_‘DC_METADATA_PID’_and_value" << "\\" <<
        iter_handle << "\_to_iRODS_object" << "\\" << curr_irods_object.path << ".\\"";
    new_response.command = temp_strm.str();
    response_deque.push_back(new_response);
} /* else if (WEXITSTATUS(status) != 0) */
else /* imeta command succeeded. */
{
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::add_meta"
            data': " << "'system' succeeded, WEXITSTATUS(status)==" << WEXITSTATUS(status) <<
            ". " << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    temp_strm.str("");
    temp_strm << "ADD_METADATA_RESPONSE_0\\" << response.remote_filename << "\\" <<
        "\\" << response.local_filename << "\\" << "\\"Call_to_‘imeta’_succeeded.."
        "Added_AVU\\" << "with_type_‘DC_METADATA_PID’_and_value" << "\\" << iter_handle <<
        "\_to_iRODS_object" << "\\" << curr_irods_object.path << ".\\"";
}

```

```
new_response.command = temp_strm.str();
response_deque.push_back(new_response);
curr_avu.clear();
curr_avu.set("DC_METADATA_PID", iter->handle);
curr_avu.irods_object_id = curr_irods_object.id;
curr_irods_object.avu_vector.push_back(curr_avu);
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    curr_irods_object.show("curr_irods_object:");
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
status = curr_avu.write_to_database(mysql_ptr, thread_ctr);
if (status != 0) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] " << "ERROR! In 'Scan_Parse_Parameter_Type::\n"
        add_metadata': " << 'Irods_AVU_Type::write_to_database' failed, returning " <<
        status << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    temp_strm.str("");
    temp_strm << "ADD_METADATA_RESPONSE_1\" " << response.remote_filename <<
    "\" " << "\" " << response.local_filename << "\" " <<
    "\"Call to 'Irods_AVU_Type::write_to_database', returning " << status << "\"";
    new_response.command = temp_strm.str();
    response_deque.push_back(new_response);
    temp_strm.str("");
} /* if (status != 0) */
#ifndef DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] " <<
        "In 'Scan_Parse_Parameter_Type::add_metadata': " <<
        "'Irods_AVU_Type::write_to_database' succeeded, returning 0." << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
curr_avu.clear();
} /* else (imeta command succeeded.) */
} /* inner for */
} /* outer for */
```

409.

```

⟨ Scan_Parse_Parameter_Type :: add_metadata definition 384 ⟩ +≡
  if (store) {
    if (¬items_written) {
      lock_cerr_mutex();
      cerr << "[Thread" << thread_ctr << "]WARNING!" <<
        "In‘Scan_Parse_Parameter_Type::add_metadata’::" << endl <<
        "'--store'option_usedbut‘items_written’==‘false’." << endl <<
        "Not_storing_metadata_file_in_iRODS_object.Continuing." << endl;
      unlock_cerr_mutex();
      temp_strm.str("");
      temp_strm << "ADD_METADATA_RESPONSE_4\\"" << response.remote_filename <<
        "\\" << "\\" << response.local_filename << "\\""--store_option_used" <<
        "but_no_XML_metadata_written_to_database.Not_storing_XML_file" <<
        "in_an_iRODS_object.\\"";
    #if DEBUG_COMPILE
      if (DEBUG) {
        lock_cerr_mutex();
        cerr << "temp_strm.str() == " << temp_strm.str() << endl;
        unlock_cerr_mutex();
      } /* if (DEBUG) */
    #endif /* DEBUG_COMPILE */
      new_response.command = temp_strm.str();
      response_deque.push_back(new_response);
    } /* if (¬items_written) */
    else /* items_written ≡ true */
    {
    #if DEBUG_COMPILE
      if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "]" << "In‘Scan_Parse_Parameter_Type::add_metadata’::" <<
          endl << "'--store'option_usedand‘items_written’==‘true’." << endl <<
          "Will_store_metadata_file_in_iRODS_object." << endl;
        unlock_cerr_mutex();
      } /* if (DEBUG) */
    #endif /* DEBUG_COMPILE */
      if (irods_object_handle_vector.size() ≡ 0 ∨ handle_vector.size() ≡ 0) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "]WARNING!" <<
          "In‘Scan_Parse_Parameter_Type::add_metadata’::" << endl <<
          "irods_object_handle_vector.size() == 0 and/or" <<
          "handle_vector.size() == 0." << endl << "Two_PIDs_required_for_passing_to" <<
          "Scan_Parse_Parameter_Type::store_dc_metadata:" << endl << "At_least_one_is_mising." << endl <<
          "Not_calling‘Scan_Parse_Parameter_Type::store_dc_metadata’." << endl <<
          "Continuing." << endl;
        unlock_cerr_mutex();
        temp_strm.str("");
        temp_strm << "ADD_METADATA_RESPONSE_5\\"" << response.remote_filename <<
          "\\" << "\\" << response.local_filename << "\\"Failed_to_store" <<
          "Dublin_Core_metadata_in_iRODS_object.\\"";
    #if DEBUG_COMPILE
      if (DEBUG) {

```

```

lock_cerr_mutex();
cerr << "temp_strm.str() == " << temp_strm.str() << endl;
unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
++warnings_occurred;
new_response.command = temp_strm.str();
response_deque.push_back(new_response);
} /* if */

```

410.

```

⟨ Scan_Parse_Parameter_Type::add_metadata definition 384 ⟩ +≡
else
if (irods_object_handle_vector.back().handle.empty() ∨ handle_vector.back().handle.empty()) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "]WARNING!" <<
        "In 'Scan_Parse_Parameter_Type::add_metadata':" << endl <<
        "'irods_object_handle_vector.back().handle' and/or "
        "'handle_vector.back().handle' is empty." << endl <<
        "Two PIDs required in 'Scan_Parse_Parameter_Type::store_dc_metadata':" <<
        endl << "At least one is missing." << endl <<
        "Not calling 'Scan_Parse_Parameter_Type::store_dc_metadata'." <<
        endl << "Continuing." << endl;
    unlock_cerr_mutex();
    temp_strm.str("");
    temp_strm << "ADD_METADATA_RESPONSE\5\" << response.remote_filename <<
        "\\" << "\"" << response.local_filename << "\"\Failed to store"
        "Dublin-Core_metadata in iRODS object.\"";
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "temp_strm.str() == " << temp_strm.str() << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
++warnings_occurred;
new_response.command = temp_strm.str();
response_deque.push_back(new_response);
} /* else if */

```

411.

Log

[LDF 2013.03.07.] Added this section.

```

<Scan_Parse_Parameter_Type::add_metadata definition 384> +≡
else
  if (irods_object_handle_vector.back().handle_id ≡ 0 ∨ handle_vector.back().handle_id ≡ 0) {
    lock_cerr_mutex();
    cerr << "[Thread_" << thread_ctr << "]WARNING!" <<
      "In `Scan_Parse_Parameter_Type::add_metadata':" << endl <<
      "'irods_object_handle_vector.back().handle_id' == 0 and/or"
      "'handle_vector.back().handle_id' == 0." << endl << "Two handle IDs required in"
      "'Scan_Parse_Parameter_Type::store_dc_metadata':"
      endl << "At least one is missing." << endl <<
      "Not calling `Scan_Parse_Parameter_Type::store_dc_metadata'." <<
      endl << "Continuing." << endl;
    unlock_cerr_mutex();
    temp_strm.str("");
    temp_strm << "ADD_METADATA_RESPONSE" << response.remote_filename <<
      "\\" << "\"" << response.local_filename << "\"Failed to store"
      "Dublin Core metadata in iRODS object."";
#endif DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "temp_strm.str() == " << temp_strm.str() << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  ++warnings_occurred;
  new_response.command = temp_strm.str();
  response_deque.push_back(new_response);
} /* else if */

```

412.

```

< Scan_Parse_Parameter_Type :: add_metadata definition 384 > +≡
else {
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "irods_object_handle_vector.back().handle'" ==
            " " << irods_object_handle_vector.back().handle << "'"
            "'handle_vector.back().handle'" == " " << " " << handle_
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

413.

```

⟨ Scan_Parse_Parameter_Type :: add_metadata definition 384 ⟩ +≡
status = store_dc_metadata(response, irods_object_handle_vector.back(), handle_vector.back(), force_store,
    irods_object_path);
if (status == 2) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "]WARNING!" <<
        "In 'Scan_Parse_Parameter_Type::add_metadata':" << endl <<
        "'Scan_Parse_Parameter_Type::store_dc_metadata' returned 2:" << endl <<
        "An iRODS object for " << response.remote_filename << " already exists" << endl <<
        "and the '--force-store' option was not used." << endl << "Continuing." << endl;
unlock_cerr_mutex();
temp_strm.str("");
temp_strm << "ADD_METADATA_RESPONSE_5" << "\n" <<
    response.remote_filename << "\n" << "\n" << response.local_filename <<
    "\n\"iRODS object" << "already exists for Dublin Core metadata" <<
    "and '--force-store' option not used.\n";
#endif /* DEBUG_COMPILE */
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "temp_strm.str() == " << temp_strm.str() << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
++warnings_occurred;
new_response.command = temp_strm.str();
response_deque.push_back(new_response);
} /* else if (status != 0) */
else if (status != 0) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "]WARNING!" <<
        "In 'Scan_Parse_Parameter_Type::add_metadata':" << endl <<
        "'Scan_Parse_Parameter_Type::store_dc_metadata' failed, " << "returning" <<
        status << "." << endl << "Failed to store Dublin Core metadata in iRODS object." <<
        "Continuing." << endl;
unlock_cerr_mutex();
temp_strm.str("");
temp_strm << "ADD_METADATA_RESPONSE_5" << "\n" <<
    response.remote_filename << "\n" << "\n" << response.local_filename << "\n" <<
    "Failed to store" << "\n" << "Dublin Core metadata in iRODS object.\n";
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "temp_strm.str() == " << temp_strm.str() << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
++warnings_occurred;
new_response.command = temp_strm.str();
response_deque.push_back(new_response);
} /* else if (status != 0) */
else /* status == 0 */

```

```

{
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread_" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::add_metadata'" << endl << "Scan_Parse_Parameter_Type::store_dc_metadata'succeeded," <<
            "returning 0." << endl << "irods_object_path'" <= " " << irods_object_path << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
temp_strm.str("");
temp_strm << "ADD_METADATA_RESPONSE_0" << "\\" << response.remote_filename << "\"" << "\\" << "\" <<
    response.local_filename << "\"" << "Stored" << "DublinCore_metadata_in_iRODS_object" <<
        "\"" << irods_object_path << "' successfully.\"";
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "temp_strm.str() == " << temp_strm.str() << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
new_response.command = temp_strm.str();
response_deque.push_back(new_response);
} /* else (status == 0) */
} /* else */
} /* else (items_written == true) */
} /* if (store) */

```

414.

```

⟨ Scan_Parse_Parameter_Type::add_metadata definition 384 ⟩ +≡
    if (items_written > 0) {
        temp_strm.str("");
        temp_strm << "ADD_METADATA_RESPONSE_0" << response.remote_filename << "\"" << "\\" << "\" <<
            response.local_filename << "\"" << "(Success)" << "";
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "temp_strm.str() == " << temp_strm.str() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    new_response.command = temp_strm.str();
    response_deque.push_back(new_response);
} /* if (items_written > 0) */

```

415.

Log

[LDF 2013.01.04.] Now sending response.
 [LDF 2013.01.11.] Improved response.

```
< Scan_Parse_Parameter_Type :: add_metadata definition 384 > +≡
else /* items_written ≡ 0 */
{
  temp_strm.str("");
  temp_strm ≪ "ADD_METADATA_RESPONSE_2_" ≪ response.remote_filename ≪ "\" ≪ "\" ≪
    response.local_filename ≪ "\" ≪ "No_XML_metadata" ≪ "written_to_database";
  if (matched) temp_strm ≪ ":_Matching_XML_data_already_present_in_database.";
  else temp_strm ≪ ".";
  temp_strm ≪ "\";
#endif DEBUG_COMPILE
if (DEBUG) {
  lock_cerr_mutex();
  cerr ≪ "temp_strm.str() == " ≪ temp_strm.str() ≪ endl;
  unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
new_response.command = temp_strm.str();
response_deque.push_back(new_response);
#endif DEBUG_COMPILE
if (DEBUG) {
  lock_cerr_mutex();
  cerr ≪ "'item_written' == 0" ≪ endl;
  unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* else (items_written ≡ 0) */
```

416.

```
< Scan_Parse_Parameter_Type :: add_metadata definition 384 > +≡
#ifndef DEBUG_COMPILE
if (DEBUG) {
  lock_cerr_mutex();
  cerr ≪ "[Thread_" ≪ thread_ctr ≪ "]_Exiting_Scan_Parse_Parameter_Type::add_metadata" ≪
    "successfully with return value 0." ≪ endl;
  unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
return 0; } /* End of Scan_Parse_Parameter_Type::add_metadata definition */
```

417. Parse metadata.. [LDF 2012.12.14.]

Log

[LDF 2012.12.14.] Added this function.
 [LDF 2013.02.13.] Moved the definition of this function from `scprpmtp.web` to this file (`spptfnc1.web`).

418.

```
<Scan_Parse_Parameter_Type::parse_metadata definition 418> ≡
int Scan_Parse_Parameter_Type::parse_metadata(vector<Dublin_Core_Metadata_Type>
    &dc_metadata_vector, Response_Type &response){ bool DEBUG = false; /* true */
set_debug_level(DEBUG, 0, 0);
int status;
stringstream temp_strm;
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] " <<
        "Entering 'Scan_Parse_Parameter_Type::parse_metadata'." << endl;
    response.show("response:");
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

See also sections 419, 420, 421, 422, 423, 424, 425, 426, and 427.

This code is used in section 544.

419. Test whether *response.temporary_file* contains metadata tags. [LDF 2013.01.10.]

```
<Scan_Parse_Parameter_Type::parse_metadata definition 418> +≡
temp_strm.str("");
temp_strm << "((cat" << response.temporary_filename <<
    ",|grep-b\"metadata\")&&echo\"0\"||echo\"1\"";
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "temp_strm.str() == " << temp_strm.str() << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
FILE *fp = popen(temp_strm.str().c_str(), "r");
if (fp == 0) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] " <<
        "ERROR! In 'Scan_Parse_Parameter_Type::p\
        arse_metadata':" << endl << "'popen' failed, returning NULL." << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    return 1;
} /* if (fp == 0) */
```

420.

```

⟨ Scan_Parse_Parameter_Type::parse_metadata definition 418 ⟩ +≡
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "[Thread]" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::parse_me"
                tadata':" << endl << "'popen' succeeded." << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
char line_buffer[256][512];
char *result;
int i = 0;
for ( ; ; ++i) {
    if (i == 256) {
        lock_cerr_mutex();
        cerr << "[Thread] " << thread_ctr << "] " <<
            "WARNING! In 'Scan_Parse_Parameter_Type::parse_metadata':" << endl <<
            "'popen' output exceeds maximum number of lines (255)." << endl <<
            "Ignoring remaining lines. Continuing." << endl;
        unlock_cerr_mutex();
        --i;
        ++warnings_occurred;
        break;
    } /* if (i == 256) */
    memset(line_buffer[i], 0, 512);
    result = fgets(line_buffer[i], 512, fp);
    if (result == 0) {
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "'fgets' returned NULL. Breaking." << endl << "'feof(fp)' == " << feof(fp) <<
                endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        break;
    } /* if (result == 0) */
    if (strlen(line_buffer[i]) == 511) {
        lock_cerr_mutex();
        cerr << "[Thread] " << thread_ctr << "] " <<
            "WARNING! In 'Scan_Parse_Parameter_Type::parse_metadata':" << endl <<
            "Length of 'line_buffer[" << i << "]' read from 'popen' output" <<
            "exceeds maximum number of characters (510):" << endl << "'strlen(line_buffer[" <<
                i << "])' == " << strlen(line_buffer[i]) << endl << "Will try to continue." << endl;
        unlock_cerr_mutex();
        ++warnings_occurred;
    } /* if (strlen(line_buffer[i]) == 511) */
#endif DEBUG_COMPILE
        if (DEBUG) {

```

```
lock_cerr_mutex();
cerr << "line_buffer[" << i << "] == " << line_buffer[i] << endl;
unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* for */
pclose(fp);
fp = 0;
#if DEBUG_COMPILE
if (DEBUG) {
lock_cerr_mutex();
cerr << "After loop: 'i' == " << i << endl;
unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

421.

```
<Scan_Parse_Parameter_Type::parse_metadata definition 418> +≡
if (i ≡ 0) {
    lock_cerr_mutex();
    cerr ≡ "[Thread]" ≡ thread_ctr ≡ "]" ≡ "ERROR! In 'Scan_Parse_Parameter_Type::p\
    arse_metadata':" ≡ endl ≡ "'fgets' read 0 lines. Failed to read 'popen' output." ≡
        endl ≡ "Exiting function unsuccessfully with return value 1." ≡ endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    return 1;
} /* if (i ≡ 0) */
status = strtol(line_buffer[i - 1], 0, 10);
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≡ "Shell command result == 'status' == " ≡ status ≡ endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≡ "[Thread]" ≡ thread_ctr ≡ "]" ≡ "ERROR! In 'Scan_Parse_Parameter_Type::p\
    arse_metadata':" ≡ endl ≡ "Shell command failed, returning " ≡
        status ≡ ". XML metadata file check failed." ≡ endl ≡
        "Exiting function unsuccessfully with return value 1." ≡ endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    return 1;
} /* if (i ≡ 0) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≡ "[Thread]" ≡ thread_ctr ≡ "]" ≡ "In 'Scan_Parse_Parameter_Type::parse_me\
            tadata':" ≡ endl ≡ "Shell command succeeded, returning 0." ≡ endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

422.

```

⟨ Scan_Parse_Parameter_Type::parse_metadata definition 418 ⟩ +≡
  if (i % 2 == 0) {
    lock_cerr_mutex();
    cerr << "[Thread]" << thread_ctr << "] " << "ERROR! In 'Scan_Parse_Parameter_Type::p\
      arse_metadata':" << endl << "Shell command returned even number of lines." <<
      "XML metadata file check failed." << endl <<
      "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    return 1;
  } /* if (i % 2 == 0) */
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "[Thread]" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::parse_me\
        tadata':" << endl << "Shell command returned odd number of lines." <<
        "This is correct." << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

423.

```

⟨ Scan_Parse_Parameter_Type ::parse_metadata definition 418 ⟩ +≡
  bool start = true;
  int curr_byte_ctr;
  char curr_tag[512];
  vector<int> byte_ctr_vector;
  for (int j = 0; j < i - 1; ++j) {
    memset(curr_tag, 0, 512);
    status = sscanf(line_buffer[j], "%d:%s", &curr_byte_ctr, curr_tag);
  #if DEBUG_COMPILE
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "status" << endl << "curr_byte_ctr" << endl << curr_byte_ctr << endl <<
        "curr_tag" << endl << curr_tag << endl;
      unlock_cerr_mutex();
    } /* if (DEBUG) */
  #endif /* DEBUG_COMPILE */
    if ((strncmp(curr_tag, "<metadata", strlen("<metadata")) == 0 & start == false) ∨ (strncmp(curr_tag,
      "</metadata", strlen("</metadata")) == 0 & start == true)) {
      lock_cerr_mutex();
      cerr << "[Thread" << thread_ctr << "] " << "ERROR! In 'Scan_Parse_Parameter_Type::p\"
        arse_metadata': " << endl << "XML metadata file not correct: " << endl <<
        "Tag mismatch: " << "start" << endl << start << ", " << "'curr_tag'" << endl << curr_tag << ". " <<
        endl << "Exiting function unsuccessfully with return value 1." << endl;
      unlock_cerr_mutex();
      ++errors_occurred;
    }
  } /* if */
  #if DEBUG_COMPILE
    else
      if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Tags match: " << "start" << endl << start << ", " << "'curr_tag'" << endl << curr_tag <<
          ". " << endl;
        unlock_cerr_mutex();
      } /* else if (DEBUG) */
  #endif /* DEBUG_COMPILE */
    byte_ctr_vector.push_back(curr_byte_ctr);
    start = !start;
  } /* for */

```

424.

```

⟨ Scan_Parse_Parameter_Type ::parse_metadata definition 418 ⟩ +≡
ifstream in_strm;
in_strm.open(response.temporary_filename.c_str());
if (!in_strm) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] " << "ERROR! In 'Scan_Parse_Parameter_Type::p\
arse_metadata':" << endl << "'in_strm'==false" << endl <<
    "Failed to open temporary file" << response.temporary_filename << " " <<
    "for reading." << endl << "Exiting function unsuccessfully with return value 1." <<
    endl;
unlock_cerr_mutex();
++errors_occurred;
return 1;
} /* if (!in_strm) */
if (!in_strm.is_open()) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] " << "ERROR! In 'Scan_Parse_Parameter_Type::p\
arse_metadata':" << endl << "'in_strm.is_open()'==false" << endl <<
    "Failed to open temporary file" << response.temporary_filename << " " <<
    "for reading." << endl << "Exiting function unsuccessfully with return value 1." <<
    endl;
unlock_cerr_mutex();
++errors_occurred;
return 1;
} /* if */
#endif /* DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "Opened temporary file" << response.temporary_filename << " " <<
        "for reading successfully." << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

425.

```

⟨ Scan_Parse_Parameter_Type ::parse_metadata definition 418 ⟩ +≡
int start_val;
int end_val;
unsigned int buffer_size = 1048576;      /* 220 */
char buffer[buffer_size];
Dublin_Core_Metadata_Type dcm; for (vector<int>::const_iterator iter = byte_ctr_vector.begin();  

    iter ≠ byte_ctr_vector.end(); ++iter) { memset(buffer, 0, buffer_size);  

in_strm.seekg(*iter);
if (!in_strm.good()) {
    cerr << "[Thread]" << thread_ctr << "] " << "ERROR! In 'Scan_Parse_Parameter_Type::p\  

    arse_metadata':" << endl << "'in_strm.good()' == 'false'" << endl <<  

    "Failed to set position for reading 'ifstream_in_strm'." << endl <<  

    "Exiting function unsuccessfully with return value 1." << endl;
unlock_cerr_mutex();
++errors_occurred;
in_strm.close();
return 1;
} /* if */
#endif DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread]" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::parse_me\  

    tadata':" << endl << "'in_strm.good()' == 'true'" << endl <<  

    "Set position for reading 'ifstream_in_strm' successfully." << endl;
unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
start_val = *iter++;
end_val = *iter;
curr_byte_ctr = end_val - start_val + 11; /* 11 == strlen("</metadata>") */
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "start_val==" << start_val << endl << "end_val==" << end_val << endl <<  

    "curr_byte_ctr==" << curr_byte_ctr << endl;
unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
if (curr_byte_ctr ≥ buffer_size) {
    cerr << "[Thread]" << thread_ctr << "] " << "ERROR! In 'Scan_Parse_Parameter_Type::p\  

    arse_metadata':" << endl << "'curr_byte_ctr' >= 'buffer_size':" << endl <<  

    "'curr_byte_ctr' == " << curr_byte_ctr << endl << "'buffer_size' == " << buffer_size <<  

    endl << "XML data too long. Can't read from 'ifstream_in_strm'." << endl <<  

    "Exiting function unsuccessfully with return value 1." << endl;
unlock_cerr_mutex();
++errors_occurred;
in_strm.close();
return 1;
} /* if */
in_strm.read(buffer, curr_byte_ctr);

```

```

if ( $\neg$ in_strm.good()) {
    cerr  $\ll$  "[Thread]"  $\ll$  thread_ctr  $\ll$  "]"  $\ll$  "ERROR! In 'Scan_Parse_Parameter_Type::parse_metadata':"
     $\ll$  endl  $\ll$  "'in_strm.good() == false'"  $\ll$ 
    endl  $\ll$  "Failed to read from 'ifstream_in_strm'."  $\ll$  endl  $\ll$ 
    "Exiting function unsuccessfully with return value 1."  $\ll$  endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    in_strm.close();
    return 1;
} /* if */
else if (in_strm.gcount()  $\neq$  curr_byte_ctr) {
    cerr  $\ll$  "[Thread]"  $\ll$  thread_ctr  $\ll$  "]"  $\ll$  "ERROR! In 'Scan_Parse_Parameter_Type::parse_metadata':"
     $\ll$  "'in_strm.gcount() != curr_byte_ctr':"
     $\ll$  endl  $\ll$  "'in_strm.gcount()'"  $\ll$  in_strm.gcount()  $\ll$  endl  $\ll$  "'curr_byte_ctr'"  $\ll$ 
    endl  $\ll$  "Failed to read from 'ifstream_in_strm'."  $\ll$  endl  $\ll$ 
    "Exiting function unsuccessfully with return value 1."  $\ll$  endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    in_strm.close();
    return 1;
} /* else if */
#if DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr  $\ll$  "in_strm.gcount() == "  $\ll$  in_strm.gcount()  $\ll$  endl
#if 0
         $\ll$  "'buffer'"  $\ll$  endl  $\ll$  buffer
#endif
         $\ll$  endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

426.

```

⟨ Scan_Parse_Parameter_Type::parse_metadata definition 418 ⟩ +≡
    status = dcm.parse(buffer);
    if (status ≠ 0) {
        cerr << "[Thread]" << thread_ctr << "] " << "ERROR! In 'Scan_Parse_Parameter_Type::p\
        arse_metadata':" << endl << "'Dublin_Core_Metadata_Type::parse' failed, returning " <<
        status << "." << endl << "Failed to parse XML data." << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        ++errors_occurred;
        in_strm.close();
        return 1;
    } /* if (status ≠ 0) */
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "[Thread]" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::parse_me\
                tadata':" << endl << "'Dublin_Core_Metadata_Type::parse' succeeded, returning 0." <<
                endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    dcm.user_id = user_id;
    dcm.irods_object_path = response.local_filename;
    dc_metadata_vector.push_back(dcm);
    dcm.clear(); } /* for */
    in_strm.close();
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << 'dc_metadata_vector.size()' == " << dc_metadata_vector.size() << endl;
        for (vector<Dublin_Core_Metadata_Type>::const_iterator iter = dc_metadata_vector.begin();
            iter ≠ dc_metadata_vector.end(); ++iter) {
            iter->show();
        }
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

427.

```

⟨ Scan_Parse_Parameter_Type::parse_metadata definition 418 ⟩ +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread]" << thread_ctr << "] Exiting 'Scan_Parse_Parameter_Type::p\
            arse_metadata'" << "successfully with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; } /* End of Scan_Parse_Parameter_Type::parse_metadata definition */

```

428. Fetch handle from database. [LDF 2013.02.07.]

Log

[LDF 2013.02.07.] Added this function.

[LDF 2013.02.13.] Moved the definition of this function from `scprpmtp.web` to this file (`spptfnc1.web`).

[LDF 2013.02.28.] Added argument **Handle_Type** &`handle`. It replaces the old **Handle_Value_Type** & argument.

429.

```
⟨ Scan_Parse_Parameter_Type ::fetch_handle_from_database definition 429 ⟩ ≡
int Scan_Parse_Parameter_Type ::fetch_handle_from_database(unsigned long int
    handle_id, Handle_Type &handle, string type){ int status;
    bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Entering 'Scan_Parse_Parameter_Type::fetch_handle_from_database'" <<
            "(with scalar arguments)." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
See also sections 430, 431, 433, 434, 435, 436, 437, 438, and 439.
This code is used in section 544.
```

430.

```

⟨ Scan_Parse_Parameter_Type ::fetch_handle_from_database definition 429 ⟩ +≡
vector<unsigned long int> handle_id_vector;
vector<Handle_Type> handle_vector;
handle_id_vector.push_back(handle_id);
status = fetch_handles_from_database(handle_id_vector, handle_vector, type);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] ERROR!" << " In 'Scan_Parse_Parameter_Type::fetch_handle_from_database'" << "(with scalar arguments):" << endl << "'Scan_Parse_Parameter_Type::fetch_handles_from_database'" << "(with vector arguments)" << " failed," << " returning" << status << "." << endl << "Exiting function unsuccessfully with return value" << status << "." << endl;
    unlock_cerr_mutex();
}
/* if (status ≠ 0) */
#endif /* DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::fetch_handle_from_database'" << "(with scalar arguments):" << endl << "'Scan_Parse_Parameter_Type::fetch_handles_from_database'" << "(with vector arguments)" << " succeeded," << " returning 0." << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
handle = handle_vector.front();

```

431.

```

⟨ Scan_Parse_Parameter_Type ::fetch_handle_from_database definition 429 ⟩ +≡
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "Exiting 'Scan_Parse_Parameter_Type::fetch_handle_from_database'" << "(with scalar arguments)" << " successfully with return value 0." << endl;
    unlock_cerr_mutex();
}
/* if (DEBUG) */
#endif /* DEBUG_COMPILE */
return 0; } /* End of Scan_Parse_Parameter_Type ::fetch_handle_from_database (with scalar arguments) definition */

```

432. Fetch handle from database (**string** argument). [LDF 2013.06.16.]

Log

[LDF 2013.06.16.] Added this function.

433.

```
<Scan_Parse_Parameter_Type::fetch_handle_from_database definition 429> +≡
int Scan_Parse_Parameter_Type::fetch_handle_from_database(string handle_str, Handle_Type
    &handle, string type){ int status;
    bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    stringstream sql strm;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Entering 'Scan_Parse_Parameter_Type::fetch_handle_from_database' "
            "(with 'string' argument). " << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

434.

```

⟨ Scan_Parse_Parameter_Type ::fetch_handle_from_database definition 429 ⟩ +≡
  sql_strm << "select distinct handle_id from handlesystem_standalone.handles"
  "where handle=''" << handle_str << "' limit 1";
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "]"
    << "In 'Scan_Parse_Parameter_Type::fetch_ha"
    "ndle_from_database'" << "(with 'string' arguments):" << endl <<
    "'sql_strm.str()'" << sql_strm.str() << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
MySQL_RES * result = 0;
unsigned int row_ctr;
unsigned int field_ctr;
MySQL_ROW curr_row;
status = submit_mysql_query(sql_strm.str().c_str(), result, &row_ctr, &field_ctr);
if (status != 0) {
  lock_cerr_mutex();
  cerr << "[Thread" << thread_ctr << "]"
  << "ERROR! In 'Scan_Pa"
  "rse_Parameter_Type::fetch_handle_from_database':"
  << endl <<
  "'Scan_Parse_Parameter_Type::submit_mysql_query' failed, returning"
  << status <<
  "."
  << endl << mysql_error(mysql_ptr) << endl <<
  "Failed to fetch handle from database."
  << endl <<
  "Exiting function unsuccessfully with return value 1."
  << endl;
  unlock_cerr_mutex();
  if (result) {
    mysql_free_result(result);
  }
  ++errors_occurred;
  return 1;
} /* if (status != 0) */
#ifndef DEBUG_COMPILE
  else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "]"
    << "In 'Scan_Parse_Parameter_Type::fetch_handle_from_database':"
    << endl <<
    "'Scan_Parse_Parameter_Type::submit_mysql_query' succeeded."
    << endl <<
    "'row_ctr'" << row_ctr << endl <<
    "'field_ctr'" << field_ctr << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

435.

```
< Scan_Parse_Parameter_Type::fetch_handle_from_database definition 429 > +≡
  if (row_ctr ≡ 0) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread]" << thread_ctr << "]"
    << "In 'Scan_Parse_Parameter_Type::fetch_handle_from_database':"
    << endl <<
    "'Scan_Parse_Parameter_Type::submit_mysql_query' returned 0 rows."
    << endl << "No handle ID fetched."
    << endl <<
    "Exiting function successfully with return value 2."
    << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  mysql_free_result(result);
  ++errors_occurred;
  return 2;
} /* if (row_ctr ≡ 0) */
```

436.

```
< Scan_Parse_Parameter_Type::fetch_handle_from_database definition 429 > +≡
#ifndef DEBUG_COMPILE
else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread]" << thread_ctr << "]"
    << "In 'Scan_Parse_Parameter_Type::fetch_handle_from_database':"
    << endl <<
    "'Scan_Parse_Parameter_Type::submit_mysql_query' returned"
    << row_ctr <<
    " rows."
    << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

437.

```

⟨ Scan_Parse_Parameter_Type ::fetch_handle_from_database definition 429 ⟩ +≡
if ((curr_row = mysql_fetch_row(result)) ≡ 0) {
    lock_cerr_mutex();
    cerr << "[Thread_]" << thread_ctr << "]_ " << "ERROR!_In_`Scan_Pa\
rse_Parameter_Type::fetch_handle_from_database':_ " << endl <<
    "'mysql_fetch_row'_failed:" << endl << mysql_error(mysql_ptr) << endl <<
    "Exiting_function_unsuccessfully_with_return_value_1." << endl;
unlock_cerr_mutex();
mysql_free_result(result);
++errors_occurred;
return 1;
} /* if (curr_row = mysql_fetch_row(result) ≡ 0) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread_]" << thread_ctr << "]_ " << "In_`Scan_Parse_Parameter_Type::fetch_ha\
ndle_from_database':_ " << endl << "'mysql_fetch_row'_succeeded." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
errno = 0;
unsigned long int handle_id = strtoul(curr_row[0], 0, 10);
if (handle_id ≡ ULONG_MAX) {
    lock_cerr_mutex();
    cerr << "[Thread_]" << thread_ctr << "]_ " << "ERROR!_In_`Scan_Pa\
rse_Parameter_Type::fetch_handle_from_database':_ " << endl <<
    "'strtoul'_failed,_returning_ULONG_MAX:' << endl << strerror(errno) <<
    endl << "Exiting_function_unsuccessfully_with_return_value_1." << endl;
unlock_cerr_mutex();
mysql_free_result(result);
++errors_occurred;
return 1;
} /* if (handle_id ≡ ULONG_MAX) */
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread_]" << thread_ctr << "]_ " << "In_`Scan_Parse_Parameter_Type::fetch_ha\
ndle_from_database':_ " << endl << "'strtoul'_succeeded." << endl << "'handle_id'_==_ " <<
        handle_id << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

438.

```
<Scan_Parse_Parameter_Type::fetch_handle_from_database definition 429> +≡
mysql_free_result(result);
result = 0;
status = fetch_handle_from_database(handle_id, handle, type);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] ERROR!" << " In 'Scan_Parse_Para-
meter_Type::fetch_handle_from_database'" << "(with 'string' argument):" <<
endl << "'Scan_Parse_Parameter_Type::fetch_handle_from_database'" <<
"(with scalar arguments) failed," << "returning" << status << "." << endl <<
"Exiting function unsuccessfully with return value" << status << "." << endl;
unlock_cerr_mutex();
return 1;
} /* if (status ≠ 0) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::fetch_ha-
ndle_from_database'" << "(with 'string' arguments):" << endl <<
"'Scan_Parse_Parameter_Type::fetch_handles_from_database'" <<
"(with vector arguments) succeeded," << "returning 0." << endl;
unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

439.

```
<Scan_Parse_Parameter_Type::fetch_handle_from_database definition 429> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Exiting 'Scan_Parse_Parameter_Type::fetch_handle_from_database'" <<
            "(with 'string' arguments) successfully with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; } /* End of Scan_Parse_Parameter_Type::fetch_handle_from_database (with 'string'
argument) definition */
```

440. Fetch handles from database. [LDF 2013.02.07.]

Log

- [LDF 2013.02.07.] Added this function.
 - [LDF 2013.02.13.] Moved the definition of this function from `scprpmtp.web` to this file (`spptfnc1.web`).
 - [LDF 2013.02.28.] Added argument `vector<Handle_Type> &handle_vector`. It replaces the old `vector<Handle_Value>` & argument.
 - [LDF 2013.08.12.] Removed code. Now calling `Handle_Type::fetch_handles_from_database`, which contains the code that was formerly in this function.
-

441.

```

⟨ Scan_Parse_Parameter_Type ::fetch_handles_from_database definition 441 ⟩ ≡
int Scan_Parse_Parameter_Type ::fetch_handles_from_database(vector<unsigned long int>
    &handle_id_vector, vector<Handle_Type> &handle_vector, string type){ bool DEBUG = false;
/* true */
set_debug_level(DEBUG, 0, 0);
int status;
stringstream temp_strm;
temp_strm << "[Thread" << thread_ctr << "]"
string thread_str = temp_strm.str();
temp_strm.str("");
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "]"
        << "Entering 'Scan_Parse_Parameter_Type::fe\
tch_handles_from_database'" << "(with vector arguments)." << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
status = Handle_Type ::fetch_handles_from_database(mysql_ptr, handle_id_vector, handle_vector,
type, thread_str);
if (status ≠ 0) {
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "]"
        << "ERROR! In 'Scan_Parse_Parameter_Type::f\
etch_handles_from_database'" << "(with vector arguments):"
        << endl <<
        "Handle_Type::fetch_handles_from_database::fetch_handles_from_database'" <<
        "failed, returning" << status << "."
        << endl <<
        "Exiting function unsuccessfully with return value" << status << "."
        << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
++errors_occurred;
return status;
} /* if (status ≠ 0) */

```

See also sections 442 and 443.

This code is used in section 544.

442.

```
<Scan_Parse_Parameter_Type::fetch_handles_from_database definition 441> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::fetch_handles_from_database' "
            << "(with vector arguments):" << endl <<
            "'Handle_Type::fetch_handles_from_database::fetch_handles_from_database'" <<
            "succeeded, returning 0." << endl << "handle_vector.size() == " << handle_vector.size() <<
            endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

443.

```
<Scan_Parse_Parameter_Type::fetch_handles_from_database definition 441> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] " <<
            "Exiting 'Scan_Parse_Parameter_Type::fetch_handles_from_database' "
            << "(with vector arguments) successfully with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; } /* End of Scan_Parse_Parameter_Type::fetch_handles_from_database (with vector
arguments) definition */
```

444. Store Dublin Core metadata (*store_dc_metadata*). [LDF 2013.03.01.]

Log

[LDF 2013.03.01.] Added this function. It's declared in `scprpmtp.web`.

[LDF 2013.03.07.] Changed arguments: Now passing **Handle_Type** references instead of a string representing the handle and **unsigned long int** values representing the handle ID.

```

< Scan_Parse_Parameter_Type::store_dc_metadata definition 444 > ≡
int Scan_Parse_Parameter_Type::store_dc_metadata(const Response_Type &response,
                                                Handle_Type &irods_object_handle, Handle_Type &dc_metadata_handle, bool force, string
                                                &dc_metadata_irods_object_path){ bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    MYSQL_RES * result = 0;
    unsigned int row_ctr;
    unsigned int field_ctr;
    stringstream sql_strm;
    stringstream temp_strm;
    FILE *fp;
    char buffer[1024];
    memset(buffer, 0, 1024);
    bool irods_object_exists = false;
    vector<Handle_Value_Triple> hvt_vector;
    Handle_Value_Triple hvt;
    Response_Type new_response;
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] " <<
            "Entering 'Scan_Parse_Parameter_Type::store_dc_metadata'." << endl;
        response.show("response:");
        cerr << "'force'" << force << endl << "irods_object_handle.handle" <<
            irods_object_handle.handle << endl << "irods_object_handle.handle_id" <<
            irods_object_handle.handle_id << endl << "dc_metadata_handle.handle" <<
            dc_metadata_handle.handle << endl << "dc_metadata_handle.handle_id" <<
            dc_metadata_handle.handle_id << endl << "irods_current_dir" <<
            irods_current_dir << endl;
        cerr << "basename(response.remote_filename.c_str())" <<
            basename(response.remote_filename.c_str()) << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

See also sections 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, and 477.

This code is used in section 544.

445. !! TODO: Make it possible to use the same directory as the iRODS object to which the metadata refers. [LDF 2013.03.08.]

```

⟨ Scan_Parse_Parameter_Type::store_dc_metadata definition 444 ⟩ +≡
dc_metadata_irods_object_path = basename(response.remote_filename.c_str());
if (dc_metadata_irods_object_path.empty()) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] ERROR! " << "In 'Scan_Parse_Para\
meter_Type::store_dc_metadata': " << endl << "'basename' returned empty string." <<
endl << "Filename for Dublin Core metadata is empty." << endl <<
"Exiting function unsuccessfully with return value 1." << endl;
unlock_cerr_mutex();
++errors_occurred;
return 1;
} /* if (dc_metadata_irods_object_path.empty()) */
dc_metadata_irods_object_path.insert(0, "/");
dc_metadata_irods_object_path.insert(0, irods_current_dir);
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::store_dc\
_metadata': " << endl << "'dc_metadata_irods_object_path'" == " \
dc_metadata_irods_object_path << '.' " << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

446. Check whether iRODS object already exists. [LDF 2013.03.06.]

```
( Scan_Parse_Parameter_Type::store_dc_metadata definition 444 ) +≡
  Irods_Object_Type curr_irods_object;
  curr_irods_object.set(user_id, dc_metadata_irods_object_path);
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    curr_irods_object.show("curr_irods_object:");
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  status = curr_irods_object.get_from_database(mysql_ptr, true);
  if (status == -1) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] ERROR!" <<
      "In 'Scan_Parse_Parameter_Type::store_dc_metadata':" << endl <<
      "'Irods_Object_Type::get_from_database' failed, returning -1." <<
      endl << "Failed to check for existence of iRODS object" <<
      endl << dc_metadata_irods_object_path << "." << endl <<
      "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    return 1;
  } /* if (status == -1) */
#endif /* DEBUG_COMPILE */
else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] " <<
      "In 'Scan_Parse_Parameter_Type::store_dc_metadata':" << endl <<
      "'Irods_Object_Type::get_from_database' succeeded, returning" << status << "." <<
      endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

447. iRODS object found in database and *force* == *false*. iRODS object not overwritten. Exit. [LDF 2013.03.07.] ■

```
( Scan_Parse_Parameter_Type::store_dc_metadata definition 444 ) +≡
if (status > 0 & !force) {
  lock_cerr_mutex();
  cerr << "[Thread" << thread_ctr << "] WARNING!" <<
    "In 'Scan_Parse_Parameter_Type::store_dc_metadata':" << endl <<
    "'Irods_Object_Type::get_from_database' returned" << status << "(>0)." <<
    endl << "iRODS object" << dc_metadata_irods_object_path << "' already exists" <<
    "and 'force' == 'false'." << endl << "Not overwriting iRODS object." << endl <<
    "Exiting function successfully with return value 2." << endl;
  unlock_cerr_mutex();
  ++errors_occurred;
  return 2;
} /* if (status > 0 & !force) */
```

448. iRODS object exists in database, but *force* \equiv *true*. Will overwrite iRODS object. [LDF 2013.03.07.]

```
<Scan_Parse_Parameter_Type::store_dc_metadata definition 444> +≡
else
  if (status > 0  $\wedge$  force) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread]" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::store_dc\
      _metadata':" << endl << "'Irods_Object_Type::get_from_database' returned" <<
      status << "(>0)." << endl << "iRODS_object" << dc_metadata_irods_object_path <<
      "'already_exists'" << "and 'force' is 'true'." << endl <<
      "Will call \"input\" with the '-f' option." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  irods_object_exists = true;
} /* else if (status > 0  $\wedge$  force) */
```

449. iRODS object not found in database. Will check iRODS system using *ils* to see if an iRODS exists. Normally, this shouldn't be the case, but an iRODS object may have been created directly using *input* or one of the Jargon APIs. [LDF 2013.03.07.]

```
<Scan_Parse_Parameter_Type::store_dc_metadata definition 444> +≡
#ifndef DEBUG_COMPILE
else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread]" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::store_dc\
      _metadata':" << endl << "'Irods_Object_Type::get_from_database' returned 0." <<
      endl << "iRODS_object" << dc_metadata_irods_object_path <<
      "'not found in database.'" << endl << "Will check using 'ils'." << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

450. iRODS object doesn't exist in the database. Write it to database. This needs to be done even if an iRODS object exists in the iRODS system. [LDF 2013.03.07.]

```
<Scan_Parse_Parameter_Type::store_dc_metadata definition 444> +≡
if (!irods_object_exists) { status = curr_irods_object.write_to_database(mysql_ptr);
if (status != 0) {
  lock_cerr_mutex();
  cerr << "[Thread] " << thread_ctr << "] ERROR! " <<
    "In 'Scan_Parse_Parameter_Type::store_dc_metadata':" << endl <<
    "'Irods_Object_Type::write_to_database' failed, returning -1." << endl <<
    "Failed to write iRODS object" << " " << dc_metadata_irods_object_path <<
    "'to database.'" << endl << "Exiting function unsuccessfully with return value 1." <<
    endl;
  unlock_cerr_mutex();
  ++errors_occurred;
  return 1;
} /* if (status != 0) */
```

451.

```
<Scan_Parse_Parameter_Type::store_dc_metadata definition 444> +≡
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "[Thread]" << thread_ctr << "]"
                << "In 'Scan_Parse_Parameter_Type::store_dc_metadata':"
                << endl <<
                "'Irods_Object_Type::write_to_database' succeeded, returning 0."
                << endl;
            curr_irods_object.show("curr_irods_object:");
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

452. Call ils.

```
<Scan_Parse_Parameter_Type::store_dc_metadata definition 444> +≡
temp_strm.str();
temp_strm << "r="env_irodsEnvFile=" << irods_env_filename << "ils"
    << dc_metadata_irods_object_path << ">&1"; echo-e"\$?\n$r\"";
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread]" << thread_ctr << "]"
            << "In 'Scan_Parse_Parameter_Type::store_dc\
                _metadata':"
            << endl << "'temp_strm.str()' =="
            << temp_strm.str() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
errno = 0;
fp = popen(temp_strm.str().c_str(), "r");
if (fp == 0) {
    lock_cerr_mutex();
    cerr << "[Thread]" << thread_ctr << "]"
        << "ERROR! In 'Scan_Parse_Parameter_Type::store_dc_metadata':"
        << endl <<
        "'popen' failed, returning NULL:"
        << endl << strerror(errno) << endl <<
        "Failed to check for existence of iRODS object using 'ils'."
        << endl <<
        "Exiting function unsuccessfully with return value 1."
        << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    return 1;
} /* if (fp == 0) */
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "[Thread]" << thread_ctr << "]"
                << "In 'Scan_Parse_Parameter_Type::store_dc\
                    _metadata':"
                << endl << "'popen' succeeded."
                << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

453.

```
<Scan_Parse_Parameter_Type::store_dc_metadata definition 444> +≡
    int temp_val;
    errno = 0;
    status = fscanf(fp, "%d", &temp_val);
    if (status ≠ 1) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "]"
            << "ERROR! In 'Scan_Parse_Parameter_Type::store_dc_metadata':"
            << endl <<
            "'fscanf' failed, returning" << status << ":" << endl <<
            strerror(errno) << endl <<
            "Failed to check for existence of iRODS object using 'ils'."
            << endl <<
            "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        ++errors_occurred;
        pclose(fp);
        return 1;
    } /* if (status ≠ 1) */
```

454.

```
<Scan_Parse_Parameter_Type::store_dc_metadata definition 444> +≡
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "[Thread" << thread_ctr << "]"
                << "In 'Scan_Parse_Parameter_Type::store_dc_metadata':"
                << endl <<
                "'fscanf' succeeded, returning 1." << endl <<
                "'temp_val' == " << temp_val << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

455.

```
<Scan_Parse_Parameter_Type::store_dc_metadata definition 444> +≡
  if (temp_val == 0) {
    lock_cerr_mutex();
    cerr << "[Thread]" << thread_ctr << "] " << "WARNING! In 'Scan_\
      Parse_Parameter_Type::store_dc_metadata':"
    endl <<
    "ils' returned 0: "
    << "An iRODS object already exists for "
    " " << dc_metadata_irods_object_path << ", and "
    << "'force' == 'false'." <<
    endl << "Not storing Dublin Core metadata in an iRODS object." << endl <<
    "Exiting function successfully with return value 2." << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    ++warnings_occurred;
    pclose(fp);
    return 2;
  } /* if (temp_val == 0) */
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "temp_val== " << temp_val << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

456. Read the output of the `ils` command. This isn't really necessary, since it's not evaluated. However, it may be useful for some purpose in the future. [LDF 2013.03.07.]

```

⟨ Scan_Parse_Parameter_Type::store_dc_metadata definition 444 ⟩ +≡
  memset(buffer, 0, 1024);
  status = fread(buffer, 1, 1024, fp);
  if (status ≡ 0) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] " << "ERROR! In 'Scan_Parse_Parameter_Type::"
    store_dc_metadata':" << endl << "'fread' failed, returning 0." <<
    endl << "Failed to retrieve output from call to 'ils'." << endl <<
    "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    pclose(fp);
    return 1;
  } /* if (status ≡ 0) */
else if (status ≡ 1024) {
  lock_cerr_mutex();
  cerr << "[Thread" << thread_ctr << "] " << "ERROR! In 'Scan_Parse_Parameter_Type::"
  store_dc_metadata':" << endl << "'fread' returned 1024: Number of chara\
cters returned by call to 'ils' " << "exceeds maximum allowed (1023)." << endl <<
  "Exiting function unsuccessfully with return value 1." << endl;
  unlock_cerr_mutex();
  ++errors_occurred;
  pclose(fp);
  return 1;
} /* else if (status ≡ 1024) */
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "status==" << status << endl << "buffer==" << buffer << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  pclose(fp);
  fp = 0;
  if (temp_val ≡ 0) irods_object_exists = true;
} /* if (!irods_object_exists) */

```

457. *force* \equiv *true*. Overwrite iRODS object, if it already exists. [LDF 2013.03.07.]

```
<Scan_Parse_Parameter_Type::store_dc_metadata definition 444> +≡
  if (force) {
#if DEBUG_COMPILE
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "[Thread]" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::store_dc\
      _metadata':" << endl << "'force'" == "true" and " " << endl;
    if (irods_object_exists) {
      cerr << "'irods_object_exists'" == "true" . " << endl << "iRODS_object" <<
        dc_metadata_irods_object_path << "'alreadyexists.' " << "Will overwrite." << endl;
    } /* if (irods_object_exists) */
    else {
      cerr << "'irods_object_exists'" == "false" . " << endl << "iRODS_object" <<
        dc_metadata_irods_object_path << "' doesn't already exist.' " <<
        "Will create." << endl;
    }
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (force) */
```

458.

```
<Scan_Parse_Parameter_Type::store_dc_metadata definition 444> +≡
  else /* ~force */
  {
#if DEBUG_COMPILE
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "[Thread]" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::store_dc\
      _metadata':" << endl << "'force'" == "false" . " << endl;
      unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

459.

```
<Scan_Parse_Parameter_Type::store_dc_metadata definition 444> +≡
```

460. *force* ≡ *false* and iRODS

```
< Scan_Parse_Parameter_Type::store_dc_metadata definition 444 > +≡
if (irods_object_exists) {
    lock_cerr_mutex();
    cerr << "[Thread_" << thread_ctr << "] " << "WARNING! In 'Scan_\n"
        Parse_Parameter_Type::store_dc_metadata':" << endl <<
        "'force'" <== 'false' and 'irods_object_exists' == 'true'." << endl <<
        "iRODS_object" << dc_metadata_irods_object_path << "','" already_exists." <<
        "Not overwriting." << "Exiting function successfully with return value 2." << endl;
    unlock_cerr_mutex();
    ++warnings_occurred;
    return 2;
} /* if (irods_object_exists) */
#endif /* DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread_" << thread_ctr << "] " <<
        "In 'Scan_Parse_Parameter_Type::store_dc_metadata':" << endl <<
        "'force'" <== 'false' and 'irods_object_exists' == 'false'." <<
        endl << "iRODS_object" << dc_metadata_irods_object_path << "','" <<
        "doesn't already exist. Will create." << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* else (¬force) */
```

461. Generate PID. [LDF 2013.03.07.]

```
< Scan_Parse_Parameter_Type::store_dc_metadata definition 444 > +≡
string curr_pid;
vector<Handle_Type> handle_vector;
status = generate_pids(mysql_ptr, default_handle_prefix, curr_pid, 0,
    /* PID vector pointer (vector<string>) * */
    1,      /* Number of PIDS */
    0,      /* Handle ID vector pointer */
    0,      /* Handle value ID vector pointer */
    true,   /* standalone handle server */
    "",     /* Institute string */
    "",     /* Suffix string */
    &handle_vector, "", /* fifo.pathname */
    user_id, username);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << "[Thread_" << thread_ctr << "] ERROR! " << "In 'Scan_Parse_Parameter_Type::store_dc_metadata':" << endl << "'generate_pids' failed, returning" <<
        status << "." << endl << "Failed to generate PID." <<
        "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    return 1;
} /* if (status ≠ 0) */
```

462.

```
< Scan_Parse_Parameter_Type :: store_dc_metadata definition 444 > +≡
else /* status ≡ 0 */
{
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread_"
            << thread_ctr << "] "
            << "In_`Scan_Parse_Parameter_Type::store_dc\
_metadata':"
            << endl << `generate_pids'_succeeded,_returning_0:"
            << endl <<
            `curr_pid' == curr_pid << endl;
        if (handle_vector.size() ≠ 0) {
            cerr << "Showing_`handle_vector.back()' :"
            << endl;
            handle_vector.back().show("handle_vector.back()");
        } /* if */
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
new_response.clear();
new_response.type = Response_Type::COMMAND_ONLY_TYPE;
temp_strm.str("");
temp_strm << "STORE_METADATA_RESPONSE_0\""
            << dc_metadata_irods_object_path <<
            "\\""
            << "\""
            << response.local_filename << "\\""
            << "\""
            << "Generated_handle"
            << handle_vector.back().handle << "_for_Dublin_Core_Metadata_iRODS_Object"
            << '\"'
            << dc_metadata_irods_object_path << ".\"";
new_response.command = temp_strm.str();
temp_strm.str("");
response_deque.push_back(new_response);
} /* else (status ≡ 0) */
curr_irods_object.handle_id_vector.push_back(handle_vector.back().handle_id);
```

463. Call **Handle_Type** :: *add_value* on *handle_vector.back()*. [LDF 2013.03.07.]

```

⟨ Scan_Parse_Parameter_Type :: store_dc_metadata definition 444 ⟩ +≡
  hvt.idx = Handle_Value_Type :: DC_METADATA_IRODS_OBJECT_INDEX;
  hvt.type =
    Handle_Value_Type :: idx_type_map[Handle_Value_Type :: DC_METADATA_IRODS_OBJECT_INDEX];
  hvt.data_str = dc_metadata_irods_object_path;
  hvt_vector.push_back(hvt);
  hvt.idx = Handle_Value_Type :: IRODS_OBJECT_REF_INDEX;
  hvt.type = Handle_Value_Type :: idx_type_map[Handle_Value_Type :: IRODS_OBJECT_REF_INDEX];
  hvt.data_str = response.local.filename;
  hvt_vector.push_back(hvt);
  hvt.idx = Handle_Value_Type :: IRODS_OBJECT_PID_INDEX;
  hvt.type = Handle_Value_Type :: idx_type_map[Handle_Value_Type :: IRODS_OBJECT_PID_INDEX];
  hvt.data_str = irods_object_handle.handle;
  hvt_vector.push_back(hvt);
  hvt.idx = Handle_Value_Type :: DC_METADATA_PID_INDEX;
  hvt.type = Handle_Value_Type :: idx_type_map[Handle_Value_Type :: DC_METADATA_PID_INDEX];
  hvt.data_str = dc_metadata_handle.handle;
  hvt_vector.push_back(hvt);
#endif DEBUG_COMPILE
if (DEBUG) {
  lock_cerr_mutex();
  cerr << "[Thread" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::store_dc\
  _metadata':" << endl << "hvt_vector:" << endl;
  for (vector<Handle_Value_Triple>::iterator iter = hvt_vector.begin(); iter != hvt_vector.end();
      ++iter) {
    cerr << "iter->idx==" << iter->idx << endl << "iter->type==" << iter->type << endl <<
      "iter->data_str==" << iter->data_str << endl;
  }
  unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
status = handle_vector.back().add_values(mysql_ptr, hvt_vector, user_id);
if (status ≠ 0) {
  lock_cerr_mutex();
  cerr << "[Thread" << thread_ctr << "] ERROR!" <<
    "In 'Scan_Parse_Parameter_Type::store_dc_metadata':" << endl <<
    "'Handle_Type::add_value' failed, returning" << status << "." <<
    endl << "Failed to add value to PID" << curr_pid << '.' << endl <<
    "Exiting function unsuccessfully with return value 1." << endl;
  unlock_cerr_mutex();
  ++errors_occurred;
  return 1;
} /* if (status ≠ 0) */
#endif DEBUG_COMPILE
else
if (DEBUG) {
  lock_cerr_mutex();
  cerr << "[Thread" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::store_dc\
  _metadata':" << endl << "'Handle_Type::add_value' succeeded, returning 0." <<
  endl << "Added value to PID" << curr_pid << " successfully." << endl;
  handle_vector.back().show("handle_vector.back() :");
}

```

```

    cerr << "response.local_filename==" << response.local_filename << endl <<
    "irods_object_handle.handle==" << irods_object_handle.handle << endl <<
    "irods_object_handle.handle_id==" << irods_object_handle.handle_id << endl <<
    "dc_metadata_handle.handle==" << dc_metadata_handle.handle << endl <<
    "dc_metadata_handle.handle_id==" << dc_metadata_handle.handle_id << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

464.

```

⟨ Scan_Parse_Parameter_Type::store_dc_metadata definition 444 ⟩ +≡
    hvt_vector.clear();
    hvt.idx = Handle_Value_Type::DC_METADATA_IRODS_OBJECT_REF_INDEX;
    hvt.type = Handle_Value_Type::idx_type_map[Handle_Value_Type::DC_METADATA_IRODS_OBJECT_REF_INDEX];
    hvt.data_str = dc_metadata_irods_object_path;
    hvt_vector.push_back(hvt);
    hvt.idx = Handle_Value_Type::DC_METADATA_IRODS_OBJECT_PID_INDEX;
    hvt.type = Handle_Value_Type::idx_type_map[Handle_Value_Type::DC_METADATA_IRODS_OBJECT_PID_INDEX];
    hvt.data_str = curr_pid;
    hvt_vector.push_back(hvt);

```

465.

```

⟨ Scan_Parse_Parameter_Type::store_dc_metadata definition 444 ⟩ +≡
    status = irods_object_handle.add_values(mysql_ptr, hvt_vector, user_id);
    if (status ≠ 0) {
        lock_cerr_mutex();
        cerr << "[Thread]" << thread_ctr << "] ERROR!" <<
            "In 'Scan_Parse_Parameter_Type::store_dc_metadata':" << endl <<
            "'Handle_Type::add_value' failed, returning" << status << "." << endl <<
            "Failed to add value to PID" << irods_object_handle.handle << '.' << endl <<
            "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        ++errors_occurred;
        return 1;
    } /* if (status ≠ 0) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread]" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::store_dc\
            _metadata':" << endl << "'Handle_Type::add_value' succeeded, returning 0." << endl <<
            "Added value to PID" << irods_object_handle.handle << " successfully." << endl;
        irods_object_handle.show("irods_object_handle:");
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

466.

```

⟨ Scan_Parse_Parameter_Type :: store_dc_metadata definition 444 ⟩ +≡
  status = dc_metadata_handle.add_values(mysql_ptr, hvt_vector, user_id);
  if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << "[Thread]" << thread_ctr << "] ERROR!" <<
      "In 'Scan_Parse_Parameter_Type::store_dc_metadata':" << endl <<
      "'Handle_Type::add_value' failed, returning" << status << "." << endl <<
      "Failed to add value to PID" << dc_metadata_handle.handle << '.' << endl <<
      "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    return 1;
  } /* if (status ≠ 0) */
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "[Thread]" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::store_dc\
        _metadata':" << endl << "'Handle_Type::add_value' succeeded, returning 0." << endl <<
        "Added value to PID" << dc_metadata_handle.handle << "' successfully." << endl;
      dc_metadata_handle.show("dc_metadata_handle:");
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

467. Set AVUs in *curr_irods_object*. [LDF 2013.03.07.]

Log

[LDF 2013.03.07.] Added this section.

```
< Scan_Parse_Parameter_Type :: store_dc_metadata definition 444 > +≡
  time_t t = time(0);
  int temp_val_1 = curr_irods_object.avu_vector.size();
  Irods_AVU_Type curr_avu("TYPE", "DC_METADATA_IRODS_OBJECT", "", t, curr_irods_object.id);
  curr_irods_object.avu_vector.push_back(curr_avu);
  curr_avu.set("PID", curr_pid, "", t, curr_irods_object.id);
  curr_irods_object.avu_vector.push_back(curr_avu);
  curr_avu.set("IRODS_OBJECT_REF", response.local_filename, "", t, curr_irods_object.id);
  curr_irods_object.avu_vector.push_back(curr_avu);
  curr_avu.set("IRODS_OBJECT_PID", irods_object_handle.handle, "", t, curr_irods_object.id);
  curr_irods_object.avu_vector.push_back(curr_avu);
  curr_avu.set("DC_METADATA_PID", dc_metadata_handle.handle, "", t, curr_irods_object.id);
  curr_irods_object.avu_vector.push_back(curr_avu);
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    curr_irods_object.show("curr_irods_object:");
    cerr << "temp_val_1=" << temp_val_1 << endl <<
      curr_irods_object.avu_vector.size() << curr_irods_object.avu_vector.size() << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

468.

```

⟨ Scan_Parse_Parameter_Type :: store_dc_metadata definition 444 ⟩ +≡
  for (int i = temp_val_1; i < curr_irods_object.avu_vector.size(); ++i) {
    status = curr_irods_object.avu_vector[i].write_to_database(mysql_ptr, thread_ctr);
    if (status ≠ 0) {
      lock_cerr_mutex();
      cerr << "[Thread_" << thread_ctr << "] ERROR!" <<
        "In 'Scan_Parse_Parameter_Type::store_dc_metadata':" << endl <<
        "'Irods_AVU_Type::write_to_database' failed, returning " << status << "." <<
        endl << "Failed to write AVU to 'gwiridsif.Irods_AVUs' database table." << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
      unlock_cerr_mutex();
      ++errors_occurred;
      return 1;
    } /* if (status ≠ 0) */
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "[Thread_" << thread_ctr << "]"
        "In 'Scan_Parse_Parameter_Type::store_dc_metadata':" << endl <<
        "'Irods_AVU_Type::write_to_database' succeeded, returning 0." << endl <<
        "Wrote AVU to 'gwiridsif.Irods_AVUs' database table successfully." << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* for */

```

469.

```
<Scan_Parse_Parameter_Type::store_dc_metadata definition 444> +≡
status = curr_irods_object.update(mysql_ptr);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "]ERROR!" <<
        "In 'Scan_Parse_Parameter_Type::store_dc_metadata':" << endl <<
        "'Irods_Object_Type::update' failed, returning" << status <<
        "." << endl << "Failed to update 'curr_irods_object'." <<
        "Exiting function unsuccessfully with return value 1." << endl;
unlock_cerr_mutex();
++errors_occurred;
return 1;
} /* if (status ≠ 0) */
#ifndef DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "]"
        "In 'Scan_Parse_Parameter_Type::store_dc_metadata':" << endl <<
        "'Irods_Object_Type::update' succeeded, returning 0." << endl;
unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

470. Call Irods_Object_Type::put_irods_object on curr_irods_object. [LDF 2013.03.07.]

```
<Scan_Parse_Parameter_Type::store_dc_metadata definition 444> +≡
status = curr_irods_object.put_irods_object(response.temporary_filename, irods_env_filename, force);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "]ERROR!" <<
        "In 'Scan_Parse_Parameter_Type::store_dc_metadata':" << endl <<
        "'Irods_Object_Type::put_irods_object' failed, returning" <<
        status << "." << endl << "Failed to put \"curr_irods_object\"."
        "Exiting function unsuccessfully with return value 1." << endl;
unlock_cerr_mutex();
++errors_occurred;
return 1;
} /* if (status ≠ 0) */
#ifndef DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "]"
        "In 'Scan_Parse_Parameter_Type::store_dc_metadata':" << endl <<
        "'Irods_Object_Type::put_irods_object' succeeded, returning 0." << endl;
unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

471.

```
<Scan_Parse_Parameter_Type::store_dc_metadata definition 444> +≡
  Irods_Object_Type ref_irods_object;
  ref_irods_object.set(user_id, response.local_filename);
  status = ref_irods_object.get_from_database(mysql_ptr, true);
  if (status ≡ -1) {
    lock_cerr_mutex();
    cerr ≪ "[Thread_" ≪ thread_ctr ≪ "]_ERROR!" ≪
      "In_‘Scan_Parse_Parameter_Type::store_dc_metadata’::" ≪ endl ≪
      "'Irods_Object_Type::get_from_database’_failed,_returning_-1." ≪ endl ≪
      "Failed_to_check_for_existence_of_iRODS_object_" ≪ "“" ≪ response.local_filename ≪
      ".” ≪ endl ≪ "Exiting_function_unsuccessfully_with_return_value_1." ≪ endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    return 1;
} /* if (status ≡ -1) */
```

472. iRODS object *response.local_filename* doesn't exist in database. This may occur if Dublin Core metadata is added for a non-existent iRODS object.

!! PLEASE NOTE: This code *does not* check the iRODS system using the *ils* command; If there's no database entry for the iRODS object, one is not created here and no AVU is added.

!! TODO: Set up a response and send it to the client. [LDF 2013.03.08.]

```
<Scan_Parse_Parameter_Type::store_dc_metadata definition 444> +≡
else
  if (status ≡ 0) {
    lock_cerr_mutex();
    cerr ≪ "[Thread_" ≪ thread_ctr ≪ "]_WARNING!" ≪
      "In_‘Scan_Parse_Parameter_Type::store_dc_metadata’::" ≪ endl ≪
      "'Irods_Object_Type::get_from_database’_returned_0:" ≪ endl ≪
      "iRODS_object_“" ≪ response.local_filename ≪ ",_not_present_in_database." ≪ endl ≪
      "Not_adding_AVU._Continuing." ≪ endl;
    unlock_cerr_mutex();
} /* else if (status ≡ 0) */
```

473.

```
<Scan_Parse_Parameter_Type::store_dc_metadata definition 444> +≡
else /* status > 0 */
{
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "[Thread_" ≪ thread_ctr ≪ "]_“" ≪
      "In_‘Scan_Parse_Parameter_Type::store_dc_metadata’::" ≪ endl ≪
      "'Irods_Object_Type::get_from_database’_succeeded,_returning_" ≪ status ≪ ".” ≪
      endl;
    ref_irods_object.show("ref_irods_object:");
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

474.

Log

[LDF 2013.08.16.] BUG FIX: Added *Irods_AVU_TYPE avu_1*. Now calling *Irods_AVU_TYPE :: write_to_database* on both *avu* and *avu_1*. Previously, only *avu* was used. It was reset and *Irods_Object :: add_avu* was called twice, but *Irods_AVU_TYPE :: write_to_database* was only called once and after *avu* had been reset.

```
< Scan_Parse_Parameter_Type :: store_dc_metadata definition 444 > +≡
    Irods_AVU_Type avu("DC_METADATA_IRODS_OBJECT_REF", dc_metadata_irods_object_path);
    Irods_AVU_Type avu_1("DC_METADATA_IRODS_OBJECT_PID", curr_pid);
    avu.irods_object_id = avu_1.irods_object_id = ref_irods_object.id;
    status = ref_irods_object.add_avu(avu, irods_env_filename, true, false, false);
    if (status ≡ 0) {
        status = ref_irods_object.add_avu(avu_1, irods_env_filename, true, false, false);
    }
    if (status ≠ 0) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "] ERROR!" <<
            "In 'Scan_Parse_Parameter_Type::store_dc_metadata':" << endl <<
            "'Irods_Object_Type::add_avu' failed, returning" << status << "." << endl <<
            "Failed to add AVU to iRODS object" << ref_irods_object.path << "." << endl <<
            "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        ++errors_occurred;
        return 1;
    } /* if (status ≠ 0) */
}
```

475.

```
< Scan_Parse_Parameter_Type :: store_dc_metadata definition 444 > +≡
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "[Thread" << thread_ctr << "]"
                << "In 'Scan_Parse_Parameter_Type::store_dc_metadata':" << endl <<
                "'Irods_Object_Type::add_avu' succeeded, returning 0."
                << endl <<
                "Added AVU to iRODS object" << ref_irods_object.path << " successfully."
                << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

476.

```

⟨ Scan_Parse_Parameter_Type::store_dc_metadata definition 444 ⟩ +≡
    status = avu.write_to_database(mysql_ptr, thread_ctr);
    if (status == 0) status = avu_1.write_to_database(mysql_ptr, thread_ctr);
    if (status != 0) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "]ERROR!" <<
            "In 'Scan_Parse_Parameter_Type::store_dc_metadata':" << endl <<
            "'Irods_AVU_Type::write_to_database' failed, returning" << status << "." << endl <<
            "Failed to write 'avu' or 'avu_1' to 'gwiridsif.Irods_AVUs' database table." <<
            endl << "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        ++errors_occurred;
        return 1;
    }
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "]"
            "In 'Scan_Parse_Parameter_Type::store_dc_metadata':" << endl <<
            "'Irods_AVU_Type::write_to_database' succeeded, returning 0." << endl <<
            "Wrote 'avu' and 'avu_1' to 'gwiridsif.Irods_AVUs' database table successfully." <<
            endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* else (status > 0) */

```

477.

```

⟨ Scan_Parse_Parameter_Type::store_dc_metadata definition 444 ⟩ +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "]"
            << "Exiting 'Scan_Parse_Parameter_Type::store_dc_metadata'" <<
            "re_dc_metadata successfully" << "with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; } /* End of Scan_Parse_Parameter_Type::store_dc_metadata */

```

478. Show certificates. [LDF 2013.05.15.]

Log

[LDF 2013.05.15.] Added this function.

```
{ Scan_Parse_Parameter_Type :: show_certificates definition 478 } ≡
int Scan_Parse_Parameter_Type :: show_certificates(Response_Type &response, char *buffer, size_t
    buffer_size, string &filename){ bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    stringstream sql_strm;
    MYSQL_RES * result = 0;
    unsigned int row_ctr;
    unsigned int field_ctr;
    MYSQL_ROW curr_row;
    int status;
    stringstream temp_strm;
    temp_strm << " [Thread]" << thread_ctr << "] ";
    string thread_ctr_str = temp_strm.str();
    temp_strm.str("");
    #if DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_ctr_str << "Entering 'Scan_Parse_Parameter_Type::show_certificates'." <<
                endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
    #endif /* DEBUG_COMPILE */
    memset(buffer, 0, buffer_size);
```

See also sections 479, 480, 481, 482, 483, 484, 485, 486, 487, and 488.

This code is used in section 544.

479.

Log

[LDF 2013.05.16.] Added this section.

```

⟨ Scan_Parse_Parameter_Type::show_certificates definition 478 ⟩ +≡
if (response.int_val ≡ 0) {
    if (privileges & SHOW_CERTIFICATES_PRIVILEGE ≡ 0_U) { /* Normally, this code should never be
        reached, because this case is caught in yyparse. [LDF 2013.05.16.] */
        lock_cerr_mutex();
        cerr << thread_ctr_str << "ERROR! In 'Scan_Parse_Parameter_Type::show_certificates':"
        endl << "User" << username << "(ID" << user_id <<
        ")isnotauthorizedto" << "showallcertificates." << endl <<
        "Exitingfunctionunsuccessfullywithreturnvalue1." << endl;
        unlock_cerr_mutex();
        strcpy(buffer, "SHOW_CERTIFICATE_RESPONSE_3");
        ++errors_occurred;
        return 1;
    } /* if (privileges & SHOW_CERTIFICATES_PRIVILEGE ≡ 0_U) */
#if DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::show_certificates':"
            endl << "User" << username << "(ID" << user_id << ")isauthorizedto" <<
            "showallcertificates." << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (response.int_val ≡ 0) */

```

480.

```

⟨ Scan_Parse_Parameter_Type :: show_certificates definition 478 ⟩ +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::show_certificates':"
        response.show("response:");
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    sql_strm << "select C.certificate_id, C.user_id, U.username, C.issuer_cert_id, C.is_ca, "
    << "C.is_proxy, C.serialNumber, C.organization, C.organizationalUnitName, "
    << "C.commonName, C.countryName, C.localityName, C.stateOrProvinceName, "
    << "C.Validity_notBefore, C.Validity_notAfter "
    << "from gwirdsif.Certificates as C, Users as U "
    << "where C.certificate_id > 0 and C.user_id = U.user_id";
    if (response.int_val == 1) sql_strm << "and C.user_id = "
    << user_id << " ";
    sql_strm << "order by certificate_id";
#endif /* DEBUG_COMPILE */
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::show_certificates':"
        << " 'sql_strm.str()' == "
        << sql_strm.str() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    status = submit_mysql_query(sql_strm.str(), result, &row_ctr, &field_ctr);
    if (status != 0) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "ERROR! In 'Scan_Parse_Parameter_Type::show_certificates':"
        << endl << "'Scan_Parse_Parameter_Type::submit_mysql_query' failed, returning"
        << status << ":" << endl << mysql_error(mysql_ptr) << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        if (result) mysql_free_result(result);
        strcpy(buffer, "SHOW CERTIFICATE RESPONSE 1");
        ++errors_occurred;
        return 1;
    } /* if (status != 0) */
#endif /* DEBUG_COMPILE */
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::show_certificates':"
            << endl << "'Scan_Parse_Parameter_Type::submit_mysql_query' succeeded, returning 0:"
            << endl << "'row_ctr' == "
            << row_ctr << endl << "'field_ctr' == "
            << field_ctr << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

481.

```

⟨ Scan_Parse_Parameter_Type::show_certificates definition 478 ⟩ +≡
vector<X509_Cert_Type> cert_vector;
X509_Cert_Type curr_cert;
for (int i = 0; i < row_ctr; ++i) {
    if ((curr_row = mysql_fetch_row(result)) == 0) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "ERROR! " << "In 'Scan_Parse_Parameter_Type::show_cer"
            "tificates':" << endl << "'mysql_fetch_row' failed:" << endl << mysql_error(mysql_ptr) <<
            endl << "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        mysql_free_result(result);
        temp_strm.str("");
        memset(buffer, 0, buffer_size);
        strcpy(buffer, "SHOW_CERTIFICATE_RESPONSE_1");
        ++errors_occurred;
        return 1;
    } /* if (curr_row = mysql_fetch_row(result) == 0) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::show_certificates':" << endl <<
            "'mysql_fetch_row' succeeded, returning 0." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
status = curr_cert.set(curr_row, thread_ctr_str);
if (status != 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! " << "In 'Scan_Parse_Parameter_Type::show_cer"
        "tificates':" << endl << "'x509_Cert_Type::set' failed, returning " << status <<
        "." << endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    temp_strm.str("");
    memset(buffer, 0, buffer_size);
    strcpy(buffer, "SHOW_CERTIFICATE_RESPONSE_1");
    ++errors_occurred;
    return 1;
} /* if (status != 0) */
#endif 0
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::show_certificates':" << endl <<
            "'x509_Cert_Type::set' succeeded, returning 0." << endl;
        curr_cert.show("curr_cert:");
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

```
#endif
```

```
cert_vector.push_back(curr_cert);
curr_cert.clear();
} /* for */
```

482.

```
< Scan_Parse_Parameter_Type :: show_certificates definition 478 > +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "cert_vector.size() == " << cert_vector.size() << endl;
        if (cert_vector.size() > 0) cerr << "Showing 'cert_vector': " << endl;
        for (vector<X509_Cert_Type>::iterator iter = cert_vector.begin(); iter != cert_vector.end();
            ++iter) {
            iter->show();
            cerr << "'iter->output()' :" << endl << iter->output() << endl;
        } /* for */
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
temp_strm.str("");
for (vector<X509_Cert_Type>::iterator iter = cert_vector.begin(); iter != cert_vector.end(); ++iter)
{
    temp_strm << "SHOW_CERTIFICATE_RESPONSE_0_" << iter->output() << "_";
} /* for */
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::show_certificates'" << endl <<
            "'temp_strm.str()' == " << endl << temp_strm.str() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

483.

```
< Scan_Parse_Parameter_Type :: show_certificates definition 478 > +≡
if (result) {
    mysql_free_result(result);
    result = 0;
}
```

484.

```

⟨ Scan_Parse_Parameter_Type :: show_certificates definition 478 ⟩ +≡
  if (temp_strm.str().length() ≥ buffer_size) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::show_certificates':" <<
      endl << "'temp_strm.str()' length() == " << temp_strm.str().length() <<
      endl << "'buffer_size' == " << buffer_size << ")" << endl <<
      "Storing contents of 'temp_strm.str()' in a temporary file." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

485.

```

⟨ Scan_Parse_Parameter_Type :: show_certificates definition 478 ⟩ +≡
  char temp_filename[] = "/tmp/gwirdsif.XXXXXX";
  errno = 0;
  int fd = mkstemp(temp_filename);
  if (fd ≡ -1) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! " << "In 'Scan_Parse_Parameter_Type::show_cer-
      tificates':" << endl << "'mkstemp' failed, returning -1:" << endl << strerror(errno) <<
      endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    temp_strm.str("");
    memset(buffer, 0, buffer_size);
    strcpy(buffer, "SHOW,CERTIFICATE,RESPONSE,1");
    ++ errors_occurred;
    return 1;
  } /* if (fd ≡ -1) */
#ifndef DEBUG_COMPILE
  else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << " " << "In 'Scan_Parse_Parameter_Type::show_certificates':" <<
      endl << "'mkstemp' succeeded. temp_filename == " << temp_filename << "." << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  temp_file_vector.push_back(temp_filename);

```

486.

```

⟨ Scan_Parse_Parameter_Type :: show_certificates definition 478 ⟩ +≡
  errno = 0;
  status = write(fd, temp_strm.str().c_str(), temp_strm.str().length());
  if (status ≤ 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR!" << "In 'Scan_Parse_Parameter_Type::show_cer\
tificates':" << endl << "'write' failed, returning" << status;
    if (status ≡ -1) cerr << ":" << endl << strerror(errno) << endl;
    else if (status ≡ 0) cerr << "(no bytes written)." << endl;
    cerr << "Failed to write to file" << temp_filename << "." << endl <<
      "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    temp_strm.str("");
    memset(buffer, 0, buffer_size);
    strcpy(buffer, "SHOW CERTIFICATE RESPONSE 1");
    ++errors_occurred;
    return 1;
  } /* if (status ≤ 0) */
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << thread_ctr_str << " " << "In 'Scan_Parse_Parameter_Type::show_certificates':" <<
        endl << "'write' succeeded. Wrote" << status << " bytes to file" <<
        temp_filename << "." << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  close(fd);
  fd = 0;
  filename = temp_filename;
  memset(buffer, 0, buffer_size); } /* if (temp_strm.str().length() ≥ buffer_size) */

```

487.

```

⟨ Scan_Parse_Parameter_Type :: show_certificates definition 478 ⟩ +≡
  else {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'Scan_Parse_Parameter_Type::show_certificates':" <<
      endl << "'temp_strm.str().length()' == " << temp_strm.str().length() <<
      " " << "(<'buffer_size'==" << buffer_size << ")" << endl <<
      "Not storing contents of 'temp_strm.str()' in a temporary file." <<
      endl << "Copying them to 'buffer'." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  strncpy(buffer, temp_strm.str().c_str(), temp_strm.str().length());
}

```

488.

```
<Scan_Parse_Parameter_Type::show_certificates definition 478> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "Exiting 'Scan_Parse_Parameter_Type::show_certificates'" <<
            "successfully with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; } /* End of Scan_Parse_Parameter_Type::show_certificates definition */
```

489. get privileges. (*get_privileges*). [LDF 2013.05.16.]

Log

[LDF 2013.05.16.] Added this function definition. The declaration is in `scprpmtp.web`.

490.

```
<Scan_Parse_Parameter_Type::get_privileges definition 490> ≡
int Scan_Parse_Parameter_Type::get_privileges(int curr_user_id, unsigned int *privs){ bool
    DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "]"
            << "Entering 'Scan_Parse_Parameter_Type::get_privileges'." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    if (!is_gwirdsif) {
        lock_cerr_mutex();
        cerr << "[Thread" << thread_ctr << "]WARNING!"
            << "In 'Scan_Parse_Parameter_Type::get_privileges':"
            << "'is_gwirdsif' == 'false'. This function is only meant to be
            used by the server" << "program 'gwirdsif'." << endl <<
            "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        ++warnings_occurred;
    }
    return 1;
} /* if (!is_gwirdsif) */
```

See also sections 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, and 510.

This code is used in section 544.

491.

```

⟨ Scan_Parse_Parameter_Type ::get_privileges definition 490 ⟩ +≡
  unsigned int temp_privileges = 0_U;
  bool set_privileges;
  /* If a value > 0 is passed for curr_user_id, then we don't want to set this->privileges. Otherwise, we
   do. [LDF 2013.05.16.] */
  if (curr_user_id ≤ 0) {
    curr_user_id = user_id;
    set_privileges = true;
  }
  else set_privileges = false;
  if (curr_user_id ≤ 0) {
    lock_cerr_mutex();
    cerr << "[Thread_]" << thread_ctr << "]_ERROR!" <<
      "In '_Scan_Parse_Parameter_Type::get_privileges':" << endl << "'curr_user_id'==_" <<
      curr_user_id << "_(<=0)." << endl << "User_unknown._Can't_retrieve_privileges." <<
      endl << "Exiting_function_unsuccessfully_with_return_value_1." << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
  }
  return 1;
} /* if (curr_user_id ≤ 0) */

```

492.

```

⟨ Scan_Parse_Parameter_Type ::get_privileges definition 490 ⟩ +≡
  MYSQL_RES * result = 0;
  unsigned int row_ctr;
  unsigned int field_ctr;
  MYSQL_ROW curr_row;
  stringstream sql_strm;
  sql_strm << "select_distinct_superuser,_delegate,_delete_handles,_" <<
    "delete_handle_values,_delete_hs_admin_handle_values,_" <<
    "delete_last_hs_admin_handle_value,_undelete_handle_values,_" <<
    "show_user_info,_show_groups,_" << "show_certificates,_show_distinguished_names,_" <<
    "show_privileges_" << "from_gwiridsif.Privileges_where_user_id=_" << curr_user_id;
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread_]" << thread_ctr << "]_sql_strm.str()=_" << sql_strm.str() << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

493.

```

⟨ Scan_Parse_Parameter_Type::get_privileges definition 490 ⟩ +≡
status = submit_mysql_query(sql_strm.str(), result, &row_ctr, &field_ctr);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << "[Thread]" << thread_ctr << "] ERROR!" <<
        "In 'Scan_Parse_Parameter_Type::get_privileges':" << endl <<
        "'Scan_Parse_Parameter_Type::submit_mysql_query' failed, returning" <<
        status << "." << endl << mysql_error(mysql_ptr) << endl <<
        "Failed to retrieve privileges for user" << curr_user_id << "." << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
if (result) {
    mysql_free_result(result);
}
++errors_occurred;
return 1;
} /* if (status ≠ 0) */
#if DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "[Thread]" << thread_ctr << "] In 'Scan_Parse_Parameter_Type::get_privileges':" <<
                endl << "'Scan_Parse_Parameter_Type::submit_mysql_query' succeeded." << endl <<
                "'row_ctr' == " << row_ctr << endl << "'field_ctr' == " << field_ctr << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

494.

```

⟨ Scan_Parse_Parameter_Type::get_privileges definition 490 ⟩ +≡
if (row_ctr ≡ 0) {
#if DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread]" << thread_ctr << "] " <<
            "In 'Scan_Parse_Parameter_Type::get_privileges':" << endl <<
            "'Scan_Parse_Parameter_Type::submit_mysql_query' returned 0 rows." <<
            endl << "No entry for user" << curr_user_id <<
            " in database table" << "gwiridsif.Privileges'." << endl <<
            "Exiting function successfully with return value 2." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    if (set_privileges) privileges = 0;
    if (privs) *privs = 0;
    return 2;
} /* if (row_ctr ≡ 0) */

```

495.

```
< Scan_Parse_Parameter_Type::get_privileges definition 490 > +≡
else
  if (row_ctr > 1) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "]WARNING!" <<
      "In 'Scan_Parse_Parameter_Type::get_privileges':" << endl <<
      "'Scan_Parse_Parameter_Type::submit_mysql_query' returned" <<
      row_ctr << " rows(>1)." << endl << "This shouldn't ever happen." << endl <<
      "Will use the data from the first row. Continuing." << endl;
    unlock_cerr_mutex();
    ++warnings_occurred;
  } /* if (row_ctr > 1) */
```

496.

```
< Scan_Parse_Parameter_Type::get_privileges definition 490 > +≡
if ((curr_row = mysql_fetch_row(result)) == 0) {
  lock_cerr_mutex();
  cerr << "[Thread" << thread_ctr << "]ERROR!" <<
    "In 'Scan_Parse_Parameter_Type::get_privileges':" << endl <<
    "'mysql_fetch_row' failed:" << endl << mysql_error(mysql_ptr) << endl <<
    "Exiting function unsuccessfully with return value 1." << endl;
  unlock_cerr_mutex();
  mysql_free_result(result);
  ++errors_occurred;
  return 1;
} /* if (curr_row = mysql_fetch_row(result) == 0) */
#ifndef DEBUG_COMPILE
else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "]In 'Scan_Parse_Parameter_Type::get_privileges':" <<
      endl << "'mysql_fetch_row' succeeded." << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

497.

```
< Scan_Parse_Parameter_Type::get_privileges definition 490 > +≡
field_ctr = 0;
unsigned long temp_val = 0UL;
```

498. superuser. [LDF 2013.05.16.]

```

⟨ Scan_Parse_Parameter_Type::get_privileges definition 490 ⟩ +≡
  if (curr_row[field_ctr] ∧ strlen(curr_row[field_ctr]) > 0) {
    errno = 0;
    temp_val = strtoul(curr_row[field_ctr], 0, 10);
    if (temp_val ≡ ULONG_MAX) {
      lock_cerr_mutex();
      cerr << "[Thread]" << thread_ctr << "] ERROR!" <<
        "In 'Scan_Parse_Parameter_Type::get_privileges':" << endl <<
        "'strtoul' failed, returning 'ULONG_MAX'." << endl << strerror(errno) <<
        endl << "Exiting function unsuccessfully with return value 1." << endl;
      unlock_cerr_mutex();
      mysql_free_result(result);
      ++errors_occurred;
      return 1;
    } /* if (temp_val ≡ ULONG_MAX) */
  #if DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "[Thread]" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::get_priv\"
          ileges':" << endl << "'strtoul' succeeded, returning" << temp_val << "." << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
  #endif /* DEBUG_COMPILE */
  if (temp_val > 0) {
    temp_privileges = ~0U;
    bitset<sizeof(unsigned int) * 8> temp_bitset(temp_privileges);
  #if DEBUG_COMPILE
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "[Thread]" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::get_priv\"
          ileges':" << endl << "'temp_val' == " << temp_val << "(>0)" << endl << "User" <<
          curr_user_id << " has " << "superuser" "privilege." << endl << "'temp_privileges' == " <<
          temp_bitset << endl << "Skipping to 'END_ASSIGN' label, because there's no" <<
          "need to check other fields." << endl;
      unlock_cerr_mutex();
    } /* if (DEBUG) */
  #endif /* DEBUG_COMPILE */
  goto END_ASSIGN; /* Don't bother checking the other fields if the user has the "superuser"
                  privilege. [LDF 2013.05.16.] */
  } /* if (temp_val) */
  /* if (curr_row[field_ctr] ∧ strlen(curr_row[field_ctr]) > 0) */
  else {
  #if DEBUG_COMPILE
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "[Thread]" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::get_priv\"
          ileges':" << endl << "'curr_row[" << field_ctr << "]' is NULL or empty." << endl <<
          "Not setting " << "superuser" "privilege for user" << curr_user_id << "." << endl;
      unlock_cerr_mutex();
    }
  
```

```
    } /* if (DEBUG) */  
#endif /* DEBUG_COMPILE */  
} /* else */
```

499. delegate. [LDF 2013.05.16.]

```
<Scan_Parse_Parameter_Type::get_privileges definition 490> +≡
++field_ctr;
if (curr_row[field_ctr] ∧ strlen(curr_row[field_ctr]) > 0) {
    errno = 0;
    temp_val = strtoul(curr_row[field_ctr], 0, 10);
    if (temp_val ≡ ULONG_MAX) {
        lock_cerr_mutex();
        cerr << "[Thread]" << thread_ctr << "] ERROR!" <<
            "In 'Scan_Parse_Parameter_Type::get_privileges':" << endl <<
            "'strtoul' failed, returning 'ULONG_MAX'." << endl << strerror(errno) <<
            endl << "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        mysql_free_result(result);
        ++errors_occurred;
        return 1;
    } /* if (temp_val ≡ ULONG_MAX) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread]" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::get_priv\"
            ileges':" << endl << "'strtoul' succeeded, returning" << temp_val << "." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    if (temp_val > 0) {
#endif DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "[Thread]" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::get_priv\"
                ileges':" << endl << "'temp_val'" << temp_val << "(>0)" << endl <<
                "Setting \"delegate\" privilege for user" << curr_user_id << "." << endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif DEBUG_COMPILE
        temp_privileges |= DELEGATE_PRIVILEGE;
    } /* if (temp_val) */
} /* if (curr_row[field_ctr] ∧ strlen(curr_row[field_ctr]) > 0) */
else {
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread]" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::get_priv\"
            ileges':" << endl << "'curr_row[" << field_ctr << "] ' is NULL or empty." << endl <<
            "Not setting \"delegate\" privilege for user" << curr_user_id << "." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif DEBUG_COMPILE
}
```

500. *delete_handles*. [LDF 2013.07.04.]

```

⟨ Scan_Parse_Parameter_Type::get_privileges definition 490 ⟩ +≡
++field_ctr;
if (curr_row[field_ctr] ∧ strlen(curr_row[field_ctr]) > 0) {
    errno = 0;
    temp_val = strtoul(curr_row[field_ctr], 0, 10);
    if (temp_val ≡ ULONG_MAX) {
        lock_cerr_mutex();
        cerr ≪ "[Thread]" ≪ thread_ctr ≪ "] ERROR!" ≪
            "In 'Scan_Parse_Parameter_Type::get_privileges':" ≪ endl ≪
            "'strtoul' failed, returning 'ULONG_MAX'." ≪ endl ≪ strerror(errno) ≪
            endl ≪ "Exiting function unsuccessfully with return value 1." ≪ endl;
        unlock_cerr_mutex();
        mysql_free_result(result);
        ++errors_occurred;
        return 1;
    } /* if (temp_val ≡ ULONG_MAX) */
#if DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr ≪ "[Thread]" ≪ thread_ctr ≪ "] " ≪ "In 'Scan_Parse_Parameter_Type::get_priv\"
                ileges':" ≪ endl ≪ "'strtoul' succeeded, returning" ≪ temp_val ≪ "." ≪ endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
        if (temp_val > 0) {
#if DEBUG_COMPILE
            if (DEBUG) {
                lock_cerr_mutex();
                cerr ≪ "[Thread]" ≪ thread_ctr ≪ "] " ≪ "In 'Scan_Parse_Parameter_Type::get_priv\"
                    ileges':" ≪ endl ≪ "'temp_val' == " ≪ temp_val ≪ "(>0)" ≪ endl ≪
                    "Setting \"delete_handles\" privilege for user" ≪ curr_user_id ≪ "." ≪ endl;
                unlock_cerr_mutex();
            } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
            temp_privileges |= DELETE_HANDLES_PRIVILEGE;
        } /* if (temp_val) */
    } /* if (curr_row[field_ctr] ∧ strlen(curr_row[field_ctr]) > 0) */
    else {
#if DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr ≪ "[Thread]" ≪ thread_ctr ≪ "] " ≪ "In 'Scan_Parse_Parameter_Type::get_priv\"
                ileges':" ≪ endl ≪ "'curr_row[" ≪ field_ctr ≪ "]' is NULL or empty." ≪ endl ≪
                "Not setting \"delete_handles\" privilege for user" ≪ curr_user_id ≪ "." ≪ endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    }
}

```

501. *delete_handle_values.* [LDF 2013.09.12.]

```

⟨ Scan_Parse_Parameter_Type::get_privileges definition 490 ⟩ +≡
++field_ctr;
if (curr_row[field_ctr] ∧ strlen(curr_row[field_ctr]) > 0) {
    errno = 0;
    temp_val = strtoul(curr_row[field_ctr], 0, 10);
    if (temp_val ≡ ULONG_MAX) {
        lock_cerr_mutex();
        cerr ≪ "[Thread]" ≪ thread_ctr ≪ "] ERROR!" ≪
            "In 'Scan_Parse_Parameter_Type::get_privileges':" ≪ endl ≪
            "'strtoul' failed, returning 'ULONG_MAX'." ≪ endl ≪ strerror(errno) ≪
            endl ≪ "Exiting function unsuccessfully with return value 1." ≪ endl;
        unlock_cerr_mutex();
        mysql_free_result(result);
        ++errors_occurred;
        return 1;
    } /* if (temp_val ≡ ULONG_MAX) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ "[Thread]" ≪ thread_ctr ≪ "] " ≪ "In 'Scan_Parse_Parameter_Type::get_priv\"
            ileges':" ≪ endl ≪ "'strtoul' succeeded, returning" ≪ temp_val ≪ "." ≪ endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    if (temp_val > 0) {
#endif DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr ≪ "[Thread]" ≪ thread_ctr ≪ "] " ≪ "In 'Scan_Parse_Parameter_Type::get_priv\"
                ileges':" ≪ endl ≪ "'temp_val' == " ≪ temp_val ≪ "(>0)" ≪ endl ≪
                "Setting \"delete_handle_values\" privilege for user" ≪ curr_user_id ≪ "." ≪
                endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        temp_privileges |= DELETE_HANDLE_VALUES_PRIVILEGE;
    } /* if (temp_val) */
} /* if (curr_row[field_ctr] ∧ strlen(curr_row[field_ctr]) > 0) */
else {
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ "[Thread]" ≪ thread_ctr ≪ "] " ≪ "In 'Scan_Parse_Parameter_Type::get_priv\"
            ileges':" ≪ endl ≪ "'curr_row[" ≪ field_ctr ≪ "]' is NULL or empty." ≪ endl ≪
            "Not setting \"delete_handle_values\" privilege for user" ≪ curr_user_id ≪ "." ≪
            endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
}
}
```

502. *delete_hs_admin_handle_values.* [LDF 2013.09.12.]

```

⟨ Scan_Parse_Parameter_Type::get_privileges definition 490 ⟩ +≡
++field_ctr;
if (curr_row[field_ctr] ∧ strlen(curr_row[field_ctr]) > 0) {
    errno = 0;
    temp_val = strtoul(curr_row[field_ctr], 0, 10);
    if (temp_val ≡ ULONG_MAX) {
        lock_cerr_mutex();
        cerr << "[Thread]" << thread_ctr << "] ERROR!" <<
            "In 'Scan_Parse_Parameter_Type::get_privileges':" << endl <<
            "'strtoul' failed, returning 'ULONG_MAX'." << endl << strerror(errno) <<
            endl << "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        mysql_free_result(result);
        ++errors_occurred;
        return 1;
    } /* if (temp_val ≡ ULONG_MAX) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread]" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::get_priv\"
            ileges':" << endl << "'strtoul' succeeded, returning" << temp_val << "." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    if (temp_val > 0) {
#endif DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "[Thread]" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::get_priv\"
                ileges':" << endl << "'temp_val' == " << temp_val << "(>0)" << endl <<
                "Setting \"delete_hs_admin_handle_values\" privilege for user" << curr_user_id <<
                "." << endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        temp_privileges |= DELETE_HS_ADMIN_HANDLE_VALUES_PRIVILEGE;
    } /* if (temp_val) */
} /* if (curr_row[field_ctr] ∧ strlen(curr_row[field_ctr]) > 0) */
else {
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread]" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::get_priv\"
            ileges':" << endl << "'curr_row[" << field_ctr << "]' is NULL or empty." <<
            endl << "Not setting \"delete_hs_admin_handle_values\" privilege for user" <<
            curr_user_id << "." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
}

```

503. *delete_last_hs_admin_handle_value*. [LDF 2013.09.12.]

```

⟨ Scan_Parse_Parameter_Type::get_privileges definition 490 ⟩ +≡
++field_ctr;
if (curr_row[field_ctr] ∧ strlen(curr_row[field_ctr]) > 0) {
    errno = 0;
    temp_val = strtoul(curr_row[field_ctr], 0, 10);
    if (temp_val ≡ ULONG_MAX) {
        lock_cerr_mutex();
        cerr ≪ "[Thread]" ≪ thread_ctr ≪ "] ERROR!" ≪
            "In 'Scan_Parse_Parameter_Type::get_privileges':" ≪ endl ≪
            "'strtoul' failed, returning 'ULONG_MAX'." ≪ endl ≪ strerror(errno) ≪
            endl ≪ "Exiting function unsuccessfully with return value 1." ≪ endl;
        unlock_cerr_mutex();
        mysql_free_result(result);
        ++errors_occurred;
        return 1;
    } /* if (temp_val ≡ ULONG_MAX) */
#if DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr ≪ "[Thread]" ≪ thread_ctr ≪ "] " ≪ "In 'Scan_Parse_Parameter_Type::get_priv\"
                ileges':" ≪ endl ≪ "'strtoul' succeeded, returning" ≪ temp_val ≪ "." ≪ endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
        if (temp_val > 0) {
#if DEBUG_COMPILE
            if (DEBUG) {
                lock_cerr_mutex();
                cerr ≪ "[Thread]" ≪ thread_ctr ≪ "] " ≪ "In 'Scan_Parse_Parameter_Type::get_priv\"
                    ileges':" ≪ endl ≪ "'temp_val' == " ≪ temp_val ≪ "(>0)" ≪ endl ≪
                    "Setting \"delete_last_hs_admin_handle_value\" privilege for user" ≪
                    curr_user_id ≪ "." ≪ endl;
                unlock_cerr_mutex();
            } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
                temp_privileges |= DELETE_LAST_HS_ADMIN_HANDLE_VALUE_PRIVILEGE;
            } /* if (temp_val) */
        } /* if (curr_row[field_ctr] ∧ strlen(curr_row[field_ctr]) > 0) */
        else {
#if DEBUG_COMPILE
            if (DEBUG) {
                lock_cerr_mutex();
                cerr ≪ "[Thread]" ≪ thread_ctr ≪ "] " ≪ "In 'Scan_Parse_Parameter_Type::get_priv\"
                    ileges':" ≪ endl ≪ "'curr_row[" ≪ field_ctr ≪ "]' is NULL or empty." ≪ endl ≪
                    "Not setting \"delete_last_hs_admin_handle_value\" privilege for user" ≪
                    curr_user_id ≪ "." ≪ endl;
                unlock_cerr_mutex();
            } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        }
    }
```

504. *undelete_handle_values*. [LDF 2013.09.12.]

```

⟨ Scan_Parse_Parameter_Type::get_privileges definition 490 ⟩ +≡
++field_ctr;
if (curr_row[field_ctr] ∧ strlen(curr_row[field_ctr]) > 0) {
    errno = 0;
    temp_val = strtoul(curr_row[field_ctr], 0, 10);
    if (temp_val ≡ ULONG_MAX) {
        lock_cerr_mutex();
        cerr ≪ "[Thread]" ≪ thread_ctr ≪ "] ERROR!" ≪
            "In 'Scan_Parse_Parameter_Type::get_privileges':" ≪ endl ≪
            "'strtoul' failed, returning 'ULONG_MAX'." ≪ endl ≪ strerror(errno) ≪
            endl ≪ "Exiting function unsuccessfully with return value 1." ≪ endl;
        unlock_cerr_mutex();
        mysql_free_result(result);
        ++errors_occurred;
        return 1;
    } /* if (temp_val ≡ ULONG_MAX) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ "[Thread]" ≪ thread_ctr ≪ "] " ≪ "In 'Scan_Parse_Parameter_Type::get_priv\"
            ileges':" ≪ endl ≪ "'strtoul' succeeded, returning" ≪ temp_val ≪ "." ≪ endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    if (temp_val > 0) {
#endif DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr ≪ "[Thread]" ≪ thread_ctr ≪ "] " ≪ "In 'Scan_Parse_Parameter_Type::get_priv\"
                ileges':" ≪ endl ≪ "'temp_val' == " ≪ temp_val ≪ "(>0)" ≪ endl ≪
                "Setting \"undelete_handle_values\" privilege for user" ≪ curr_user_id ≪ "." ≪
                endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        temp_privileges |= UNDELETE_HANDLE_VALUES_PRIVILEGE;
    } /* if (temp_val) */
} /* if (curr_row[field_ctr] ∧ strlen(curr_row[field_ctr]) > 0) */
else {
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ "[Thread]" ≪ thread_ctr ≪ "] " ≪ "In 'Scan_Parse_Parameter_Type::get_priv\"
            ileges':" ≪ endl ≪ "'curr_row[" ≪ field_ctr ≪ "]' is NULL or empty." ≪ endl ≪
            "Not setting \"undelete_handle_values\" privilege for user" ≪ curr_user_id ≪
            "." ≪ endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
}
}
```

505. show user info (*show_user_info*). [LDF 2013.05.17.]

Log

[LDF 2013.05.17.] Added this section.

```

⟨ Scan_Parse_Parameter_Type::get_privileges definition 490 ⟩ +≡
  ++field_ctr;
  if (curr_row[field_ctr] ∧ strlen(curr_row[field_ctr]) > 0) {
    errno = 0;
    temp_val = strtoul(curr_row[field_ctr], 0, 10);
    if (temp_val ≡ ULONG_MAX) {
      lock_cerr_mutex();
      cerr << "[Thread" << thread_ctr << "] ERROR!" <<
        "In 'Scan_Parse_Parameter_Type::get_privileges':" << endl <<
        "'strtoul' failed, returning 'ULONG_MAX'." << endl << strerror(errno) <<
        endl << "Exiting function unsuccessfully with return value 1." << endl;
      unlock_cerr_mutex();
      mysql_free_result(result);
      ++errors_occurred;
      return 1;
    } /* if (temp_val ≡ ULONG_MAX) */
  #if DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "[Thread" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::get_priv\"
          ileges':" << endl << "'strtoul' succeeded, returning" << temp_val << "." << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
  #endif /* DEBUG_COMPILE */
  if (temp_val > 0) {
  #if DEBUG_COMPILE
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "[Thread" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::get_priv\"
          ileges':" << endl << "'temp_val' == " << temp_val << "(>0)" << endl <<
          "Setting \"show_user_info\" privilege for user" << curr_user_id << "." << endl;
      unlock_cerr_mutex();
    } /* if (DEBUG) */
  #endif /* DEBUG_COMPILE */
    temp_privileges |= SHOW_USER_INFO_PRIVILEGE;
  } /* if (temp_val) */
  } /* if (curr_row[field_ctr] ∧ strlen(curr_row[field_ctr]) > 0) */
  else {
  #if DEBUG_COMPILE
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "[Thread" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::get_priv\"
          ileges':" << endl << "'curr_row[" << field_ctr << "]' is NULL or empty." << endl <<
          "Not setting \"show_user_info\" privilege for user" << curr_user_id << "." << endl;
      unlock_cerr_mutex();
    } /* if (DEBUG) */
  
```

```
#endif /* DEBUG_COMPILE */  
}
```

506. show groups (*show-groups*). [LDF 2013.06.05.]

Log

[LDF 2013.06.05.] Added this section.

```

⟨ Scan_Parse_Parameter_Type::get_privileges definition 490 ⟩ +≡
  ++field_ctr;
  if (curr_row[field_ctr] ∧ strlen(curr_row[field_ctr]) > 0) {
    errno = 0;
    temp_val = strtoul(curr_row[field_ctr], 0, 10);
    if (temp_val ≡ ULONG_MAX) {
      lock_cerr_mutex();
      cerr << "[Thread" << thread_ctr << "] ERROR!" <<
        "In 'Scan_Parse_Parameter_Type::get_privileges':" << endl <<
        "'strtoul' failed, returning 'ULONG_MAX'." << endl << strerror(errno) <<
        endl << "Exiting function unsuccessfully with return value 1." << endl;
      unlock_cerr_mutex();
      mysql_free_result(result);
      ++errors_occurred;
      return 1;
    } /* if (temp_val ≡ ULONG_MAX) */
  #if DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "[Thread" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::get_priv\"
          ileges':" << endl << "'strtoul' succeeded, returning" << temp_val << "." << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
  #endif /* DEBUG_COMPILE */
  if (temp_val > 0) {
  #if DEBUG_COMPILE
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "[Thread" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::get_priv\"
          ileges':" << endl << "'temp_val' == " << temp_val << "(>0)" << endl <<
          "Setting \"show_groups\" privilege for user" << curr_user_id << "." << endl;
      unlock_cerr_mutex();
    } /* if (DEBUG) */
  #endif /* DEBUG_COMPILE */
    temp_privileges |= SHOW_GROUPS_PRIVILEGE;
  } /* if (temp_val) */
  } /* if (curr_row[field_ctr] ∧ strlen(curr_row[field_ctr]) > 0) */
  else {
  #if DEBUG_COMPILE
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "[Thread" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::get_priv\"
          ileges':" << endl << "'curr_row[" << field_ctr << "] ' is NULL or empty." << endl <<
          "Not setting \"show_groups\" privilege for user" << curr_user_id << "." << endl;
      unlock_cerr_mutex();
    } /* if (DEBUG) */
  
```

```
#endif /* DEBUG_COMPILE */  
}
```

507. show certificates. [LDF 2013.05.16.]

```

⟨ Scan_Parse_Parameter_Type::get_privileges definition 490 ⟩ +≡
++field_ctr;
if (curr_row[field_ctr] ∧ strlen(curr_row[field_ctr]) > 0) {
    errno = 0;
    temp_val = strtoul(curr_row[field_ctr], 0, 10);
    if (temp_val ≡ ULONG_MAX) {
        lock_cerr_mutex();
        cerr ≪ "[Thread]" ≪ thread_ctr ≪ "] ERROR!" ≪
            "In 'Scan_Parse_Parameter_Type::get_privileges':" ≪ endl ≪
            "'strtoul' failed, returning 'ULONG_MAX'." ≪ endl ≪ strerror(errno) ≪
            endl ≪ "Exiting function unsuccessfully with return value 1." ≪ endl;
        unlock_cerr_mutex();
        mysql_free_result(result);
        ++errors_occurred;
        return 1;
    } /* if (temp_val ≡ ULONG_MAX) */
#if DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr ≪ "[Thread]" ≪ thread_ctr ≪ "] " ≪ "In 'Scan_Parse_Parameter_Type::get_priv\"
                ileges':" ≪ endl ≪ "'strtoul' succeeded, returning" ≪ temp_val ≪ "." ≪ endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
        if (temp_val > 0) {
#if DEBUG_COMPILE
            if (DEBUG) {
                lock_cerr_mutex();
                cerr ≪ "[Thread]" ≪ thread_ctr ≪ "] " ≪ "In 'Scan_Parse_Parameter_Type::get_priv\"
                    ileges':" ≪ endl ≪ "'temp_val' == " ≪ temp_val ≪ "(>0)" ≪ endl ≪
                    "Setting \"show_certificates\" privilege for user" ≪ curr_user_id ≪ "." ≪ endl;
                unlock_cerr_mutex();
            } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
            temp_privileges |= SHOW_CERTIFICATES_PRIVILEGE;
        } /* if (temp_val) */
    } /* if (curr_row[field_ctr] ∧ strlen(curr_row[field_ctr]) > 0) */
    else {
#if DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr ≪ "[Thread]" ≪ thread_ctr ≪ "] " ≪ "In 'Scan_Parse_Parameter_Type::get_priv\"
                ileges':" ≪ endl ≪ "'curr_row[" ≪ field_ctr ≪ "]' is NULL or empty." ≪ endl ≪
                "Not setting \"show_certificates\" privilege for user" ≪ curr_user_id ≪ "." ≪
                endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    }
}

```

508. show distinguished names. [LDF 2013.05.16.]

```

⟨ Scan_Parse_Parameter_Type::get_privileges definition 490 ⟩ +≡
++field_ctr;
if (curr_row[field_ctr] ∧ strlen(curr_row[field_ctr]) > 0) {
    errno = 0;
    temp_val = strtoul(curr_row[field_ctr], 0, 10);
    if (temp_val ≡ ULONG_MAX) {
        lock_cerr_mutex();
        cerr ≪ "[Thread]" ≪ thread_ctr ≪ "] ERROR!" ≪
            "In 'Scan_Parse_Parameter_Type::get_privileges':" ≪ endl ≪
            "'strtoul' failed, returning 'ULONG_MAX'." ≪ endl ≪ strerror(errno) ≪
            endl ≪ "Exiting function unsuccessfully with return value 1." ≪ endl;
        unlock_cerr_mutex();
        mysql_free_result(result);
        ++errors_occurred;
        return 1;
    } /* if (temp_val ≡ ULONG_MAX) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ "[Thread]" ≪ thread_ctr ≪ "] " ≪ "In 'Scan_Parse_Parameter_Type::get_priv\"
            ileges':" ≪ endl ≪ "'strtoul' succeeded, returning" ≪ temp_val ≪ "." ≪ endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    if (temp_val > 0) {
#endif DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr ≪ "[Thread]" ≪ thread_ctr ≪ "] " ≪ "In 'Scan_Parse_Parameter_Type::get_priv\"
                ileges':" ≪ endl ≪ "'temp_val' == " ≪ temp_val ≪ "(>0)" ≪ endl ≪
                "Setting \"show_distinguished_names\" privilege for user" ≪ curr_user_id ≪
                "." ≪ endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        temp_privileges |= SHOW_DISTINGUISHED_NAMES_PRIVILEGE;
    } /* if (temp_val) */
} /* if (curr_row[field_ctr] ∧ strlen(curr_row[field_ctr]) > 0) */
else {
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ "[Thread]" ≪ thread_ctr ≪ "] " ≪ "In 'Scan_Parse_Parameter_Type::get_priv\"
            ileges':" ≪ endl ≪ "'curr_row[" ≪ field_ctr ≪ "]' is NULL or empty." ≪ endl ≪
            "Not setting \"show_distinguished_names\" privilege for user" ≪ curr_user_id ≪
            "." ≪ endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
}

```

509. show privileges. [LDF 2013.05.16.]

```
<Scan_Parse_Parameter_Type::get_privileges definition 490> +≡
++field_ctr;
if (curr_row[field_ctr] ∧ strlen(curr_row[field_ctr]) > 0) {
    errno = 0;
    temp_val = strtoul(curr_row[field_ctr], 0, 10);
    if (temp_val ≡ ULONG_MAX) {
        lock_cerr_mutex();
        cerr << "[Thread]" << thread_ctr << "] ERROR!" <<
            "In 'Scan_Parse_Parameter_Type::get_privileges':" << endl <<
            "'strtoul' failed, returning ULONG_MAX." << endl << strerror(errno) <<
            endl << "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        mysql_free_result(result);
        ++errors_occurred;
        return 1;
    } /* if (temp_val ≡ ULONG_MAX) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread]" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::get_priv\"
            ileges':" << endl << "'strtoul' succeeded, returning" << temp_val << "." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    if (temp_val > 0) {
#endif DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "[Thread]" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::get_priv\"
                ileges':" << endl << "'temp_val'" << temp_val << "(>0)" << endl <<
                "Setting \"show privileges\" privilege for user" << curr_user_id << "." << endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        temp_privileges |= SHOW_PRIVILEGES_PRIVILEGE;
    } /* if (temp_val) */
} /* if (curr_row[field_ctr] ∧ strlen(curr_row[field_ctr]) > 0) */
else {
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread]" << thread_ctr << "] " << "In 'Scan_Parse_Parameter_Type::get_priv\"
            ileges':" << endl << "'curr_row[" << field_ctr << "] ' is NULL or empty." << endl <<
            "Not setting \"show privileges\" privilege for user" << curr_user_id << "." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
}
```

510.

```

⟨ Scan_Parse_Parameter_Type :: get_privileges definition 490 ⟩ +≡
END_ASSIGN:
  if (privs) *privs = temp_privileges;
  if (set_privileges) privileges = temp_privileges;
  mysql_free_result(result);
  result = 0;
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << thread_ctr << "] Exiting 'Scan_Parse_Parameter_Type::get_privileges'" << " successfully with return value 0." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  if (result) mysql_free_result(result);
  return 0; } /* Scan_Parse_Parameter_Type :: get_privileges */

```

511. Set user info. (*set_user_info*). [LDF 2013.05.19.]

Log

[LDF 2013.05.19.] Added this function definition. The declaration is in **scprpmtp.web**.

```

⟨ Scan_Parse_Parameter_Type :: set_user_info definition 511 ⟩ ≡
int Scan_Parse_Parameter_Type :: set_user_info(User_Info_Type &user_info) const
{
  user_info.user_id = user_id;
  user_info.username = username;
  user_info.certificate = user_cert;
  user_info.distinguished_name = distinguished_name;
  user_info.privileges = privileges;
  user_info.irods_password_encrypted = irods_password_encrypted;
  user_info.irods_password_encrypted_timestamp = irods_password_encrypted_timestamp;
  user_info.irods_current_dir = irods_current_dir;
  user_info.irods_homedir = irods_homedir;
  user_info.irods_zone = irods_zone;
  user_info.irods_default_resource = irods_default_resource;
  user_info.default_handle_prefix_id = default_handle_prefix_id;
  user_info.default_handle_prefix = default_handle_prefix;
  user_info.default_institute_id = default_institute_id;
  user_info.default_institute_name = default_institute_name;
  return 0;
} /* End of Scan_Parse_Parameter_Type :: set_user_info definition */

```

This code is used in section 544.

512. Show privileges. (*show_privileges*). [LDF 2013.05.22.]

Log

[LDF 2013.05.22.] Added this function definition. The declaration is in `scprpmtp.web`.

```

⟨ Scan_Parse_Parameter_Type :: show_privileges definition 512 ⟩ ≡
int Scan_Parse_Parameter_Type :: show_privileges(unsigned int privileges, ostream * strm, bool
                                                verbose)
{
    stringstream temp_strm;
    temp_strm << "privileges:" << endl;
    if (verbose) {
        bitset<sizeof(unsigned int)*8> b(privileges);
        temp_strm << privileges << "(_decimal)_(" << oct << privileges << "(_octal)_(" << hex <<
            privileges << "(_hexadecimal)_(" << dec << endl << b << "(_binary)_(" << endl;
    }
    temp_strm << "_____superuser:_____"
    ((privileges & SUPERUSER_PRIVILEGE) ? 1 : 0) << "(_uu1)_(" << endl <<
    "____delegate:_____"
    ((privileges & DELEGATE_PRIVILEGE) ? 1 : 0) << "(_uu2)_(" << endl << "____delete_handles:_____"
    ((privileges & DELETE_HANDLES_PRIVILEGE) ? 1 : 0) << "(_uu4)_(" <<
    endl << "____delete_handle_values:_____"
    ((privileges & DELETE_HANDLE_VALUES_PRIVILEGE) ? 1 : 0) <<
    "(_uu8)_(" << endl << "____delete_hs_admin_handle_values:_____"
    ((privileges & DELETE_HS_ADMIN_HANDLE_VALUES_PRIVILEGE) ? 1 : 0) <<
    "(_uu16)_(" << endl << "____delete_last_hs_admin_handle_value:_u"
    ((privileges & DELETE_LAST_HS_ADMIN_HANDLE_VALUE_PRIVILEGE) ? 1 : 0) <<
    "(_uu32)_(" << endl << "____undelete_handle_values:_____"
    ((privileges & UNDELETE_HANDLE_VALUES_PRIVILEGE) ? 1 : 0) <<
    "(_uu64)_(" << endl << "____show_user_info:_____"
    ((privileges & SHOW_USER_INFO_PRIVILEGE) ? 1 : 0) << "(_128)_(" << endl <<
    "____show_groups:_____"
    ((privileges & SHOW_GROUPS_PRIVILEGE) ? 1 : 0) << "(_256)_(" << endl << "____show_certificates:_____"
    ((privileges & SHOW_CERTIFICATES_PRIVILEGE) ? 1 : 0) <<
    "(_512)_(" << endl << "____show_distinguished_names:_____"
    ((privileges & SHOW_DISTINGUISHED_NAMES_PRIVILEGE) ? 1 : 0) << "(_1024)_(" << endl <<
    "____show_privileges:_____"
    ((privileges & SHOW_PRIVILEGES_PRIVILEGE) ? 1 : 0) << "(_2048)_(" << endl;
    if (*strm != 0) *strm << temp_strm.str();
    else cerr << temp_strm.str();
    return 0;
} /* End of Scan_Parse_Parameter_Type :: show_privileges definition */

```

This code is used in section 544.

513. Get highest value (*get_highest_value*). [LDF 2013.06.06.]

Get the highest value of column *column* from database table *table* and store it in *val*. If **bool** *incr* \equiv true, increment it before returning. *incr* is optional, with default false (*val* is not incremented). This function returns 0 upon success, otherwise 1. [LDF 2013.06.06.]

Log

[LDF 2013.06.06.] Added this function definition. The declaration is in `scprpmtp.web`.

```
< Scan_Parse_Parameter_Type :: get_highest_value definition 513 > ≡
int Scan_Parse_Parameter_Type :: get_highest_value(MYSQL * mysql_ptr, string table, string
                                                 column, unsigned long int & val, bool incr){ bool DEBUG = false; /* true */
                                                 set_debug_level(DEBUG, 0, 0);
int status = 0;
stringstream sql_strm;
MYSQL_RES * result = 0;
MYSQL_ROW curr_row;
unsigned int row_ctr;
unsigned int field_ctr;
stringstream temp_strm;
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "Entering `Scan_Parse_Parameter_Type::get_highest_value'." << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

See also sections 514, 515, 516, 517, 518, and 519.

This code is used in section 544.

514.

```
< Scan_Parse_Parameter_Type :: get_highest_value definition 513 > +≡
sql_strm << "select " << column << " from " << table << " order by " << column <<
" desc limit 1";
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In `Scan_Parse_Parameter_Type::get_highest_value': " << endl <<
        "sql_strm.str() == " << sql_strm.str() << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

515.

```

⟨ Scan_Parse_Parameter_Type :: get_highest_value definition 513 ⟩ +≡
status = :: submit_mysql_query(sql_strm.str().c_str(), result, mysql_ptr, &row_ctr, &field_ctr);
if (status ≠ 0) {
    cerr ≪ "ERROR! In 'Scan_Parse_Parameter_Type::get_highest_value':" ≪ endl ≪
        "'submit_mysql_query' failed, returning" ≪ status ≪ "." ≪ endl ≪
        "Failed to retrieve highest value of " ≪ table ≪ "." ≪ column ≪ "." ≪ endl ≪
        "Exiting function unsuccessfully with return value 1." ≪ endl;
unlock_cerr_mutex();
if (result) {
    mysql_free_result(result);
}
return 1;
} /* if (status ≠ 0) */
#endif /* DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "In 'Scan_Parse_Parameter_Type::get_highest_value':" ≪
        endl ≪ "'submit_mysql_query' succeeded, returning 0." ≪ endl ≪
        "Retrieved highest value of " ≪ table ≪ "." ≪ column ≪ "' successfully." ≪ endl;
unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

516.

```

⟨ Scan_Parse_Parameter_Type :: get_highest_value definition 513 ⟩ +≡
if ((curr_row = mysql_fetch_row(result)) ≡ 0) {
    lock_cerr_mutex();
    cerr ≪ "ERROR! In 'Scan_Parse_Parameter_Type::get_highest_value':" ≪ endl ≪
        "'mysql_fetch_row' failed:" ≪ endl ≪ mysql_error(mysql_ptr) ≪ endl ≪
        "Failed to read highest value of " ≪ table ≪ "." ≪ column ≪ "' from " ≪
        "database." ≪ endl ≪ "Exiting function unsuccessfully with return value 1." ≪ endl;
unlock_cerr_mutex();
if (result) {
    mysql_free_result(result);
}
return 1;
} /* if (curr_row = mysql_fetch_row(result) ≡ 0) */
#endif /* DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "In 'Scan_Parse_Parameter_Type::get_highest_value':" ≪ endl ≪
        "'mysql_fetch_row' succeeded." ≪ endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

517.

```
<Scan_Parse_Parameter_Type::get_highest_value definition 513> +≡
  if (curr_row[0] == 0 ∨ strlen(curr_row[0]) == 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Scan_Parse_Parameter_Type::get_highest_value':"
    << endl << "'curr_row[0]' is NULL or empty."
    << endl << "Failed to read highest value of "
    << table << "."
    << column << "' from "
    << "database. Can't set 'val'."
    << endl <<
    "Exiting function unsuccessfully with return value 1."
    << endl;
    unlock_cerr_mutex();
  }
  if (result) {
    mysql_free_result(result);
  }
  return 1;
} /* if (curr_row[0] == 0 ∨ strlen(curr_row[0]) == 0) */
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "curr_row[0] == "
      << curr_row[0]
      << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

518.

```

⟨ Scan_Parse_Parameter_Type ::get_highest_value definition 513 ⟩ +≡
  errno = 0;
  val = strtoul(curr_row[0], 0, 10);
  if (val == ULONG_MAX) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Scan_Parse_Parameter_Type::get_highest_value':"
    << endl << "'strtoul' failed, returning 'ULONG_MAX':"
    << endl << "Error: " << strerror(errno) << endl <<
    "Failed to convert highest value of " << table <<
    "." << column << " from " << "database.Failed to set 'val'." << endl <<
    "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
  if (result) {
    mysql_free_result(result);
  }
  return 1;
} /* if (val == ULONG_MAX) */
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "In 'Scan_Parse_Parameter_Type::get_highest_value':"
      << "'val' == " << val << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  if (incr) {
    ++val; /* Increment val. */
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'Scan_Parse_Parameter_Type::get_highest_value':"
    << "Incremented 'val'. New value: " << val << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  } /* if (incr) */
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "In 'Scan_Parse_Parameter_Type::get_highest_value':"
      << "Not incrementing 'val' == " << val << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

519.

```
< Scan_Parse_Parameter_Type ::get_highest_value definition 513 > +≡
    mysql_free_result(result);
    result = 0;
    sqlStrm.str("");
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Exiting `Scan_Parse_Parameter_Type::get_highest_value'" <<
            "successfully with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; } /* End of Scan_Parse_Parameter_Type::get_highest_value definition */
```

520. Get input (*get_input*). [LDF 2013.07.11.]

Log

[LDF 2013.07.11.] Added this function definition. The declaration is in `scprpmtp.web`.

```
< Scan_Parse_Parameter_Type ::get_input definition 520 > ≡
int Scan_Parse_Parameter_Type ::get_input(void){ bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    char buffer[BUFFER_SIZE];
    memset(buffer, 0, BUFFER_SIZE);
    int char_ctr = 0;
    int fd;
    char temp_filename[21] = "/tmp/gwirdcli.XXXXXX";
    string thread_str;
    stringstream temp_strm;
    bool write_to_file = false;
    ofstream out_strm;
    if (thread_ctr ≥ 0) {
        temp_strm << "[Thread" << thread_ctr << "]";
        thread_str = temp_strm.str();
        temp_strm.str("");
    }
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering `Scan_Parse_Parameter_Type::get_input'." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

See also sections 521, 522, 523, 524, 525, 526, 527, and 528.

This code is used in section 544.

521.

```
(Scan_Parse_Parameter_Type::get_input definition 520) +≡
  if (!is_gwirdcli) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::get_input':"
    << endl << "'is_gwirdcli' == 'false'. Not 'gwirdcli'. This is currently not permitted."
    << endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    return 1;
  }
```

522. If standard input is connected to a terminal, EOF will normally never be read. However, it probably will occur if it's connected to a pipe. If EOF is read, this function exits with return value 6.

If the output of echo or cat has been piped to gwirdcli using the shell, it will not be possible to get additional input via standarad input through a terminal. In this case, it doesn't make sense to set terminate_on_end_input = false. However, it does no harm, since this function simply exits.

!! TODO: It would be possible to change this function so that it could read from other sources, e.g., a file descriptor connected to a pipe. However, I don't think this feature is needed at the present time.

```
(Scan_Parse_Parameter_Type::get_input definition 520) +≡
  if (cin.eof()) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'Scan_Parse_Parameter_Type::get_input':"
    << endl << "Read EOF. Exiting successfully with return value 6."
    << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  return 6;
}
```

523.

```
(Scan_Parse_Parameter_Type::get_input definition 520) +≡
  if (data_filename == "-") data_filename = "";
  if (!data_filename.empty()) {
    lock_cerr_mutex();
    cerr << thread_str << "WARNING! In 'Scan_Parse_Parameter_Type::get_input':"
    << endl << "'data_filename' == "
    << data_filename << endl << "'data_filename' is not empty or \"-\"."
    << endl << "Not getting input from standard input."
    << endl << "Exiting function unsuccessfully with return value 2."
    << endl;
    unlock_cerr_mutex();
    ++warnings_occurred;
    return 2;
  }
```

524.

```
⟨ Scan_Parse_Parameter_Type :: get_input definition 520 ⟩ +≡
if (strlen(data_buffer) > 0) {
    lock_cerr_mutex();
    cerr << thread_str << "WARNING! In 'Scan_Parse_Parameter_Type::get_input':"
    endl << "'strlen(data_buffer)' == " << strlen(data_buffer) << "(>0)" << endl <<
    "'data_buffer' is not empty. Not getting input from standard input." << endl <<
    "Exiting function unsuccessfully with return value 2." << endl;
unlock_cerr_mutex();
++warnings_occurred;
return 2;
}
```

525. If the global variable **unsigned int suppress_prompt** ≡ 0, the prompt is printed to standard output. Otherwise, it's not. If *suppress_prompt* ≡ 1, the prompt is suppressed, but *suppress_prompt* is set to 0 following the code for printing the prompt. The effect is that the prompt will only be suppressed once. If *suppress_prompt* > 1, it is not decremented, so that the prompt will always be suppressed. [LDF 2013.08.29.]

Log

[LDF 2013.08.29.] Added code for suppressing the prompt.

```
⟨ Scan_Parse_Parameter_Type :: get_input definition 520 ⟩ +≡
if (suppress_prompt ≡ 0) {
    lock_cerr_mutex();
    lock_cout_mutex();
    cout << "Enter commands:" << endl << "Type commands follow"
    wed_by_<ENTER>. Multiple lines may be entered." << endl <<
    "Enter a single period ('.') on a line to finish." << endl <<
    "Use 'q' (or 'Q') command to quit." << endl;
unlock_cout_mutex();
unlock_cerr_mutex();
}
else if (suppress_prompt ≡ 1) suppress_prompt = 0;
```

526.

```

⟨ Scan_Parse_Parameter_Type ::get_input definition 520 ⟩ +≡
    char_ctr = 0; for ( ; ; ) { memset(buffer, 0, BUFFER_SIZE);
        cin.getline(buffer, BUFFER_SIZE);
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "cin.gcount() == " << cin.gcount() << endl << "cin.eof() == " << cin.eof() << endl;
        if (cin.gcount() == 0) cerr << "No characters read." << endl;
        else
            cerr << "Read " << strlen(buffer) << " characters." << endl << "buffer == " << buffer << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    if (cin.gcount() == 0 & cin.eof()) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "No characters read and cin.eof() == 'false'." << "Continuing." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        continue;
    }
    else if (cin.gcount() == 0 & cin.eof()) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "No characters read and cin.eof() == 'true'." << "Breaking." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        break;
    }
    else if (strlen(buffer) == 1 & tolower(buffer[0]) == 'q') {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Read " << buffer[0] << ". Will quit." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        memset(data_buffer, 0, BUFFER_SIZE);
        data_filename = "";
        return 4;
    }
    else if (strlen(buffer) == 1 & buffer[0] == '.') {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Read a single period." << "Breaking." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

```
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        break;
    }
    else {
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Read" << strlen(buffer) << "characters." << endl << "buffer" << buffer << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    if (write_to_file) {
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "'write_to_file'" << 'true' . Writing_to 'out_strm' . " << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        out_strm.write(buffer, strlen(buffer));
        out_strm << endl;
    }
    else if ((char_ctr + strlen(buffer) + 1) < BUFFER_SIZE) {
        strcat(data_buffer, buffer);
        strcat(data_buffer, "\n");
        char_ctr += strlen(buffer) + 1;
    }
    else {
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "(char_ctr+strlen(buffer)+1)>=BUFFER_SIZE:" << endl <<
            "char_ctr" << endl << "strlen(buffer)" << endl <<
            "strlen(buffer)" << endl << "char_ctr+strlen(buffer)+1" << endl <<
            "(char_ctr + strlen(buffer) + 1)" << endl << "BUFFER_SIZE" << endl <<
            "(char_ctr+strlen(buffer)+1)>=BUFFER_SIZE" << endl <<
            ((char_ctr + strlen(buffer)) >= BUFFER_SIZE) << endl <<
            "Writing to file." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

527.

```

⟨ Scan_Parse_Parameter_Type ::get_input definition 520 ⟩ +≡
  errno = 0;
  fd = mkstemp(temp_filename);
  if (fd == -1) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::get_input':"
      endl << "'mkstemp' failed, returning -1:" << endl << strerror(errno) <<
      endl << "Failed to create temporary file." << endl <<
      "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    return 1;
  } /* if (fd == -1) */
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << thread_str << "In 'Scan_Parse_Parameter_Type::get_input':"
        endl << "'mkstemp' succeeded." << endl << "'temp_filename' == "
          << temp_filename << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  close(fd); /* We just need the name. [LDF 2013.07.10.] */
  fd = 0;
  data_filename = temp_filename;
  temp_file_vector.push_back(data_filename);
  out_strm.open(data_filename.c_str());
  if (!(out_strm & out_strm.is_open())) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::get_input':"
      endl << "'out_strm.open()' failed." << endl << "Failed to open temporary output file"
      endl << temp_filename << "."
      endl << "Exiting function unsuccessfully with return value 1."
      endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    return 1;
  } /* if (!(out_strm & out_strm.is_open())) */
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << thread_str << "In 'Scan_Parse_Parameter_Type::get_input':"
        endl << "'out_strm.open()' succeeded." << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  out_strm.write(data_buffer, strlen(data_buffer));
  if (out_strm.good()) out_strm.write(buffer, strlen(buffer));
  if (!(out_strm.good())) {
    lock_cerr_mutex();

```

```

cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::get_input': "
    "out_strm.write()' failed," << endl << "out_strm.good() == 'false'." <<
    endl << "Failed to write to temporary file" << data_filename << "."
    << endl << "Exiting function unsuccessfully with return value 1." << endl;
unlock_cerr_mutex();
++errors_occurred;
return 1;
} /* if (!out_strm.good()) */
#endif /* DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

528.

```

< Scan_Parse_Parameter_Type::get_input definition 520 > +≡
    memset(data_buffer, 0, BUFFER_SIZE);
    write_to_file = true; } /* else */
} /* else */
if (cin.eof()) {
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "cin.eof() == 'true'." << endl << "Breaking." << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    cin.clear();
    break;
}
} /* for */
if (write_to_file) out_strm.close();
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "Exiting 'Scan_Parse_Parameter_Type::get_input' successfully"
        " with return value 0." << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
return 0; } /* End of Scan_Parse_Parameter_Type::get_input definition */

```

529. Mark iRODS objects for deletion. [LDF 2013.08.07.]

Log

[LDF 2013.08.07.] Added this function.

530.

```

⟨ Scan_Parse_Parameter_Type :: mark_irods_objects_for_deletion definition 530 ⟩ ≡
int Scan_Parse_Parameter_Type :: mark_irods_objects_for_deletion(Response_Type &response, char
    *buffer_ptr, size_t buffer_size){ bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    size_t pos;
    stringstream temp_strm;
    string thread_str;
    bool delete_from_archive;
    bool delete_from_database;
    if (thread_ctr > 0) {
        temp_strm << "[Thread" << thread_ctr << "]";
        thread_str = temp_strm.str();
        temp_strm.str("");
    }
    if (response.options & 2U & response.options & 4U) delete_from_database = true;
    if (response.options & 4U) delete_from_archive = false;
    else delete_from_archive = true;
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering 'Scan_Parse_Parameter_Type::mark_irods_objects_for"
            _deletion'." << endl;
        response.show("response:");
        if (response.options & 2U & response.options & 4U) {
            cerr << "'database' option and 'database-only' options are both set." <<
                endl << "This is an error. It will be handled below and this function"
                "will exit unsuccessfully." << endl;
        }
        else if (response.options & 2U) {
            cerr << "'database' option is set." << endl <<
                "Will delete entry/entries for iRODS object(s)" <<
                "from 'gwiridsif.Irods_Objects' database table" << endl <<
                "and AVUs from 'gwiridsif.Irods_AVUs' database table." << endl;
        }
        else if (response.options & 4U) {
            cerr << "'database-only' option is set." << endl <<
                "Will delete entry/entries for iRODS object(s)" <<
                "from 'gwiridsif.Irods_Objects' database table" << endl <<
                "and AVUs from 'gwiridsif.Irods_AVUs' database table." << endl <<
                "Will not delete iRODS object(s) from archive." << endl;
        }
        cerr << "user_id==" << user_id << endl << "irods_env_filename==" << irods_env_filename <<
            endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

See also sections 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, and 541.

This code is used in section 544.

531. Error handling: ‘database’ and ‘database-only’ options both specified. [LDF 2013.08.12.]
 This should never occur, because the case is caught and handled in *yyparse*. [LDF 2013.08.12.]

Log

[LDF 2013.08.12.] Added this section.

```
<Scan_Parse_Parameter_Type::mark_irods_objects_for_deletion definition 530> +≡
  if (response.options & 2U & response.options & 4U) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! " << "In 'Scan_Parse_Parameter_Type::mark_irods_objects_for_deletion':"
    << endl << "'database' and 'datbase-only' options are both set." << endl << "This is not permitted."
    << endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    temp_strm.str("");
    temp_strm << "MARK_IRODS_OBJECT_FOR_DELETION_RESPONSE" << GW_ERROR <<
    " " << response.options << "U" << "\"(no iRODS objects)\\" " << "OUL" <<
    "\Option_error:'database' and 'database-only' options both specified" <<
    "This is not permitted.\\"";
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "temp_strm.str() == " << temp_strm.str() << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  strcpy(buffer_ptr,temp_strm.str().c_str());
  ++errors_occurred;
  return 1;
} /* if (response.options & 2U & response.options & 4U) */
```

532.

```
<Scan_Parse_Parameter_Type::mark_irods_objects_for_deletion definition 530> +≡
  if (response.string_vector.size() == 0) {
    lock_cerr_mutex();
    cerr << thread_str << "WARNING! In 'Scan_Parse_Parameter_Type::mark_irods_objects_for_deletion':"
    << endl << "'response.string_vector' is empty. No iRODS objects to mark for deletion." <<
    endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    ++warnings_occurred;
    temp_strm.str("");
    temp_strm << "MARK_IRODS_OBJECT_FOR_DELETION_RESPONSE" << GW_ERROR <<
    " " << response.options << "U" << "\"(no iRODS objects)\\" " << "OUL" <<
    "\Error: No iRODS objects found to mark for deletion\";
    strcpy(buffer_ptr,temp_strm.str().c_str());
    return 1;
} /* if (response.string_vector.size() == 0) */
```

533.

(Scan_Parse_Parameter_Type :: mark_irods_objects_for_deletion definition 530) +≡ Response_Type new_response;

534.

(Scan_Parse_Parameter_Type :: mark_irods_objects_for_deletion definition 530) +≡

```

string new_flags;
pos = response.flags.find("-f");
if (pos != string::npos) new_flags += "-f";
pos = response.flags.find("-n");
if (pos != string::npos) new_flags += "-n";
pos = response.flags.find("-r");
if (pos != string::npos) new_flags += "-r";
pos = response.flags.find("-U");
if (pos != string::npos) new_flags += "-U";
pos = response.flags.find("-v");
if (pos != string::npos) new_flags += "-v";
pos = response.flags.find("-V");
if (pos != string::npos) new_flags += "-V";
pos = response.flags.find("--empty");
if (pos != string::npos) new_flags += "--empty";
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Scan_Parse_Parameter_Type::mark_irods_objects_for_deletion':"
        endl << "'new_flags' == " << new_flags << endl;
    unlock_cerr_mutex();
}
/* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

535.

```

⟨ Scan_Parse_Parameter_Type :: mark_irods_objects_for_deletion definition 530 ⟩ +≡
vector<Irods_Object_Type> irods_object_vector;
Irods_Object_Type curr_irods_object;
for (vector<string>::const_iterator iter = response.string_vector.begin();
     iter ≠ response.string_vector.end(); ++iter) {
    temp_strm.str("");
    pos = iter->find('/');
    if (pos ≡ string::npos) temp_strm ≪ irods_current_dir ≪ "/";
    temp_strm ≪ *iter;
    curr_irods_object.set(user_id, temp_strm.str());
    status = curr_irods_object.get_from_database(mysql_ptr);
    if (status < 0) {
        lock_cerr_mutex();
        cerr ≪ thread_str ≪ "ERROR! In 'Scan_Parse_Parameter_Type::m\
            ark_irods_objects_for_deletion':"
        ≪ endl ≪ "'Irods_Object_Type::get_from_database' failed, returning"
        ≪ status ≪ "."
        ≪ endl ≪ "Exiting function unsuccessfully with return value 1."
        ≪ endl;
        unlock_cerr_mutex();
        ++errors_occurred;
        temp_strm.str("");
        temp_strm ≪ "MARK_IRODS_OBJECT_FOR_DELETION_RESPONSE"
        ≪ GW_SERVER_SIDE_DATABASE_ERROR ≪ " "
        ≪ response.options ≪ "U"
        ≪ "\"(empty)\""
        ≪ "OUL"
        ≪ "\"Error: Failed to retrieve iRODS object(s) from database\"";
        strcpy(buffer_ptr, temp_strm.str().c_str());
        return 1;
    } /* if (status < 0) */
    else if (status ≡ 0) {
        lock_cerr_mutex();
        cerr ≪ thread_str ≪ "WARNING! In 'Scan_Parse_Parameter_Type:\\
            :mark_irods_objects_for_deletion':"
        ≪ endl ≪ "'Irods_Object_Type::get_from_database' returned 0 rows."
        ≪ endl ≪ "Failed to fetch iRODS object with path"
        ≪ *iter ≪ ","
        ≪ "from 'gwirldsif.Irods_Objects' database table."
        ≪ endl ≪ "Will try to continue."
        ≪ endl;
        unlock_cerr_mutex();
        ++warnings_occurred;
        continue;
    } /* else if (status ≡ 0) */
    else if (status > 1) {
        lock_cerr_mutex();
        cerr ≪ thread_str ≪ "WARNING! In 'Scan_Parse_Parameter_Type::mark_irods_objects_\\
            for_deletion':"
        ≪ endl ≪ "'Irods_Object_Type::get_from_database' returned"
        ≪ status ≪ " rows (> 1)."
        ≪ endl ≪ "This shouldn't be possible."
        ≪ endl ≪ "Will try to continue."
        ≪ endl;
        unlock_cerr_mutex();
        ++warnings_occurred;
        continue;
    } /* else if (status ≡ 0) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {

```

```

lock_cerr_mutex();
cerr << thread_str << "In 'Scan_Parse_Parameter_Type::mark_irods_objects_for_delet\"
ion':" << endl << "'Irods_Object_Type::get_from_database' succeeded, returning\
1." << endl;
unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
if ((!(response.options & 4U) & curr_irods_object.marked_for_deletion_from_archive ==
false) || ((response.options & 2U || response.options & 4U) &
curr_irods_object.marked_for_deletion_from_gwirdsif_db == false)) {
irods_object_vector.push_back(curr_irods_object);
}
else {
lock_cerr_mutex();
cerr << thread_str << "WARNING! In 'Scan_Parse_Parameter_Type::mark_irods_objects_\
for_deletion': " << endl << "iRODS_object" << curr_irods_object.id << " " <<
"" << curr_irods_object.path << ", already marked for deletion." << endl <<
"Not marking again. Continuing." << endl;
unlock_cerr_mutex();
temp_strm.str("");
temp_strm << "MARK_IRODS_OBJECT_FOR_DELETION_RESPONSE" <<
GW_IRODS_OBJECT_ALREADY_MARKED_FOR_DELETION << " " << response.options <<
"U" << "\n" << curr_irods_object.path << "\n";
if (curr_irods_object.delete_from_archive_timestamp > 0 & !(response.options & 2U || response.options &
4U)) temp_strm << curr_irods_object.delete_from_archive_timestamp << "UL";
else if (curr_irods_object.delete_from_gwirdsif_db_timestamp >
0 & (response.options & 2U || response.options & 4U))
temp_strm << curr_irods_object.delete_from_gwirdsif_db_timestamp << "UL";
else if (curr_irods_object.delete_from_archive_timestamp ==
curr_irods_object.delete_from_gwirdsif_db_timestamp)
temp_strm << curr_irods_object.delete_from_archive_timestamp << "UL";
else { /* In this case, the iRODS object is already marked for deletion from both the archive
and the gwirdsif.Irods_Objects database table, but the timestamps differ. There's no way
to decide which timestamp to send, so we just send 0UL. This case shouldn't occur too often,
so I'm not going to bother trying to fix it at present. !! TODO: Do something about this,
maybe. [LDF 2013.08.08.] */
temp_strm << "0UL";
}
temp_strm << "\Error: iRODS_object already marked for deletion\"";
new_response.type = Response_Type::COMMAND_ONLY_TYPE;
new_response.command = temp_strm.str();
response_deque.push_back(new_response);
++warnings_occurred;
} /* else */
curr_irods_object.clear();
} /* for */

```

536.

```
<Scan_Parse_Parameter_Type::mark_irods_objects_for_deletion definition 530> +≡
if (irods_object_vector.size() ≡ 0) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "WARNING! In 'Scan_Parse_Parameter_Type::mark_irods_objects_\
for_deletion': " ≪ endl ≪ "'irods_object_vector.size()' ≡ 0" ≪
    endl ≪ "Failed to retrieve any 'Irods_Object_Type' objects." ≪ endl ≪
    "Exiting function unsuccessfully with return value 2." ≪ endl;
unlock_cerr_mutex();
++warnings_occurred;
return 2;
} /* if (irods_object_vector.size() ≡ 0) */
```

537.

```
<Scan_Parse_Parameter_Type::mark_irods_objects_for_deletion definition 530> +≡
#ifndef DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "In 'Scan_Parse_Parameter_Type::mark_irods_objects_for_delet\
ion': " ≪ endl ≪ "'irods_object_vector.size()' ≡ " ≪ irods_object_vector.size() ≪ endl;
    for (vector<Irods_Object_Type>::iterator iter = irods_object_vector.begin();
        iter ≠ irods_object_vector.end(); ++iter) {
        iter->show();
    } /* for */
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

538. The delay value is stored in *delay*. It's passed to **Irods_Object_Type**::*mark_for_deletion* as a reference. The delay value that's used is *response.delay_value*. [LDF 2013.08.14.]

```

⟨Scan_Parse_Parameter_Type::mark_irods_objects_for_deletion definition 530⟩ +≡
time_t delay = static_cast<time_t>(0);
status = Irods_Object_Type::mark_for_deletion(irods_object_vector, mysql_ptr, response, user_id,
                                              irods_env_filename, delay, thread_str, false);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::mark_irods_objects_for_deletion':"
        << endl << "'Irods_Object_Type::mark_for_deletion' failed, returning" << status <<
        "." << endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    return 1;
} /* if (status ≠ 0) */
#endif /* DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Scan_Parse_Parameter_Type::mark_irods_objects_for_deletion':"
        << endl << "'Irods_Object_Type::mark_for_deletion' succeeded, returning"
        "0." << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

539.

```

⟨Scan_Parse_Parameter_Type::mark_irods_objects_for_deletion definition 530⟩ +≡
new_response.type = Response_Type::COMMAND_ONLY_TYPE;
for (vector<Irods_Object_Type>::iterator iter = irods_object_vector.begin();
     iter ≠ irods_object_vector.end(); ++iter) {
    temp_strm.str("");
    temp_strm << "MARK_IRODS_OBJECT_FOR_DELETION_RESPONSE_0" << response.options << "U"
        << "\n" << iter->path << "\n" << delay << "UL\"Success\"";
    new_response.command = temp_strm.str();
    response_deque.push_back(new_response);
} /* for */

```

540.

```

⟨ Scan_Parse_Parameter_Type :: mark_irods_objects_for_deletion definition 530 ⟩ +≡
if (purge_irods_archive_thread_id == static_cast<pthread_t>(0)) {
    lock_cerr_mutex();
    cerr << thread_str << "NOTICE: In 'Scan_Parse_Parameter_Type::mark_irods_objects_"
        "for_deletion': " << endl << "'purge_irods_archive_thread' == 0: " <<
        endl << "\"Purge_iRODS_archive_thread\" not running." <<
        endl << "iRODS_objects_marked_for_immediate_deletion_will_"
        "be_deleted_the_next_time 'gwiridsif' is started with" <<
        endl << "purging the iRODS_archive enabled." << endl <<
        "Not calling 'pthread_cond_signal'. Continuing." << endl;
    unlock_cerr_mutex();
} /* if */
else {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Scan_Parse_Parameter_Type::mark_irods_objects_for_delet"
            "ion': " << endl << "'purge_irods_archive_thread_id' != 0: " <<
            endl << "\"Purge_iRODS_archive_thread\" running." <<
            "Calling 'pthread_cond_signal' to wake it up." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    pthread_mutex_lock(&purge_irods_archive_mutex);
    status = pthread_cond_signal(&purge_irods_archive_cond);
    if (status != 0) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::mark_irods_objects_f"
            "or_deletion': " << endl << "'pthread_cond_signal' failed, returning" <<
            status << ":" << endl << "Error: " << strerror(status) << endl <<
            "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        return 1;
    }
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Scan_Parse_Parameter_Type::mark_irods_objects_for_delet"
            "ion': " << endl << "'pthread_cond_signal' succeeded, returning 0." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    pthread_mutex_unlock(&purge_irods_archive_mutex);
} /* else */

```

541.

```
<Scan_Parse_Parameter_Type::mark_irods_objects_for_deletion definition 530> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Exiting `Scan_Parse_Parameter_Type::mark_irods_objects_for_"
            "deletion'" << "successfully with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0;
/* End of Scan_Parse_Parameter_Type::mark_irods_objects_for_deletion definition */
```

542.

```
<Garbage 303> +≡
```

543. Putting spptfnc1.web together.

544. This is what's compiled.

```
<Include files 3>
using namespace std;
using namespace gwrdifpk;
<External function declarations 32>
<Scan_Parse_Parameter_Type::get_handle definition 327>
<Scan_Parse_Parameter_Type::get_metadata definition 341>
<Scan_Parse_Parameter_Type::add_metadata definition 384>
<Scan_Parse_Parameter_Type::parse_metadata definition 418>
<Scan_Parse_Parameter_Type::fetch_handle_from_database definition 429>
<Scan_Parse_Parameter_Type::fetch_handles_from_database definition 441>
<Scan_Parse_Parameter_Type::store_dc_metadata definition 444>
<Scan_Parse_Parameter_Type::show_certificates definition 478>
<Scan_Parse_Parameter_Type::get_privileges definition 490>
<Scan_Parse_Parameter_Type::set_user_info definition 511>
<Scan_Parse_Parameter_Type::show_privileges definition 512>
<Scan_Parse_Parameter_Type::get_highest_value definition 513>
<Scan_Parse_Parameter_Type::get_input definition 520>
<Scan_Parse_Parameter_Type::mark_irods_objects_for_deletion definition 530>
#ifndef 0
<Garbage 303>
#endif
```

545. This is what's written to the header file spptfnc1.h. [LDF 2013.02.13.]

```
<spptfnc1.h 545> ≡
#ifndef SPPTFNC1_H
#define SPPTFNC1_H 1 /* Empty */
#endif
```

546. Scan_Parse_Parameter_Type function definitions 2.

547. Include files.

```
<Include files 3> +≡
#include <stdlib.h>      /* Standard Library for C */
#include <stdio.h>
#include <sys/mman.h>
#include <errno.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <limits.h>
#include <string.h>
#include <cctype.h>
#include <algorithm>      /* Standard Template Library (STL) for C++ */
#include <bitset>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <map>
#include <string>
#include <time.h>
#include <math.h>
#include <sstream>
#include <vector>
#include <deque>
#include <stack>
#include <set>
#include <pthread.h>      /* POSIX threads */
#include <gcrypt.h>        /* for gcry_control */
#include <gnutls/gnutls.h>
#include <gnutls/x509.h>
#include <mysql.h>
#include <expat.h>
#if HAVE_CONFIG_H
#include <config.h>
#endif
#undef NAME_LEN
#undef LOCAL_HOST
#include "rspercds.h++"
#include "glblcnst.h++"
#include "glblvrbl.h++"
#include "excptntp.h++"
#include "utilfncts.h++"
#include "grouptp.h++"
#include "hdlvltp.h++"
#include "irdsavtp.h++"
#include "helper.h++"
#include "tanfncts.h++"
#include "pidfncts.h++"
#include "parser.h++"
#include "scanner.h++"
#include "rspnstp.h++"
```

```
#include "irdsobtp.h++"
#include "hndltype.h++"
#include "dcmtdtpp.h++"
#include "x509cert.h++"
#include "dstngnmt.h++"
#include "scprpmtp.h++"
#include "usrinftp.h++"
```

548.

(External function declarations 32) +≡
int yyparse(yyscan_t parameter);

549. Scanner_Type functions 2. [LDF 2013.08.16.]**550. Undelete files (*undelete_files*).** [LDF 2013.08.16.]**Log**

[LDF 2013.08.16.] Added this function.

```
(Scan_Parse_Parameter_Type::undelete_files definition 550) ≡
int Scan_Parse_Parameter_Type::undelete_files(Response_Type &response, string thread_str){
    bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status = 0;
    Response_Type new_response;
    stringstream temp_strm;
    string filename_str;
#if DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering 'Scan_Parse_Parameter_Type::undelete_files'." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    vector<Response_Type> response_vector;
    bool database = false;
    bool archive = true;
    unsigned int option_ctr = 0U;
    if (response.options & 1U) {
        option_ctr |= 1U;
        database = true;
    }
    if (response.options & 2U) {
        option_ctr |= 2U;
        database = true;
        archive = false;
    }
}
```

See also sections 551, 552, 553, 554, 555, 556, and 557.

This code is used in section 560.

551.

```
⟨ Scan_Parse_Parameter_Type :: undelete_files definition 550 ⟩ +≡
  if (response.string_vector.size() == 0) {
    new_response.type = Response_Type::COMMAND_ONLY_TYPE;
    temp_strm << "UNDELETE" << RESPONSE_1 << option_ctr << "U" <<
    "\\" << "No" << iRODS_objects_specified "";
    new_response.command = temp_strm.str();
    response_deque.push_back(new_response);
    return 1;
} /* if (response.string_vector.size() == 0) */
```

552.

```

⟨ Scan_Parse_Parameter_Type :: undelete_files definition 550 ⟩ +≡
size_t pos;
vector<Irods_Object_Type> irods_object_vector;
Irods_Object_Type curr_irods_object;
bool set_ellipsis = false; for (vector<string>::iterator iter = response.string_vector.begin();
    iter ≠ response.string_vector.end(); ++iter) { curr_irods_object.clear();
pos = iter->find("/");
if (pos ≡ string::npos) {
    iter->insert(0, "/");
    iter->insert(0, irods_current_dir);
}
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Scan_Parse_Parameter_Type::undelete_files':"
        << endl <<
        "'*iter' == " << *iter << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
if (filename_str.empty()) filename_str = *iter;
else if (set_ellipsis ≡ false) {
    filename_str += "...";
    set_ellipsis = true;
}
curr_irods_object.clear();
status = curr_irods_object.set(user_id, *iter);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::undelete_files':"
        << endl <<
        "'Irods_Object_Type::set' failed, returning " << status << "."
        << endl <<
        "Will try to continue." << endl;
    unlock_cerr_mutex();
new_response.type = Response_Type::COMMAND_ONLY_TYPE;
temp_strm << "UNDELETE_RESPONSE_2" << option_ctr << "U\" " << *iter << "\\" "
        << "\Failed_to_set_iRODS_object\"";
new_response.command = temp_strm.str();
response_deque.push_back(new_response);
++errors_occurred;
continue;
} /* if (status ≠ 0) */
#ifndef DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Scan_Parse_Parameter_Type::undelete_files':"
        << endl <<
        "'Irods_Object_Type::set' succeeded, returning 0." << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

553.

```

<Scan_Parse_Parameter_Type::undelete_files definition 550> +≡
    status = curr_irods_object.get_from_database(mysql_ptr, false);
    if (status ≤ 0) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::undelete_files':"
            << endl <<
            "'Irods_Object_Type::get_from_database' failed, returning "
            << status << "."
            << endl <<
            "Will try to continue."
            << endl;
        unlock_cerr_mutex();
        new_response.type = Response_Type::COMMAND_ONLY_TYPE;
        temp_strm << "UNDELETE_RESPONSE" << option_ctr << "U\""
            << *iter << "\""
            << "\Failed to retrieve iRODS object from 'gwiridsif' database\"";
        new_response.command = temp_strm.str();
        response_deque.push_back(new_response);
        ++errors_occurred;
        continue;
    } /* if (status ≠ 0) */
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_str << "In 'Scan_Parse_Parameter_Type::undelete_files':"
                << endl <<
                "'Irods_Object_Type::set' succeeded, returning 0."
                << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
        irods_object_vector.push_back(curr_irods_object); } /* for */

```

554.

```

<Scan_Parse_Parameter_Type::undelete_files definition 550> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Scan_Parse_Parameter_Type::undelete_files':"
            << endl <<
            "'irods_object_vector.size()' == "
            << irods_object_vector.size() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

555.

```
<Scan_Parse_Parameter_Type::undelete_files definition 550> +≡
  if (irods_object_vector.size() ≡ 0) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "ERROR! In 'Scan_Parse_Parameter_Type::undelete_files':"
    endl ≪ "'irods_object_vector.size()' == 0." ≪ endl ≪
    "No iRODS objects retrieved from 'gwiridsif' database." ≪ endl ≪
    "Exiting function unsuccessfully with return value 1." ≪ endl;
    unlock_cerr_mutex();
    new_response.type = Response_Type::COMMAND_ONLY_TYPE;
    temp_strm ≪ "UNDELETE_RESPONSE_1" ≪ option_ctr ≪ "U\""
    ≪ filename_str ≪ "\" "
    ≪ "\\"No iRODS objects retrieved from 'gwiridsif' database\"";
    new_response.command = temp_strm.str();
    response_deque.push_back(new_response);
    ++errors_occurred;
  }
  /* if (irods_object_vector.size() ≡ 0) */
```

556.

```

⟨ Scan_Parse_Parameter_Type::undelete_files definition 550 ⟩ +≡
status = Irods_Object_Type::undelete_irods_objects(irods_object_vector, mysql_ptr, archive, database,
    response_vector, irods_env_filename, thread_str);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::undelete_files':"
        endl << 'Irods_Object_Type::undelete_irods_objects' failed, returning "
        status << "." << endl << "Failed to undelete iRODS objects." << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
unlock_cerr_mutex();
new_response.type = Response_Type::COMMAND_ONLY_TYPE;
temp_strm << "UNDELETE_RESPONSE_1" << option_ctr << "U" << "\n" << filename_str <<
    "\\" << "Failed to undelete iRODS objects\"";
new_response.command = temp_strm.str();
response_deque.push_back(new_response);
++errors_occurred;
return 1;
}
else {
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Scan_Parse_Parameter_Type::undelete_files':"
        << endl <<
        "'Irods_Object_Type::undelete_irods_objects' succeeded, returning 0."
        << endl <<
        "Undeleted iRODS objects successfully." << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
new_response.type = Response_Type::COMMAND_ONLY_TYPE;
temp_strm << "UNDELETE_RESPONSE_0" << option_ctr << "U" << "\n" << filename_str <<
    "\\" << "Undeleted iRODS objects successfully\"";
new_response.command = temp_strm.str();
response_deque.push_back(new_response);
} /* else */

```

557.

```

⟨ Scan_Parse_Parameter_Type::undelete_files definition 550 ⟩ +≡
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "Exiting 'Scan_Parse_Parameter_Type::undelete_files' "
        "successfully with return value 0." << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
return 0; } /* End of Scan_Parse_Parameter_Type::undelete_files definition */

```

558.

⟨ Garbage 303 ⟩ +≡

559. Putting `spptfnc2.web` together.

560. This is what's compiled.

```
<Include files 3>
using namespace std;
using namespace gwrdifpk;
<External function declarations 32>
<Scan_Parse_Parameter_Type::undelete_files definition 550>
#if 0
    <Garbage 303>
#endif
```

561. This is what's written to the header file `spptfnc2.h`. [LDF 2013.02.13.]

```
<spptfnc2.h 561>≡
#ifndef SPPTFNC2_H
#define SPPTFNC2_H 1     /* Empty */
#endif
```

562. `Scan_Parse_Parameter_Type` server action function definitions.

563. Include files.

```
<Include files 3> +≡
#include <stdlib.h>      /* Standard Library for C */
#include <stdio.h>
#include <sys/mman.h>
#include <errno.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <limits.h>
#include <string.h>
#include <algorithm>      /* Standard Template Library (STL) for C++ */
#include <bitset>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <map>
#include <string>
#include <time.h>
#include <math.h>
#include <sstream>
#include <vector>
#include <deque>
#include <stack>
#include <set>
#include <pthread.h>      /* POSIX threads */
#include <gcrypt.h>        /* for gcry-control */
#include <gnutls/gnutls.h>
#include <gnutls/x509.h>
#include <mysql.h>
#include <expat.h>
#if HAVE_CONFIG_H
#include <config.h>
#endif
#undef NAME_LEN
#undef LOCAL_HOST
#include "rspercds.h++"
#include "glblcnst.h++"
#include "glblvrbl.h++"
#include "excptntp.h++"
#include "utilfncs.h++"
#include "grouptp.h++"
#include "hndlvltp.h++"
#include "irdsavtp.h++"
#include "helper.h++"
#include "tanfncts.h++"
#include "pidfncts.h++"
#include "parser.h++"
#include "scanner.h++"
#include "rspnstp.h++"
#include "irdsobtp.h++"
```

```
#include "hdltype.h++"
#include "dcmtddtp.h++"
#include "x509cert.h++"
#include "dstngnmt.h++"
#include "scprpmtp.h++"
#include "usrinftp.h++"
#include "prsrfncts.h++"
```

564. Server action functions. [LDF 2013.05.27.]

Log

[LDF 2013.05.27.] Added this section. The declarations are in scprpmtp.web.

565.

Log

[LDF 2013.05.29.] Added this function.

[LDF 2013.05.16.] BUG FIX: Added code for issuing a warning if *response.command* is empty. I've been caught by this a couple of times.

[LDF 2013.05.29.] This entry is from a time when the code in this function was part of a conditional in *exchange_data_with_client* in *exchncli.web*. [LDF 2013.05.29.]

```
( Scan_Parse_Parameter_Type::server_action_command_only definition 565 ) ≡
int Scan_Parse_Parameter_Type::server_action_command_only(Response_Type &response){ bool
    DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    char buffer[BUFFER_SIZE];
    char *buffer_ptr = buffer;
    memset(buffer, 0, BUFFER_SIZE);
    stringstream temp_strm;
    string thread_str;
    temp_strm << "[Thread" << thread_ctr << "] ";
    thread_str = temp_strm.str();
    temp_strm.str("");
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering 'Scan_Parse_Parameter_Type::server_action_command_only':"
            << endl << "'response.type' == "
            Response_Type::typename_map[response.type] << "(" << response.type << ")" << endl <<
            "'response.command' == "
            << response.command << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    if (response.command.empty()) {
        lock_cerr_mutex();
        cerr << thread_str << "WARNING! "
            << "In 'Scan_Parse_Parameter_Type::server_action_command_only':"
            << "'response.command' is empty." << endl <<
            "Sending a NULL byte to client, so the latter won't block, "
            << "but this is almost certainly a programming error." << endl;
        unlock_cerr_mutex();
        ++warnings_occurred;
        status = send_to_peer(0, 1);
    } /* if (response.command.empty()) */
}
```

See also sections 566 and 567.

This code is used in section 698.

566.

```
< Scan_Parse_Parameter_Type::server_action_command_only definition 565 > +≡
else {
    strcpy(buffer, response.command.c_str());
    buffer_ptr = buffer;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_command_only' :"
            << "buffer" <=> endl << buffer << endl << "Sending to client." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    status = send_to_peer(&buffer_ptr, 0);
} /* else */
```

567.

```
< Scan_Parse_Parameter_Type::server_action_command_only definition 565 > +≡
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::server_action_command_only' :"
        << "'Scan_Parse_Parameter_Type::send_to_peer' failed, returning"
        << status << "." << endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    return 1;
} /* if (status ≠ 0) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_command_only' :"
            << endl << "'Scan_Parse_Parameter_Type::send_to_peer' succeeded, returning 0." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Exiting 'Scan_Parse_Parameter_Type::server_action_command_only' "
            << "successfully" << "with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; } /* End of Scan_Parse_Parameter_Type::server_action_command_only definition */
```

568. *server_action_send_file.* [LDF 2013.05.29.]

Log

[LDF 2012.11.20.] In the case that *response.command.empty()* \equiv *false*, now calling **Scan_Parse_Parameter_Type**::*server_action_send_file* only once, passing both *&buffer_ptr* and *response.local_filename* to it as arguments. Previously, the latter function would only accept one of the two.

[LDF 2013.05.29.] Added this function. Previously, the code it contains was in a conditional in *exchange_data_with_client*.

```
⟨ Scan_Parse_Parameter_Type::server_action_send_file definition 568 ⟩ ≡
int Scan_Parse_Parameter_Type::server_action_send_file(Response_Type &response){ bool
    DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    char buffer[BUFFER_SIZE];
    char *buffer_ptr = buffer;
    memset(buffer, 0, BUFFER_SIZE);
    stringstream temp_strm;
    string thread_str;
    temp_strm << "[Thread" << thread_ctr << "]";
    thread_str = temp_strm.str();
    temp_strm.str("");
#define DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering 'Scan_Parse_Parameter_Type::se\
rver_action_send_file':" << endl << "'response.type' == " <<
        Response_Type::typename_map[response.type] << "(" << response.type << ")" << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
See also sections 569 and 570.
This code is used in section 698.
```

569.

```
<Scan_Parse_Parameter_Type::server_action_send_file definition 568> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_send_file':" << endl;
        response.show("response:");
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    if (response.command.empty()) {
        status = send_to_peer(0, 0, response.local_filename);
    }
    else {
        memset(buffer, 0, BUFFER_SIZE);
        strcpy(buffer, response.command.c_str());
        buffer_ptr = buffer;
        status = send_to_peer(&buffer_ptr, 0, response.local_filename);
    }
    if (status ≠ 0) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::server_action_send_file':" <<
            endl << "'Scan_Parse_Parameter_Type::send_to_peer' failed, returning" << status <<
            "." << endl << "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        ++errors_occurred;
    }
    return 1;
} /* if (status ≠ 0) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_send_file':" << endl <<
            "'Scan_Parse_Parameter_Type::send_to_peer' succeeded, returning 0." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

570.

```
<Scan_Parse_Parameter_Type::server_action_send_file definition 568> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Exiting 'Scan_Parse_Parameter_Type::server_action_send_file'\n
            ' successfully'" << "with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; } /* End of Scan_Parse_Parameter_Type::server_action_send_file definition */
```

571. *server_action_receive_put_file*. [LDF 2013.05.29.]

Log

[LDF 2013.05.29.] Added this function.

```
{ Scan_Parse_Parameter_Type :: server_action_receive_put_file definition 571 } ≡
int Scan_Parse_Parameter_Type :: server_action_receive_put_file(Response_Type &response)
{
    return 0;
}
```

This code is used in section 698.

572. *server_action_receive_metadata_file*. [LDF 2013.05.29.]

Log

[LDF 2013.05.29.] Added this function.

```
{ Scan_Parse_Parameter_Type :: server_action_receive_metadata_file definition 572 } ≡
int Scan_Parse_Parameter_Type :: server_action_receive_metadata_file(Response_Type &response)
{
    return 0;
}
```

This code is used in section 698.

573. *server_action_send_handle*. [LDF 2013.05.29.]

Log

[LDF 2013.05.29.] Added this function.

```
< Scan_Parse_Parameter_Type :: server_action_send_handle definition 573 > ≡
int Scan_Parse_Parameter_Type :: server_action_send_handle(Response_Type &response){ bool
    DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    char buffer[BUFFER_SIZE];
    char *buffer_ptr = buffer;
    memset(buffer, 0, BUFFER_SIZE);
    char temp_buffer[256];
    memset(temp_buffer, 0, 256);
    stringstream temp_strm;
    ofstream out_strm;
    string thread_str;
    temp_strm << "[Thread" << thread_ctr << "] ";
    thread_str = temp_strm.str();
    temp_strm.str("");
    int fd = 0;
    char temp_filename[] = "/tmp/gwirdsif.XXXXXX";
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering 'Scan_Parse_Parameter_Type::se\
rver_action_send_handle': " << endl << "'response.type' == " <<
        Response_Type :: typename_map[response.type] << "(" << response.type << ")" << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

See also sections 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, and 591.

This code is used in section 698.

574.

```

⟨ Scan_Parse_Parameter_Type :: server_action_send_handle definition 573 ⟩ +≡
  if (response.handle == 0) {
    lock_cerr_mutex();
    cerr << thread_str << "WARNING! In 'Scan_Parse_Parameter_Type::server_action_send\
      _handle': " << endl << "'response.handle' is NULL. Not sending handle to client." <<
      endl;
    unlock_cerr_mutex();
    ++warnings_occurred;
    strcpy(buffer, "GET_HANDLE_RESPONSE_6");
    status = send_to_peer(&buffer_ptr, 0);
    if (status != 0) {
      lock_cerr_mutex();
      cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::\
        server_action_send_handle': " << endl << "'Scan_Parse_Parameter\
          _Type::send_to_peer' failed, returning " << status << "." << endl <<
          "Exiting function unsuccessfully with return value 1." << endl;
      unlock_cerr_mutex();
      return 1;
    } /* if (status != 0) */
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_send_handle': " <<
        endl << "'Scan_Parse_Parameter_Type::send_to_peer' succeeded, returning 0." <<
        endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  return 2;
} /* if (response.handle == 0) */
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_send_handle': " <<
      endl << "Sending handle to client." << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  unsigned char temp_val;
  string hexl_str;
  bool write_to_file = false;
  bool force_write = false; /* true */
  memset(buffer, 0, BUFFER_SIZE); for (map<unsigned long int,
    Handle_Value_Type>::const_iterator iter = response.handle->handle_value_map.begin();
    iter != response.handle->handle_value_map.end(); ++iter) { memset(temp_buffer, 0, 256);
  temp_strm.str(""); temp_strm << "GET_HANDLE_RESPONSE_0_HANDLE_" << "\\" << iter->second.handle << "\\";
  if (!iter->second.filename.empty()) temp_strm << "\FILE\\" << iter->second.filename << "\\";

```

```
temp_strm << "IDX" << iter->second.idx << "TYPE\" << iter->second.type << "\" <<
"DATA_LENGTH" << iter->second.data_length << "U";
```

575.

```
( Scan_Parse_Parameter_Type :: server_action_send_handle definition 573 ) +≡
  if (iter->second.data & iter->second.data_length > 0 & (iter->second.type ≡ "EMAIL" ∨ iter->second.type ≡
    "DESC" ∨ iter->second.type ≡ "IRODS_OBJECT" ∨ iter->second.type ≡ "IRODS_OBJECT_REF" ∨
    iter->second.type ≡ "IRODS_OBJECT_PID" ∨ iter->second.type ≡ "DC_METADATA" ∨ iter->second.type ≡
    "DC_METADATA_PID" ∨ iter->second.type ≡ "DC_METADATA_IRODS_OBJECT" ∨ iter->second.type ≡
    "DC_METADATA_IRODS_OBJECT_PID" ∨ iter->second.type ≡ "DC_METADATA_IRODS_OBJECT_REF")) {
  memcpy(temp_buffer, iter->second.data, iter->second.data_length);
  temp_strm << "DATA\" << temp_buffer << "\";
```

}

576.

Log

[LDF 2013.04.26.] Added code for handling binary data.

```
( Scan_Parse_Parameter_Type :: server_action_send_handle definition 573 ) +≡
  else if (iter->second.data & iter->second.data_length > 0) /* binary data */
  {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'Scan_Parse_Parameter_Type::server_action_send_handle': Binary data == " <<
      endl;
    fwrite(iter->second.data, 1, iter->second.data_length, stderr);
    cerr << endl;
    for (int i = 0; i < iter->second.data_length; ++i) {
      cerr << "iter->second.data[" << i << "==" << iter->second.data[i] << endl;
      temp_val = static_cast<unsigned char>(iter->second.data[i]);
      cerr << "temp_val == " << static_cast<unsigned int>(temp_val) << endl;
      cerr << "isprint(iter->second.data[" << i << "]) == " <<
        static_cast<bool>(isprint(iter->second.data[i])) << endl;
    } /* for */
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  hexl_str = "";
  status = hexl_encode(iter->second.data, iter->second.data_length, hexl_str, #1e);
  if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << thread_str << "WARNING! In 'Scan_Parse_Parameter_Type::server_action_send\
      _handle': " << endl << "'hexl_encode' failed, returning " << status << "." << endl <<
      "Failed to encode data from 'data' field of handle" << "in hexadecimal." << endl <<
      "Not sending encoded contents of 'data' field to client." << endl << "Continuing." <<
      endl;
    unlock_cerr_mutex();
    ++warnings_occurred;
    temp_strm << "DATA\" (binary) \"";
  } /* if (status ≠ 0) */
}
```

577.

```

⟨ Scan_Parse_Parameter_Type :: server_action_send_handle definition 573 ⟩ +≡
else      /* status ≡ 0, hexl_encode succeeded */
{
#if DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In\u0027Scan_Parse_Parameter_Type::server_action_send_handle':\u0027" <<
            endl << '\u0027hexl_encode'\u0027\u0027succeeded,\u0027\u0027returning\u00270." << endl << '\u0027hexl_str'\u0027\u0027==\u0027" <<
            hexl_str << endl;
        unlock_cerr_mutex();
    }      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */
    temp_strm << "DATA_HEXL_ENCODED" << hexl_str << "\u0027";
}      /* else (status ≡ 0, hexl_encode succeeded) */
}      /* else (binary data) */

```

578.

```

⟨ Scan_Parse_Parameter_Type :: server_action_send_handle definition 573 ⟩ +≡
else
if (iter->second.data ≡ 0 ∧ iter->second.data_length > 0) {
    lock_cerr_mutex();
    cerr << thread_str << "WARNING! In\u0027Scan_Parse_Parameter_Type::server_action_send\
        _handle':\u0027" << endl << "'iter->second.data'\u0027\u0027==\u00270\u0027\u0027and\u0027'iter->sec\
        ond.data_length'\u0027\u0027==\u00270" << iter->second.data_length << "\u0027(>\u00270)\u0027." << endl <<
        "This\u0027shouldn't\u0027happen.\u0027\u0027Not\u0027sending\u0027contents\u0027of\u0027'data'\u0027field\u0027" <<
        "\u0027of\u0027handle\u0027value\u0027to\u0027client." << endl << "Continuing." << endl;
    unlock_cerr_mutex();
    ++warnings_occurred;
}      /* else if */

```

579.

```

⟨ Scan_Parse_Parameter_Type :: server_action_send_handle definition 573 ⟩ +≡
else
if (iter->second.data ≠ 0 ∧ iter->second.data_length ≡ 0) {
    lock_cerr_mutex();
    cerr << thread_str << "WARNING! In\u0027Scan_Parse_Parameter_Type\
        ::server_action_send_handle':\u0027" << endl <<
        "'iter->second.data'\u0027\u0027!=\u00270\u0027\u0027and\u0027'iter->second.data_length'\u0027\u0027==\u00270." <<
        endl << "This\u0027shouldn't\u0027happen.\u0027\u0027Not\u0027sending\u0027contents\u0027of\u0027'data'\u0027field\u0027" <<
        "\u0027of\u0027handle\u0027value\u0027to\u0027client." << endl << "Continuing." << endl;
    unlock_cerr_mutex();
    ++warnings_occurred;
}      /* else if */

```

580.

```
<Scan_Parse_Parameter_Type::server_action_send_handle definition 573> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG & iter->second.data == 0) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_send_handle':"
        endl << "'iter->second.data'==0." << endl <<
            "No data in 'data' field of handle value to send to client." << endl;
        unlock_cerr_mutex();
    }
    if (DEBUG & iter->second.data_length == 0) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_send_handle':"
        endl << "'iter->second.data_length'==0." << endl;
        unlock_cerr_mutex();
    }
#endif /* DEBUG_COMPILE */
```

581.

```
<Scan_Parse_Parameter_Type::server_action_send_handle definition 573> +≡
temp_strm << "TTL_TYPE" << iter->second.ttl_type << " "
<< "TTL" << iter->second.ttl << " "
<< "TIMESTAMP" << iter->second.timestamp << "U"
<< "REFS_LENGTH" << iter->second.refs_length <<
" ";
```

582. refs. [LDF 2013.04.28.]

The contents of the **refs** field of the handle value is always treated as binary data. [LDF 2013.04.28.]

Log

[LDF 2013.04.28.] Added this section.

```
<Scan_Parse_Parameter_Type::server_action_send_handle definition 573> +≡
    if (iter->second.refs & iter->second.refs_length > 0) {
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "In 'Scan_Parse_Parameter_Type::server_action_send_handle':"
            << "'refs_length'=="
            << iter->second.refs_length << "(>0)" << endl <<
                "'refs' is always treated as binary data. Will send"
                << "to client." << endl <<
                "'refs'=="
            fwrite(iter->second.refs, 1, iter->second.refs_length, stderr);
            cerr << endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

583.

```

⟨ Scan_Parse_Parameter_Type :: server_action_send_handle definition 573 ⟩ +≡
hexl_str = "";
status = hexl_encode(iter->second.refs, iter->second.refs_length, hexl_str, #1e);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << thread_str << "WARNING! In 'Scan_Parse_Parameter_Type::server_action_send\
_Handle': " << endl << "'hexl_encode' failed, returning " << status << ". " << endl <<
    "Failed to encode data from 'refs' field of handle " << "in hexadecimal." << endl <<
    "Not sending encoded contents of 'refs' field to client." << endl << "Continuing." <<
    endl;
    unlock_cerr_mutex();
    ++warnings_occurred;
    temp_strm << "REFS "(binary)" ;
} /* if (status ≠ 0) */

```

584.

```

⟨ Scan_Parse_Parameter_Type :: server_action_send_handle definition 573 ⟩ +≡
else /* status ≡ 0, hexl_encode succeeded */
{
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_send_handle': " <<
            endl << "'hexl_encode' succeeded, returning 0. " << endl << "'hexl_str' == " <<
            hexl_str << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    temp_strm << "REFS_HEXL_ENCODED" << hexl_str << " ";
} /* else (status ≡ 0, hexl_encode succeeded) */
} /* if (iter->second.refs ∧ iter->second.refs_length > 0) */

```

585.

```

⟨ Scan_Parse_Parameter_Type :: server_action_send_handle definition 573 ⟩ +≡
else
    if (iter->second.refs ≡ 0 ∧ iter->second.refs_length > 0) {
        lock_cerr_mutex();
        cerr << thread_str << "WARNING! In 'Scan_Parse_Parameter_Type::server_action_send\
_Handle': " << endl << "'iter->second.refs' == 0 and 'iter->sec\
ond.refs_length' == " << iter->second.refs_length << " (> 0)." << endl <<
        "This shouldn't happen. Not sending contents of 'refs' field " <<
        "of handle value to client." << endl << "Continuing." << endl;
        unlock_cerr_mutex();
        ++warnings_occurred;
    } /* else if */

```

586.

```
< Scan_Parse_Parameter_Type :: server_action_send_handle definition 573 > +≡
else
  if (iter->second.refs ≠ 0 ∧ iter->second.refs_length ≡ 0) {
    lock_cerr_mutex();
    cerr << thread_str << "WARNING! In 'Scan_Parse_Parameter_Type' \
      ::server_action_send_handle': " << endl <<
      "'iter->second.refs' != 0 and 'iter->second.refs_length' == 0. " <<
      endl << "This shouldn't happen. Not sending contents of 'refs' field" <<
      "of handle value to client." << endl << "Continuing." << endl;
    unlock_cerr_mutex();
    ++warnings_occurred;
  } /* else if */
```

587.

```
< Scan_Parse_Parameter_Type :: server_action_send_handle definition 573 > +≡
#ifndef DEBUG_COMPILE
  if (DEBUG ∧ iter->second.refs ≡ 0) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_send_handle': " <<
      endl << "'iter->second.refs' == 0. " << endl <<
      "No data in 'refs' field of handle value to send to client." << endl;
    unlock_cerr_mutex();
  }
  if (DEBUG ∧ iter->second.refs_length ≡ 0) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_send_handle': " <<
      endl << "'iter->second.refs_length' == 0. " << endl;
    unlock_cerr_mutex();
  }
#endif /* DEBUG_COMPILE */
```

588.

```
< Scan_Parse_Parameter_Type :: server_action_send_handle definition 573 > +≡
temp_strm << "ADMIN_READ" << iter->second.admin_read << " " << "ADMIN_WRITE" <<
  iter->second.admin_write << " " << "PUB_READ" << iter->second.pub_read << " " << "PUB_WRITE" <<
  iter->second.pub_write << " " << "HANDLE_ID" << iter->second.handle_id << "UL" <<
  "HANDLE_VALUE_ID" << iter->second.handle_value_id << "UL" << "IRODS_OBJECT_ID" <<
  iter->second.irods_object_id << "UL" << "CREATED" << iter->second.created << "U" <<
  "LAST_MODIFIED" << iter->second.last_modified << "U" << "DELETE_FROM_DATABASE_TIMESTAMP" <<
  iter->second.delete_from_database_timestamp << "U" << "CREATED_BY_USER" <<
  iter->second.created_by_user_id << "U" << "\\" << iter->second.created_by_user_name << "\\" <<
  "MARKED_FOR_DELETION" << iter->second.marked_for_deletion << endl << endl;
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "temp_strm.str() == " << temp_strm.str() << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

589.

Log

[LDF 2013.05.29.] Added this section.

```

⟨ Scan_Parse_Parameter_Type::server_action_send_handle definition 573 ⟩ +≡
if (force_write ∨ (write_to_file ≡ false ∧ (strlen(buffer) + temp_strm.str().size()) ≥ BUFFER_SIZE)) {
    write_to_file = true;
    force_write = false;
    errno = 0;
    fd = mkstemp(temp_filename);
    if (fd ≡ -1) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::server_action_send_h\
andle': " << endl << "'mkstemp' failed, returning -1: " << endl << strerror(errno) <<
        endl << "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        return 1;
    } /* if (fd ≡ -1) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_send_handle':" <<
            endl << "'mkstemp' succeeded." << endl << "'temp_filename' == " << temp_filename <<
            endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    temp_file_vector.push_back(temp_filename);
    close(fd);
    out_strm.open(temp_filename);
    out_strm << buffer;
    memset(buffer, 0, BUFFER_SIZE);
} /* if */

```

590.

```

⟨ Scan_Parse_Parameter_Type::server_action_send_handle definition 573 ⟩ +≡
    if (write_to_file == true) {
#if 0
        cerr << "write_to_file==" << write_to_file << endl;
        cerr << "temp_filename==" << temp_filename << endl;
#endif
        out_strm << temp_strm.str();
    } /* if (write_to_file == true) */
    else /* write_to_file == false */
    {
        strcat(buffer, temp_strm.str().c_str());
    }
} /* for */
out_strm.close();
if (write_to_file) status = send_to_peer(0, 0, string(temp_filename));
else status = send_to_peer(&buffer_ptr, 0);
if (status != 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::server_action_send_h\
andle'" << endl << "'Scan_Parse_Parameter_Type::send_to_peer' failed, returning" <<
        status << "." << endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    return 1;
} /* if (status != 0) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_send_handle':" <<
            endl << "'Scan_Parse_Parameter_Type::send_to_peer' succeeded, returning 0." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    delete response.handle;
    response.handle = 0;

```

591.

```

⟨ Scan_Parse_Parameter_Type::server_action_send_handle definition 573 ⟩ +≡
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Exiting 'Scan_Parse_Parameter_Type::server_action_send_hand\
le'" << "successfully" << "with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; } /* End of Scan_Parse_Parameter_Type::server_action_send_handle definition */

```

592.

Log

[LDF 2013.05.29.] Added this function.

[LDF 2013.05.29.] This entry is from a time when the code in this function was part of a conditional in *exchange_data_with_client* in **exchncli.web**. [LDF 2013.05.29.]

```
< Scan_Parse_Parameter_Type::server_action_ls definition 592 > ≡
int Scan_Parse_Parameter_Type::server_action_ls(Response_Type &response){ bool
    DEBUG = false;      /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    char buffer[BUFFER_SIZE];
    char *buffer_ptr = buffer;
    memset(buffer, 0, BUFFER_SIZE);
    stringstream temp_strm;
    string temp_filename;
    string thread_str;
    temp_strm << "[Thread]" << thread_ctr << "]";
    thread_str = temp_strm.str();
    temp_strm.str("");
#define DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering 'Scan_Parse_Parameter_Type::server_action_ls':"
        endl << "'response.type' == " << Response_Type::typename_map[response.type] <<
            "(" << response.type << ")" << endl;
        unlock_cerr_mutex();
    }      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */
```

See also sections 593 and 594.

This code is used in section 698.

593.

```

⟨ Scan_Parse_Parameter_Type::server_action_ls definition 592 ⟩ +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "response.string_vector.size() == " << response.string_vector.size() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    status = ls(buffer_ptr, BUFFER_SIZE, &temp_filename, &response);
    if (status != 0) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::server_action_ls':"
            << endl <<
            "'Scan_Parse_Parameter_Type::ls' failed, returning " << status << "."
            << "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        ++errors_occurred;
        return 1;
    } /* if (status != 0) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_ls':"
            << endl <<
            "'Scan_Parse_Parameter_Type::ls' succeeded, returning 0." << endl;
        cerr << "buffer_ptr == " << buffer_ptr << endl << "temp_filename == "
            << temp_filename << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    status = send_to_peer(&buffer_ptr, 0, temp_filename);
    if (status != 0) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::server_action_ls':"
            << endl <<
            "'Scan_Parse_Parameter_Type::send_to_peer' failed, returning " << status << "."
            << endl << "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        ++errors_occurred;
        return 1;
    } /* if (status != 0) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_ls':"
            << endl <<
            "'Scan_Parse_Parameter_Type::send_to_peer' succeeded, returning 0." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

594.

```
<Scan_Parse_Parameter_Type::server_action_ls definition 592> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Exiting 'Scan_Parse_Parameter_Type::server_action_ls' successfully"
            << "with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; /* End of Scan_Parse_Parameter_Type::server_action_ls definition */
}
```

595. *server_action_pwd*. [LDF 2013.05.29.]

Log

[LDF 2013.05.29.] Added this function.

```
<Scan_Parse_Parameter_Type::server_action_pwd definition 595> ≡
int Scan_Parse_Parameter_Type::server_action_pwd(Response_Type &response){ bool
    DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    char buffer[BUFFER_SIZE];
    char *buffer_ptr = buffer;
    memset(buffer, 0, BUFFER_SIZE);
    stringstream temp_strm;
    string thread_str;
    temp_strm << "[Thread" << thread_ctr << "] ";
    thread_str = temp_strm.str();
    temp_strm.str("");
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering 'Scan_Parse_Parameter_Type::server_action_pwd':"
            << endl << "'response.type'" <= << Response_Type::typename_map[response.type] <<
            "(" << response.type << ")" << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
}
```

See also sections 596 and 597.

This code is used in section 698.

596.

```

⟨ Scan_Parse_Parameter_Type :: server_action_pwd definition 595 ⟩ +≡
    status = pwd(buffer_ptr, BUFFER_SIZE);
    if (status ≠ 0) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::server_action_pwd':"
        endl << "'Scan_Parse_Parameter_Type::pwd' failed, returning" << status << "."
        endl << "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        ++errors_occurred;
        return 1;
    } /* if (status ≠ 0) */
    status = send_to_peer(&buffer_ptr, 0);
    if (status ≠ 0) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::server_action_pwd':"
        endl << "'Scan_Parse_Parameter_Type::send_to_peer' failed, returning" << status <<
            "."
        endl << "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        ++errors_occurred;
        return 1;
    } /* if (status ≠ 0) */
#endif /* DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_pwd':"
    endl << "'Scan_Parse_Parameter_Type::pwd' succeeded, returning 0." << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

597.

```

⟨ Scan_Parse_Parameter_Type :: server_action_pwd definition 595 ⟩ +≡
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "Exiting 'Scan_Parse_Parameter_Type::server_action_pwd' successfully"
    << "with return value 0." << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
return 0; /* End of Scan_Parse_Parameter_Type::server_action_pwd definition */

```

598. *server_action_cd.* [LDF 2013.05.29.]

Log

[LDF 2013.05.29.] Added this function.

```

⟨ Scan_Parse_Parameter_Type :: server_action_cd definition 598 ⟩ ≡
int Scan_Parse_Parameter_Type :: server_action_cd(Response_Type &response){ bool
    DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    char buffer[BUFFER_SIZE];
    char *buffer_ptr = buffer;
    memset(buffer, 0, BUFFER_SIZE);
    stringstream temp_strm;
    string thread_str;
    temp_strm << "[Thread_" << thread_ctr << "]_";
    thread_str = temp_strm.str();
    temp_strm.str("");
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering 'Scan_Parse_Parameter_Type::server_action_cd':"
        endl << "'response.type' == " << Response_Type::typename_map[response.type] <<
            "(" << response.type << ")" << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

See also sections 599 and 600.

This code is used in section 698.

599.

```

< Scan_Parse_Parameter_Type::server_action_cd definition 598 > +≡
status = cd(response.dirname, buffer_ptr, BUFFER_SIZE);
if (status ≡ 3) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "WARNING! In 'Scan_Parse_Parameter_Type::server_action_cd':"
    endl ≪ "'Scan_Parse_Parameter_Type::cd' failed, returning 3." ≪ endl;
    unlock_cerr_mutex();
    ++warnings_occurred;
    temp_strm.str("");
    temp_strm ≪ "CD RESPONSE 7 \""
    ≪ response.dirname ≪ "\" "
    ≪ irods_current_dir ≪ "\" "
    ≪ irods_current_dir ≪ "\" "
    ≪ "'Scan_Parse_Parameter_Type::cd' failed\"";
    strcpy(buffer, temp_strm.str().c_str());
} /* if (status ≡ 3) */
else if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "ERROR! In 'Scan_Parse_Parameter_Type::server_action_cd':"
    endl ≪ "'Scan_Parse_Parameter_Type::cd' failed, returning" ≪ status ≪ "."
    endl ≪ "Exiting function unsuccessfully with return value 1."
    unlock_cerr_mutex();
    return 1;
} /* if (status ≠ 0) */
status = send_to_peer(&buffer_ptr, 0);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "ERROR! In 'Scan_Parse_Parameter_Type::server_action_cd':"
    endl ≪ "'Scan_Parse_Parameter_Type::send_to_peer' failed, returning" ≪ status ≪ "."
    endl ≪ "Exiting function unsuccessfully with return value 1."
    unlock_cerr_mutex();
    ++errors_occurred;
    return 1;
} /* if (status ≠ 0) */
#if DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "In 'Scan_Parse_Parameter_Type::server_action_cd':"
    endl ≪ "'Scan_Parse_Parameter_Type::cd' succeeded, returning 0."
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

600.

```
<Scan_Parse_Parameter_Type::server_action_cd definition 598> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Exiting `Scan_Parse_Parameter_Type::server_action_cd' successfully"
            << "with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; /* End of Scan_Parse_Parameter_Type::server_action_cd definition */

```

601. *server_action_mkdir*. [LDF 2013.05.29.]

Log

[LDF 2013.05.29.] Added this function.

```
<Scan_Parse_Parameter_Type::server_action_mkdir definition 601> ≡
int Scan_Parse_Parameter_Type::server_action_mkdir(Response_Type &response){ bool
    DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    char buffer[BUFFER_SIZE];
    char *buffer_ptr = buffer;
    memset(buffer, 0, BUFFER_SIZE);
    stringstream temp_strm;
    string thread_str;
    temp_strm << "[Thread" << thread_ctr << "] ";
    thread_str = temp_strm.str();
    temp_strm.str("");
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering `Scan_Parse_Parameter_Type::server_action_mkdir':"
            << endl << "response.type" <= " " << Response_Type::typename_map[response.type] <<
            "(" << response.type << ")" << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

See also sections 602 and 603.

This code is used in section 698.

602.

```

<Scan_Parse_Parameter_Type::server_action_mkdir definition 601> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "response.flags==" << response.flags << endl <<
            "response.string_vector.size()==" << response.string_vector.size() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    status = mkdir(response, buffer_ptr, BUFFER_SIZE);
    if (status != 0) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::server_action_mkdir':" <<
            endl << "'Scan_Parse_Parameter_Type::mkdir' failed, returning" << status << "." <<
            endl << "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        return 1;
    } /* if (status != 0) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_mkdir':" << endl <<
            "'Scan_Parse_Parameter_Type::mkdir' succeeded, returning 0." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    status = send_to_peer(&buffer_ptr);
    if (status != 0) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::server_action_mkdir':" <<
            endl << "'Scan_Parse_Parameter_Type::send_to_peer' failed, returning" << status << "." <<
            endl << "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        return 1;
    } /* if (status != 0) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_mkdir':" << endl <<
            "'Scan_Parse_Parameter_Type::send_to_peer' succeeded, returning 0." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

603.

```
< Scan_Parse_Parameter_Type :: server_action_mkdir definition 601 > +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Exiting 'Scan_Parse_Parameter_Type::server_action_mkdir' successfully" << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; } /* End of Scan_Parse_Parameter_Type::server_action_mkdir definition */
```

604. server_action_mark_irods_objects_for_deletion. [LDF 2013.08.07.]

Log

[LDF 2013.08.07.] Added this function.

```
< Scan_Parse_Parameter_Type :: server_action_mark_irods_objects_for_deletion definition 604 > ≡
int Scan_Parse_Parameter_Type :: server_action_mark_irods_objects_for_deletion(Response_Type
    &response){ bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    int ret_val = 0;
    char buffer[BUFFER_SIZE];
    char *buffer_ptr = buffer;
    memset(buffer, 0, BUFFER_SIZE);
    stringstream temp_strm;
    string thread_str;
    temp_strm << "[Thread" << thread_ctr << "] ";
    thread_str = temp_strm.str();
    temp_strm.str("");
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering " << "'Scan_Parse_Parameter_Type::server_acti" << "on_mark_irods_objects_for_deletion': " << endl << "'response.type'" << Response_Type::typename_map[response.type] << "(" << response.type << ")" << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

See also sections 605 and 606.

This code is used in section 698.

605.

```

<Scan_Parse_Parameter_Type::server_action_mark_irods_objects_for_deletion definition 604> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "response.flags==" << response.flags << endl <<
            "response.string_vector.size()==" << response.string_vector.size() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    status = mark_irods_objects_for_deletion(response, buffer_ptr, BUFFER_SIZE);
    if (status ≠ 0) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR!" << "In 'Scan_Parse_Para\"
            meter_Type::server_action_mark_irods_objects_for_deletion':" << endl <<
            "'Scan_Parse_Parameter_Type::mark_irods_objects_for_deletion' failed, returning" <<
            status << "." << endl;
        unlock_cerr_mutex();
        ret_val = 2;
    } /* if (status ≠ 0) */
#endif DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_a\
                ction_mark_irods_objects_for_deletion':" << endl <<
                "'Scan_Parse_Parameter_Type::mark_irods_objects_for_deletion' succeeded, returning" <<
                "0." << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    if (buffer_ptr ≠ 0 ∧ strlen(buffer_ptr) > 0) status = send_to_peer(&buffer_ptr);
    else status = send_to_peer(0, 1);
    if (status ≠ 0) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR!" << "In 'Scan_Parse_Para\"
            meter_Type::server_action_mark_irods_objects_for_deletion':" << endl <<
            "'Scan_Parse_Parameter_Type::send_to_peer' failed, returning" << status << "." <<
            endl;
        unlock_cerr_mutex();
        ret_val = 1;
    } /* if (status ≠ 0) */
#endif DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_a\
                ction_mark_irods_objects_for_deletion':" << endl <<
                "'Scan_Parse_Parameter_Type::send_to_peer' succeeded, returning0." << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

606.

```
<Scan_Parse_Parameter_Type::server_action_mark_irods_objects_for_deletion definition 604> +≡
  if (ret_val ≠ 0) {
    lock_cerr_mutex();
    cerr << thread_str << "In\`‘Scan_Parse_Parameter_Type::server_a\
          ction_mark_irods_objects_for_deletion’：“ << endl <<
          "Exiting\`unsuccessfully\`with\`return\`value\`" << ret_val << ".\`" << endl;
    unlock_cerr_mutex();
    return ret_val;
  }
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << thread_str << "Exiting\`" << “Scan_Parse_Parameter_Type::server_acti\
          on_mark_irods_objects_for_deletion’\`" << "successfully\`" <<
          "with\`return\`value\`0.\`" << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  return 0; }
/* End of Scan_Parse_Parameter_Type::server_action_mark_irods_objects_for_deletion definition */
```

607. *server_action_get.* [LDF 2013.05.29.]

Log

[LDF 2013.05.29.] Added this function.

```

⟨ Scan_Parse_Parameter_Type :: server_action_get definition 607 ⟩ ≡
int Scan_Parse_Parameter_Type :: server_action_get(Response_Type &response){ bool
    DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    char buffer[BUFFER_SIZE];
    char *buffer_ptr = buffer;
    memset(buffer, 0, BUFFER_SIZE);
    stringstream temp_strm;
    string thread_str;
    temp_strm << "[Thread" << thread_ctr << "]";
    thread_str = temp_strm.str();
    temp_strm.str("");
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering 'Scan_Parse_Parameter_Type::server_action_get':"
        endl << "'response.type' == " << Response_Type::typename_map[response.type] <<
            "(" << response.type << ")" << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

See also sections 608, 609, 610, 611, 612, and 613.

This code is used in section 698.

608.

```

⟨ Scan_Parse_Parameter_Type::server_action_get definition 607 ⟩ +≡
    status = get(response.local_filename, response.flags, response.remote_filename, buffer_ptr, BUFFER_SIZE);
    if (status ≠ 0) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::server_action_get':"
        endl << "'Scan_Parse_Parameter_Type::get' failed, returning" << status << "."
        endl << "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        return 1;
    } /* if (status ≠ 0) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_get':"
        endl << "'Scan_Parse_Parameter_Type::get' succeeded, returning 0." << endl;
#endif 0
        response.show("response:");
#endif
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

609.

```

⟨ Scan_Parse_Parameter_Type::server_action_get definition 607 ⟩ +≡
    if (strlen(buffer_ptr) ≡ 0) {
#endif DEBUG_COMPILE
        if (DEBUG) /* strlen(buffer_ptr) ≡ 0 */
        {
            lock_cerr_mutex();
            cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_get':"
            endl << "'Scan_Parse_Parameter_Type::get' didn't fill 'buffer_ptr'."
            endl << "This is what's supposed to happen."
            endl << "Sending response to client, queuing the \"SEND\" command"
            endl << "and continuing."
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

610. The characters '\002' and '\003' are ASCII STX ("Start of Text") and ETX ("End of Text"), respectively. They are both converted to the double quotation mark character $T\ddot{A}2$ in *zzlex*, i.e., the scanner function of the client program *gwirdcli*. Using control characters to include the double quotation mark in a delimited string is easier than trying to escape the actual character the right number of times. [LDF 2013.04.05.]

```
< Scan_Parse_Parameter_Type::server_action_get definition 607 > +≡
    temp_strm.str("");
    temp_strm << "GET\FILE\RESPONSE";
    if (response.remote_filename.empty()) temp_strm << "\" << response.local_filename << "\"";
    else temp_strm << "\" << response.remote_filename << "\";
    temp_strm << "2\"Success.\u0026Queuing\002SEND\FILE\003\command.\"";
    strcpy(buffer_ptr, temp_strm.str().c_str());
    temp_strm.str("");
```

611.

```
< Scan_Parse_Parameter_Type::server_action_get definition 607 > +≡
    } /* if (strlen(buffer_ptr) == 0) */
```

612. If *Scan_Parse_Parameter_Type::get* fails, it will create a response to tell the client about the error. If *buffer* is empty at this point, we send a single NULL byte to the client to prevent it from blocking. [LDF 2013.05.29.]

```
< Scan_Parse_Parameter_Type::server_action_get definition 607 > +≡
    if (strlen(buffer) > 0) status = send_to_peer(&buffer_ptr, 0);
    else status = send_to_peer(0, 1);
    if (status != 0) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::server_action_get':"
            << endl << "'Scan_Parse_Parameter_Type::send_to_peer' failed, returning" << status <<
            ". " << endl << "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        return 1;
    } /* if (status != 0) */
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_get':"
                << endl << "'Scan_Parse_Parameter_Type::send_to_peer' succeeded, returning 0."
                << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

613.

```
<Scan_Parse_Parameter_Type::server_action_get definition 607> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Exiting 'Scan_Parse_Parameter_Type::server_action_get' successfully"
            << "with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; /* End of Scan_Parse_Parameter_Type::server_action_get definition */

```

614. server_action_send_metadata. [LDF 2013.05.29.]

Log

[LDF 2013.05.29.] Added this function.

```
<Scan_Parse_Parameter_Type::server_action_send_metadata definition 614> ≡
int Scan_Parse_Parameter_Type::server_action_send_metadata(Response_Type &response){ bool
    DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    char buffer[BUFFER_SIZE];
    char *buffer_ptr = buffer;
    memset(buffer, 0, BUFFER_SIZE);
    stringstream temp_strm;
    string thread_str;
    temp_strm << "[Thread" << thread_ctr << "] ";
    thread_str = temp_strm.str();
    temp_strm.str("");
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering 'Scan_Parse_Parameter_Type::server_action_send_metadata':"
            << endl << "'response.type' == "
            Response_Type::typename_map[response.type] << "(" << response.type << ")" << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

See also sections 615, 616, and 617.

This code is used in section 698.

615.

```
<Scan_Parse_Parameter_Type::server_action_send_metadata definition 614> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_send_metadata':"
            << endl << "Sending metadata to client." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

616.

```
<Scan_Parse_Parameter_Type::server_action_send_metadata definition 614> +≡
    strcpy(buffer, response.command.c_str());
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "response.local_filename=" << response.local_filename << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    status = send_to_peer(&buffer_ptr, 0, response.local_filename);
    if (status ≠ 0) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::server_action_send_m"
            "etadata':" << endl << "'Scan_Parse_Parameter_Type::send_to_peer' failed, returning"
            " " << status << ". " << endl << "Exiting function unsuccessfully with return value 1."
            << endl;
        unlock_cerr_mutex();
        return 1;
    } /* if (status ≠ 0) */
#endif /* DEBUG_COMPILE */
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_send_metadata':"
            << endl << "'Scan_Parse_Parameter_Type::send_to_peer' succeeded, returning 0." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

617.

```
< Scan_Parse_Parameter_Type :: server_action_send_metadata definition 614 > +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Exiting 'Scan_Parse_Parameter_Type::server_action_send_meta" \
            "data' successfully" << "with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; /* End of Scan_Parse_Parameter_Type::server_action_send_metadata definition */

```

618. *server_action_end_server.* [LDF 2013.05.30.]

Log

[LDF 2013.05.30.] Added this function.

```
< Scan_Parse_Parameter_Type :: server_action_end_server definition 618 > ≡
int Scan_Parse_Parameter_Type :: server_action_end_server(Response_Type &response){ bool
    DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    char buffer[BUFFER_SIZE];
    char *buffer_ptr = buffer;
    memset(buffer, 0, BUFFER_SIZE);
    stringstream temp_strm;
    string thread_str;
    temp_strm << "[Thread" << thread_ctr << "] ";
    thread_str = temp_strm.str();
    temp_strm.str("");
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering 'Scan_Parse_Parameter_Type::se" \
            "rver_action_end_server':" << endl << "'response.type'" << \
            Response_Type::typename_map[response.type] << "(" << response.type << ")" << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

See also sections 619, 620, 621, 622, and 623.

This code is used in section 698.

619.

```
< Scan_Parse_Parameter_Type :: server_action_end_server definition 618 > +≡
if (end_server_enabled) {
    strcpy(buffer, "END_SERVER_RESPONSE_0");
} /* if (end_server_enabled) */

```

620. Normally this code should never be reached, because the case that *end_server_enabled* ≡ *false* is handled in the parser rule. [LDF 2013.04.03.]

```
<Scan_Parse_Parameter_Type::server_action_end_server definition 618> +≡
else /* end_server_enabled ≡ false */
{
    lock_cerr_mutex();
    cerr << thread_str << "WARNING! In 'Scan_Parse_Parameter_Type::server_action_end_\
server'" << endl << "'response.type' == 'END_SERVER_TYPE' and " <<
    "'end_server_enabled' == 'false'." << endl <<
    "This shouldn't be possible. However, it doesn't cause any harm." << endl <<
    "Not ending server. Will send failure notice to client and continue." << endl;
unlock_cerr_mutex();
++warnings_occurred;
strcpy(buffer, "END_SERVER_RESPONSE_1");
} /* else (end_server_enabled ≡ false) */
```

621.

```
<Scan_Parse_Parameter_Type::server_action_end_server definition 618> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "'buffer': " << buffer << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    status = send_to_peer(&buffer_ptr, 0);
    if (status ≠ 0) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::server_action_end_server':" <<
            endl << "'Scan_Parse_Parameter_Type::send_to_peer' failed, returning " << status <<
            "." << endl << "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        return 1;
    } /* if (status ≠ 0) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_end_server':" <<
            endl << "'Scan_Parse_Parameter_Type::send_to_peer' succeeded, returning 0." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

622.

Log

[LDF 2013.04.18.] Added this section.

```
⟨ Scan_Parse_Parameter_Type::server_action_end_server definition 618 ⟩ +≡
if (end_server_enabled) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_end_server':"
    endl << "'END_SERVER' command succeeded." <<
        "Exiting function successfully with return value 2." << endl;
    unlock_cerr_mutex();
    return 2;
} /* if (end_server_enabled) */
```

623.

```
⟨ Scan_Parse_Parameter_Type::server_action_end_server definition 618 ⟩ +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Exiting 'Scan_Parse_Parameter_Type::server_action_end_server'"
            " successfully" << "with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; } /* End of Scan_Parse_Parameter_Type::server_action_end_server definition */
```

624. *server_action_sleep.* [LDF 2013.05.29.]

Log

[LDF 2013.05.29.] Added this function.

```

⟨ Scan_Parse_Parameter_Type :: server_action_sleep definition 624 ⟩ ≡
int Scan_Parse_Parameter_Type :: server_action_sleep(Response_Type &response){ bool
    DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    char buffer[BUFFER_SIZE];
    char *buffer_ptr = buffer;
    memset(buffer, 0, BUFFER_SIZE);
    stringstream temp_strm;
    string thread_str;
    temp_strm << "[Thread_" << thread_ctr << "]_";
    thread_str = temp_strm.str();
    temp_strm.str("");
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering_`Scan_Parse_Parameter_Type::server_action_sleep':"
            << endl << `response.type'_ ==_> Response_Type::typename_map[response.type] <<
            "(_" << response.type << ")" << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

See also sections 625, 626, 627, 628, 629, and 630.

This code is used in section 698.

625.

```

⟨ Scan_Parse_Parameter_Type :: server_action_sleep definition 624 ⟩ +≡
if (sleep_server_enabled) {
    temp_strm.str("");
    temp_strm << "SLEEP_RESPONSE_0_" << response.int_val;
    strncpy(buffer, temp_strm.str().c_str(), BUFFER_SIZE);
    temp_strm.str("");
} /* if (sleep_server_enabled) */

```

626. Normally this code should never be reached, because the case that *sleep-server-enabled* \equiv *false* is handled in the parser rule. [LDF 2013.04.03.]

```
<Scan_Parse_Parameter_Type::server_action_sleep definition 624> +≡
else /* sleep-server-enabled ≡ false */
{
    lock_cerr_mutex();
    cerr << thread_str << "WARNING! In 'Scan_Parse_Parameter_Type::server_action_sleep':"
        endl << "'response.type' == 'SLEEP_TYPE' and "
        "'sleep_server_enabled' == 'false'." << endl <<
        "This shouldn't be possible. However, it doesn't cause any harm." << endl <<
        "Not ending server. Will send failure notice to client and continue." << endl;
    unlock_cerr_mutex();
    ++warnings_occurred;
    temp_strm.str("");
    temp_strm << "SLEEP_RESPONSE_1" << response.int_val;
    strncpy(buffer, temp_strm.str().c_str(), BUFFER_SIZE);
    temp_strm.str("");
}
/* else (sleep-server-enabled ≡ false) */
```

627.

```
<Scan_Parse_Parameter_Type::server_action_sleep definition 624> +≡
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "'buffer': " << buffer << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
status = send_to_peer(&buffer_ptr, 0);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::server_action_sleep':"
        endl << "'Scan_Parse_Parameter_Type::send_to_peer' failed, returning " << status <<
        "." << endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    return 1;
} /* if (status ≠ 0) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_sleep': " << endl <<
            "'Scan_Parse_Parameter_Type::send_to_peer' succeeded, returning 0." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

628.

```
<Scan_Parse_Parameter_Type::server_action_sleep definition 624> +≡
    if (sleep_server_enabled) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_sleep':" << endl <<
            "Going to sleep for " << response.int_val << " seconds...";
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    status = sleep(response.int_val);
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_sleep':" << endl <<
            "Woke up after " << (response.int_val - status) << " seconds." << endl;
        if (status ≠ 0)
            cerr << "Was supposed to sleep for " << response.int_val << " seconds." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    } /* if (sleep_server_enabled) */
```

629. This code should never be reached, because this case is caught in *yyparse*. [LDF 2013.04.19.]

```
<Scan_Parse_Parameter_Type::server_action_sleep definition 624> +≡
#ifndef DEBUG_COMPILE
else
    if (DEBUG) /* sleep_server_enabled ≡ false */
    {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_sleep':" << endl <<
            "'sleep_server_enabled' == 'false'. Not going to sleep. Continuing." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) (sleep_server_enabled ≡ false) */
#endif /* DEBUG_COMPILE */
```

630.

```
<Scan_Parse_Parameter_Type::server_action_sleep definition 624> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Exiting 'Scan_Parse_Parameter_Type::server_action_sleep' successfully"
            << "with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; } /* End of Scan_Parse_Parameter_Type::server_action_sleep definition */
```

631. *server_action_show_certificate.* [LDF 2013.05.29.]

Log

[LDF 2013.05.29.] Added this function.

```
{ Scan_Parse_Parameter_Type::server_action_show_certificate definition 631 } ≡
int Scan_Parse_Parameter_Type::server_action_show_certificate(Response_Type &response){
    bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    char buffer[BUFFER_SIZE];
    char *buffer_ptr = buffer;
    memset(buffer, 0, BUFFER_SIZE);
    stringstream temp_strm;
    string thread_str;
    temp_strm << "[Thread_" << thread_ctr << "] ";
    thread_str = temp_strm.str();
    temp_strm.str("");
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering 'Scan_Parse_Parameter_Type::se\
rver_action_show_certificate': " << endl << "'response.type' == " <<
        Response_Type::typename_map[response.type] << "(" << response.type << ")" << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

See also sections 632 and 633.

This code is used in section 698.

632.

```

⟨ Scan_Parse_Parameter_Type::server_action_show_certificate definition 631 ⟩ +≡
string temp_filename;
status = show_certificates(response, buffer, BUFFER_SIZE, temp_filename);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << thread_str << "WARNING! In 'Scan_Parse_Parameter_Type\\
        ::server_action_show_certificate': " << endl <<
        "'Scan_Parse_Parameter_Type::show_certificates' failed, returning" <<
        status << "." << endl << "Will try to continue." << endl;
    unlock_cerr_mutex();
    ++warnings_occurred;
} /* if (status ≠ 0) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_show_certificate': " <<
            endl << "'Scan_Parse_Parameter_Type::show_certificates' succeeded, returning" <<
            "0." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "'buffer': " << buffer << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    status = send_to_peer(&buffer_ptr, 0, temp_filename);
    if (status ≠ 0) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type:\\
            server_action_show_certificate': " << endl <<
            "'Scan_Parse_Parameter_Type::send_to_peer' failed, returning" <<
            status << "." << "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        return 1;
    } /* if (status ≠ 0) */
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_show_certificate': " <<
                endl << "'Scan_Parse_Parameter_Type::send_to_peer' succeeded, returning" << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

633.

```
<Scan_Parse_Parameter_Type::server_action_show_certificate definition 631> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Exiting 'Scan_Parse_Parameter_Type::server_action_show_certificate'" << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; } /* End of Scan_Parse_Parameter_Type::server_action_show_certificate definition */
```

634. server_action_get_metadata. [LDF 2013.05.29.]

Response_Type ::**GET_METADATA_TYPE** is also used for the GET PID and GET PIDS commands (which are synonymous). [LDF 2013.05.23.]

Log

[LDF 2013.05.29.] Added this function.

```
<Scan_Parse_Parameter_Type::server_action_get_metadata definition 634> ≡
int Scan_Parse_Parameter_Type::server_action_get_metadata(Response_Type &response){ bool
    DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    char buffer[BUFFER_SIZE];
    char *buffer_ptr = buffer;
    memset(buffer, 0, BUFFER_SIZE);
    stringstream temp_strm;
    string thread_str;
    temp_strm << "[Thread_" << thread_ctr << "]_";
    thread_str = temp_strm.str();
    temp_strm.str("");
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering 'Scan_Parse_Parameter_Type::server_action_get_metadata':"
            << endl << "'response.type' == "
            Response_Type::typename_map[response.type] << "(" << response.type << ")" << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

See also sections 635 and 636.

This code is used in section 698.

635.

```

< Scan_Parse_Parameter_Type::server_action_get_metadata definition 634 > +≡
    string temp_filename;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In\u0027Scan_Parse_Parameter_Type::server_action_get_metadata:'" <<
            endl << "response.type\u003dResponse_Type::GET_METADATA_TYPE" << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    int ctr;
    bool do_output = (static_cast<unsigned int>(response.int_val) & 1) ? false : true;
    status = get_metadata(response.local_filename, response.int_val, &ctr, 0, buffer, BUFFER_SIZE, do_output);
    if (status != 0) {
        lock_cerr_mutex();
        cerr << thread_str << "WARNING! In\u0027Scan_Parse_Parameter_Type::server_action_get_\u0027
            metadata:'" << endl << "'Scan_Parse_Parameter_Type::get_metadata'\u003dfailed,\u0027 <<
            "returning\u0027" << status << "." << endl << "Will\u0027try\u0027to\u0027continue." << endl;
        unlock_cerr_mutex();
        ++warnings_occurred;
    } /* if (status != 0) */
#endif DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_str << "In\u0027Scan_Parse_Parameter_Type::server_action_get_metadata:'" <<
                endl << "'Scan_Parse_Parameter_Type::get_metadata'\u003dsucceeded,\u0027 << "returning\u00270." <<
                endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "'buffer':\u0027" << buffer << endl << "'temp_filename':\u0027" << temp_filename << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    status = send_to_peer(&buffer_ptr, 0, temp_filename);
    if (status != 0) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In\u0027Scan_Parse_Parameter_Type::server_action_get_me\
            tadata:'" << endl << "'Scan_Parse_Parameter_Type::send_to_peer'\u003dfailed,\u0027returning\u0027" <<
            status << "." << endl << "Exiting\u0027function\u0027unsuccessfully\u0027with\u0027return\u0027value\u00271." << endl;
        unlock_cerr_mutex();
        return 1;
    } /* if (status != 0) */
#endif DEBUG_COMPILE
    else
        if (DEBUG) {

```

```

lock_cerr_mutex();
cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_get_metadata':" <<
    endl << 'Scan_Parse_Parameter_Type::send_to_peer' succeeded, returning 0." << endl;
unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

636.

```

< Scan_Parse_Parameter_Type::server_action_get_metadata definition 634 > +≡
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "Exiting 'Scan_Parse_Parameter_Type::server_action_get_metadata'
        successfully" << "with return value 0." << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
return 0; } /* End of Scan_Parse_Parameter_Type::server_action_get_metadata definition */

```

637. *server_action_get_handle*. [LDF 2013.05.29.]

If **Scan_Parse_Parameter_Type**::*get_handle* succeeds, it sets up a response of type **Response_Type**::SEND_HANDLE and pushes it onto *param.response_deque*. Therefore, the successful case cannot be handled within this conditional. The error cases could, because the responses are of type **Response_Type**::COMMAND_ONLY_TYPE and they could be sent from here. However, I don't think the advantage is so great that I would want to change the code in **Scan_Parse_Parameter_Type**::*get_handle* at this time (or possibly any other time, either). [LDF 2013.05.23.]

Log

[LDF 2013.05.29.] Added this function.

```
< Scan_Parse_Parameter_Type::server_action_get_handle definition 637 > ≡
int Scan_Parse_Parameter_Type::server_action_get_handle(Response_Type &response){ bool
    DEBUG = false;      /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    char buffer[BUFFER_SIZE];
    char *buffer_ptr = buffer;
    memset(buffer, 0, BUFFER_SIZE);
    stringstream temp_strm;
    string thread_str;
    string temp_filename;
    temp_strm << "[Thread_" << thread_ctr << "]_";
    thread_str = temp_strm.str();
    temp_strm.str("");
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering 'Scan_Parse_Parameter_Type::se\
rver_action_get_handle':" << endl << "'response.type' == " <<
        Response_Type::typename_map[response.type] << "(" << response.type << ")" << endl;
        unlock_cerr_mutex();
    }      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */
```

See also sections 638, 639, and 640.

This code is used in section 698.

638.

```

<Scan_Parse_Parameter_Type::server_action_get_handle definition 637> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_get_handle':" << endl <<
            "response.type==Response_Type::GET_HANDLE_TYPE" << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    string temp_str = (response.int_val == 0) ? response.pid_str : response.local_filename;
    status = get_handle(temp_str, response.int_val);
    if (status == 2) {
        lock_cerr_mutex();
        cerr << thread_str << "WARNING! In 'Scan_Parse_Parameter_Type::server_action_get_\nhandle':" << endl << "'Scan_Parse_Parameter_Type::get_handle' returned 2: Handle\nnot found." << endl << "Continuing." << endl;
        unlock_cerr_mutex();
        ++warnings_occurred;
    } /* if (status == 2) */
    else if (status != 0) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::server_action_get_handle':" <<
            endl << "'Scan_Parse_Parameter_Type::get_handle' failed," << "returning" << status <<
            "." << endl << "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        return 1;
        ++errors_occurred;
    } /* else if (status != 0) */
#endif DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_get_handle':" <<
                endl << "'Scan_Parse_Parameter_Type::get_handle' succeeded," << "returning 0." <<
                endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

639.

Log

[LDF 2013.07.03.] BUG FIX: Now storing the return value of *send_to_peer* in *status*.

```
<Scan_Parse_Parameter_Type::server_action_get_handle definition 637> +≡
status = send_to_peer(0, 1);
/* Send a single NULL byte to client, so it won't block. [LDF 2013.05.23.] */
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::server_action_get_handle':"
        << endl << "'Scan_Parse_Parameter_Type::send_to_peer' failed, returning" << status <<
        "." << endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    return 1;
} /* if (status ≠ 0) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_get_handle':"
            << endl << "'Scan_Parse_Parameter_Type::send_to_peer' succeeded, returning 0." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

640.

```
<Scan_Parse_Parameter_Type::server_action_get_handle definition 637> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Exiting 'Scan_Parse_Parameter_Type::server_action_get_handle'
            e' successfully" << "with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; } /* End of Scan_Parse_Parameter_Type::server_action_get_handle definition */
```

641. *server_action_send_tan_list.* [LDF 2013.05.29.]

Log

[LDF 2013.05.29.] Added this function.

```

⟨ Scan_Parse_Parameter_Type :: server_action_send_tan_list definition 641 ⟩ ≡
int Scan_Parse_Parameter_Type :: server_action_send_tan_list(Response_Type &response){ bool
    DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    char buffer[BUFFER_SIZE];
    char *buffer_ptr = buffer;
    memset(buffer, 0, BUFFER_SIZE);
    stringstream temp_strm;
    string thread_str;
    temp_strm << "[Thread_" << thread_ctr << "]_";
    thread_str = temp_strm.str();
    temp_strm.str("");
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering 'Scan_Parse_Parameter_Type::se\
rver_action_send_tan_list': " << endl << "'response.type' == " <<
        Response_Type::typename_map[response.type] << "(" << response.type << ")" << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

See also sections 642 and 643.

This code is used in section 698.

642.

```

⟨ Scan_Parse_Parameter_Type::server_action_send_tan_list definition 641 ⟩ +≡
status = send_tan_list();
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << thread_str << "WARNING! In 'Scan_Parse_Parameter_Type::server_action_send\
_tan_list':" << endl << "'Scan_Parse_Parameter_Type::send_tan_list' failed," <<
    "returning" << status << "." << endl << "Will try to continue." << endl;
    unlock_cerr_mutex();
    ++warnings_occurred;
} /* if (status ≠ 0) */
#if DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_send_tan_list':" <<
        endl << "'Scan_Parse_Parameter_Type::send_tan_list' succeeded," <<
        "returning 0." << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
send_to_peer(0, 1); /* Send a single NULL byte to client, so it won't block. [LDF 2013.05.23.] */

```

643.

```

⟨ Scan_Parse_Parameter_Type::server_action_send_tan_list definition 641 ⟩ +≡
#if DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "Exiting 'Scan_Parse_Parameter_Type::server_action_send_tan_\
list' successfully" << "with return value 0." << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
return 0; /* End of Scan_Parse_Parameter_Type::server_action_send_tan_list definition */

```

644. *server_action_process_pending.* [LDF 2013.05.29.]

Log

[LDF 2013.05.29.] Added this function.

```

⟨ Scan_Parse_Parameter_Type :: server_action_process_pending definition 644 ⟩ ≡
int Scan_Parse_Parameter_Type :: server_action_process_pending(Response_Type &response){
    bool DEBUG = false;      /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    char buffer[BUFFER_SIZE];
    char *buffer_ptr = buffer;
    memset(buffer, 0, BUFFER_SIZE);
    string temp_filename;
    stringstream temp_strm;
    string thread_str;
    temp_strm << "[Thread_" << thread_ctr << "]_";
    thread_str = temp_strm.str();
    temp_strm.str("");
#define DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering 'Scan_Parse_Parameter_Type::se\
rver_action_process_pending': " << endl << "'response.type' == " <<
        Response_Type::typename_map[response.type] << "(" << response.type << ")" << endl;
        unlock_cerr_mutex();
    }      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */

```

See also sections 645, 646, 647, and 648.

This code is used in section 698.

645.

```

⟨Scan_Parse_Parameter_Type::server_action_process_pending definition 644⟩ +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) { lock_cerr_mutex();
        cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_process_pending'" <<
            endl << "response.type==" Response_Type::PROCESS_PENDING_TYPE" << endl;
        cerr << "response_deque.size()==" << response_deque.size() << endl <<
            "delayed_response_deque.size()==" << delayed_response_deque.size() << endl; if
            (response_deque.size() > 0) { cerr << "response_deque:" << endl; for ( deque < Response_Type
            > ::iterator iter = response_deque.begin();
        iter != response_deque.end(); ++iter )
    {
        cerr << "Type:" << Response_Type::typename_map[iter->type] << endl;
        if (iter->type == Response_Type::COMMAND_ONLY_TYPE)
            cerr << "Command:" << iter->command << endl;
    }
    cerr << endl; } /* if */
    if (delayed_response_deque.size() > 0) { cerr << "delayed_response_deque:" << endl; for ( deque <
        Response_Type > ::iterator iter = delayed_response_deque.begin();
    iter != delayed_response_deque.end(); ++iter )
    {
        cerr << "Type:" << Response_Type::typename_map[iter->type] << endl;
        if (iter->type == Response_Type::COMMAND_ONLY_TYPE)
            cerr << "Command:" << iter->command << endl;
    }
    cerr << endl; } /* if */
    unlock_cerr_mutex(); } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

646.

Log

[LDF 2013.08.12.] BUG FIX: Added code for testing `response_deque.size()`. Previously, `temp_deque.push_back()` caused an error, if `response_deque` was empty.

```

< Scan_Parse_Parameter_Type ::server_action_process_pending definition 644 > +≡
deque < Response_Type > temp_deque; deque < Response_Type > ::iterator iter_1;
if (response_deque.size() > 0) {
    iter_1 = response_deque.begin();
    temp_deque.push_back(*iter_1);
    ++iter_1;
}
else iter_1 = response_deque.end();
if (iter_1 != response_deque.end()) {
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "After_increment:" << endl << "iter_1->type':uuuuu" <<
        Response_Type::typename_map[iter_1->type] << endl << "iter_1->command':u" <<
        iter_1->command << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
for ( ; iter_1 != response_deque.end(); ++iter_1) {
    if (iter_1->no_delay) temp_deque.push_back(*iter_1);
    else delayed_response_deque.push_back(*iter_1);
}
response_deque.clear();
response_deque = temp_deque;
temp_deque.clear();
#endif DEBUG_COMPILE
if (DEBUG) { lock_cerr_mutex();
cerr << "After_inserting:" << endl; if (response_deque.size() > 0) {
    cerr << "response_deque:" << endl; for ( deque < Response_Type > ::iterator
        iter = response_deque.begin();
    iter != response_deque.end(); ++iter )
    {
        cerr << "Type:uu" << Response_Type::typename_map[iter->type] << endl;
        if (iter->type == Response_Type::COMMAND_ONLY_TYPE)
            cerr << "Command:uu" << iter->command << endl;
    }
    cerr << endl; } /* if */
if (delayed_response_deque.size() > 0) { cerr << "delayed_response_deque:" << endl; for ( deque <
    Response_Type > ::iterator iter = delayed_response_deque.begin();
    iter != delayed_response_deque.end(); ++iter )
    {
        cerr << "Type:uu" << Response_Type::typename_map[iter->type] << endl;
        if (iter->type == Response_Type::COMMAND_ONLY_TYPE)
            cerr << "Command:uu" << iter->command << endl;
    }
}

```

```

    cerr << endl; } /* if */
    unlock_cerr_mutex(); } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (iter_1 != response_deque.end()) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "No following responses." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

647.

```

< Scan_Parse_Parameter_Type::server_action_process_pending definition 644 > +≡
#ifndef 0 /* 1 For testing. [LDF 2013.05.24.] */ /* Send a response to the client. [LDF 2013.05.24.] */
    strcpy(buffer, "PROCESS_PENDING_OPERATIONS_RESPONSE_0");
    send_to_peer(&buffer_ptr, 0);
    memset(buffer, 0, BUFFER_SIZE);
#else
    send_to_peer(0, 1); /* Just send a single NULL byte, so the client won't block. [LDF 2013.05.24.] */
#endif
    if (status != 0) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::"
            server_action_process_pending': " << endl <<
            "'Scan_Parse_Parameter_Type::send_to_peer' failed, returning" <<
            status << ". " << endl << "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        return 1;
    } /* if (status != 0) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_process_pending': "
            endl << "'Scan_Parse_Parameter_Type::send_to_peer' succeeded, returning 0." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

648.

```
< Scan_Parse_Parameter_Type :: server_action_process_pending definition 644 > +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Exiting `Scan_Parse_Parameter_Type::server_action_process_p`"
            << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; } /* End of Scan_Parse_Parameter_Type::server_action_process_pending definition */
```

649. *server_action_get_user_info.* [LDF 2013.05.30.]

This function will only be called if the GET_USER_INFO command was called using the DELAY option. Otherwise, *get_user_info_func* is called immediately in *yyparse*.

Under normal circumstances, the user info can be retrieved and sent to the client immediately. The DELAY option is for the case that commands are used that affect the user info before calling the GET_USER_INFO command in a single batch of input. [LDF 2013.05.24.]

Log

[LDF 2013.05.30.] Added this function.

```
< Scan_Parse_Parameter_Type :: server_action_get_user_info definition 649 > +≡
int Scan_Parse_Parameter_Type :: server_action_get_user_info(Response_Type &response){ bool
    DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    char buffer[BUFFER_SIZE];
    char *buffer_ptr = buffer;
    memset(buffer, 0, BUFFER_SIZE);
    stringstream temp_strm;
    string thread_str;
    temp_strm << "[Thread_" << thread_ctr << "] ";
    thread_str = temp_strm.str();
    temp_strm.str("");
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering `Scan_Parse_Parameter_Type::se`"
            << endl << "'response.type' == "
            Response_Type::typename_map[response.type] << "(" << response.type << ")" << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

See also sections 650 and 651.

This code is used in section 698.

650.

```

⟨ Scan_Parse_Parameter_Type::server_action_get_user_info definition 649 ⟩ +≡
status = get_user_info_func(this, response.string_val.c_str());
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << thread_str << "WARNING! In 'Scan_Parse_Parameter_Type::server_action_get_＼
        user_info':" << endl << "'get_user_info_func' failed, returning" << status << "."
    endl << "Will try to continue." << endl;
    unlock_cerr_mutex();
    ++warnings_occurred;
} /* if (status ≠ 0) */
#endif /* DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_get_user_info':"
    endl << "'get_user_info_func' succeeded, returning 0." << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
send_to_peer(0, 1); /* Send NULL byte so client won't block. [LDF 2013.05.24.] */
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::server_action_get_us＼
        er_info':" << endl << "'Scan_Parse_Parameter_Type::send_to_peer' failed, returning"
    endl << status << "."
    endl << "Exiting function unsuccessfully with return value 1."
    endl;
    unlock_cerr_mutex();
    return 1;
} /* if (status ≠ 0) */
#endif /* DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_get_user_info':"
    endl << "'Scan_Parse_Parameter_Type::send_to_peer' succeeded, returning 0."
    endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

651.

```

⟨ Scan_Parse_Parameter_Type::server_action_get_user_info definition 649 ⟩ +≡
#endif /* DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "Exiting 'Scan_Parse_Parameter_Type::server_action_get_user_＼
        info' successfully"
    endl << "with return value 0." << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
return 0; } /* End of Scan_Parse_Parameter_Type::server_action_get_user_info definition */

```

652. *server_action_create_handle*. [LDF 2013.05.30.]

Log

[LDF 2013.05.30.] Added this function.

```

⟨ Scan_Parse_Parameter_Type :: server_action_create_handle definition 652 ⟩ ≡
int Scan_Parse_Parameter_Type :: server_action_create_handle(Response_Type &response){ bool
    DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    char buffer[BUFFER_SIZE];
    char *buffer_ptr = buffer;
    memset(buffer, 0, BUFFER_SIZE);
    stringstream temp_strm;
    string thread_str;
    temp_strm << "[Thread_" << thread_ctr << "]_";
    thread_str = temp_strm.str();
    temp_strm.str("");
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering 'Scan_Parse_Parameter_Type::se\
rver_action_create_handle': " << endl << "'response.type' == " <<
        Response_Type::typename_map[response.type] << "(" << response.type << ")" << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

See also sections 653 and 654.

This code is used in section 698.

653.

```

< Scan_Parse_Parameter_Type ::server_action_create_handle definition 652 > +≡
status = generate_pids(mysql_ptr, response.pid_prefix_str, response.pid_str, 0, 1, 0, 0, true,
    response.pid_institute_str, response.pid_suffix_str, 0, "", user_id, username);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "ERROR! In 'Scan_Parse_Parameter_Type::server_action_create\
        _handle': " ≪ endl ≪ "'generate_pids' failed, returning" ≪ status ≪
        ". " ≪ endl ≪ "Failed to create PID\\"" ≪ response.pid_str ≪ "\\". " ≪ endl ≪
        "Will send error message to client and continue." ≪ endl;
    unlock_cerr_mutex();
    temp_strm.str("");
    temp_strm ≪ "CREATE_HANDLE_RESPONSE_1\\"" ≪ response.pid_str ≪ "\\"";
    strcpy(buffer, temp_strm.str().c_str());
    temp_strm.str("");
    ++errors_occurred;
} /* if (status ≠ 0) */
else {
#endif /* DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ thread_str ≪ "In 'Scan_Parse_Parameter_Type::server_action_create_handle': " ≪
            endl ≪ "'generate_pids' succeeded, returning 0." ≪ endl ≪ "Created PID\\"" ≪
            response.pid_str ≪ "\\" successfully." ≪ endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    temp_strm.str("");
    temp_strm ≪ "CREATE_HANDLE_RESPONSE_0\\"" ≪ response.pid_str ≪ "\\"";
    strcpy(buffer, temp_strm.str().c_str());
    temp_strm.str("");
} /* else */
status = send_to_peer(&buffer_ptr, 0);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "ERROR! In 'Scan_Parse_Parameter_Type::server_action_create\
        _handle': " ≪ endl ≪ "'Scan_Parse_Parameter_Type::send_to_peer' failed, returning\
        " ≪ status ≪ ". " ≪ endl ≪ "Exiting function unsuccessfully with return value 1." ≪
        endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    return 1;
} /* if (status ≠ 0) */
#endif /* DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ thread_str ≪ "In 'Scan_Parse_Parameter_Type::server_action_create_handle': " ≪
            endl ≪ "'Scan_Parse_Parameter_Type::send_to_peer' succeeded, returning 0." ≪ endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

654.

```
< Scan_Parse_Parameter_Type :: server_action_create_handle definition 652 > +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Exiting 'Scan_Parse_Parameter_Type::server_action_create_ha\
ndle'" << " successfully" << " with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; /* End of Scan_Parse_Parameter_Type::server_action_create_handle definition */

```

655. *server_action_add_handle_value.* [LDF 2013.06.15.]

Log

[LDF 2013.06.15.] Added this function.

```
< Scan_Parse_Parameter_Type :: server_action_add_handle_value definition 655 > ≡
int Scan_Parse_Parameter_Type :: server_action_add_handle_value(Response_Type &response){
    bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    char buffer[BUFFER_SIZE];
    char *buffer_ptr = buffer;
    memset(buffer, 0, BUFFER_SIZE);
    stringstream temp_strm;
    string thread_str;
    temp_strm << "[Thread" << thread_ctr << "] ";
    thread_str = temp_strm.str();
    temp_strm.str("");
    Handle_Type handle;
    Handle_Value_Triple hvt;
    int ret_val = 0;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering 'Scan_Parse_Parameter_Type::se\
rver_action_add_handle_value':" << endl << "'response.type' == " <<
        Response_Type::typename_map[response.type] << "(" << response.type << ")" << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

See also sections 656, 657, 658, and 659.

This code is used in section 698.

656.

```

⟨ Scan_Parse_Parameter_Type :: server_action_add_handle_value definition 655 ⟩ +≡
  if (response.pid_str.empty() ∨ (response.hvt.idx ≡ 0 ∧ response.hvt.type.empty() ∧
    response.hvt.data_str.empty())) {
    lock_cerr_mutex();
    cerr << thread_str << "WARNING! In 'Scan_Parse_Parameter_Type' "
    ::server_action_add_handle_value' :" << endl <<
    "'Response_Type' response' doesn't contain sufficient data to add a handle value:" <<
    endl << "response.pid_str==" << response.pid_str << endl << "response.hvt.idx==" <<
    response.hvt.idx << endl << "response.hvt.type==" << response.hvt.type <<
    endl << "response.hvt.data_str==" << response.hvt.data_str << endl <<
    "Will send error message to client and continue." << endl;
    unlock_cerr_mutex();
    temp_strm.str("");
    temp_strm << "ADD_HANDLE_VALUE_RESPONSE\4\" " << response.pid_str << "\0\0\0\0";
    strcpy(buffer, temp_strm.str().c_str());
    temp_strm.str("");
    ++errors_occurred;
    ret_val = 2;
    goto FINISH;
} /* if */

```

657.

```

⟨ Scan_Parse_Parameter_Type :: server_action_add_handle_value definition 655 ⟩ +≡
status = fetch_handle_from_database(response.pid_str, handle);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::server_action_add_ha-
    ndle_value': " << endl << "'Scan_Parse_Parameter_Type::fetch_handle-
    e_from_database' failed, returning " << status << "." << endl <<
    "Failed to fetch handle object for PID " << response.pid_str << "\n" <<
    "from database." << endl << "Will send error message to client and continue." << endl;
unlock_cerr_mutex();
temp_strm.str("");
temp_strm << "ADD_HANDLE_VALUE_RESPONSE_3\n" << response.pid_str << "\n0\n\n\n";
strcpy(buffer, temp_strm.str().c_str());
temp_strm.str("");
++errors_occurred;
ret_val = 2;
goto FINISH;
} /* if (status ≠ 0) */
#endif /* DEBUG_COMPILE */
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_add_handle_value': " <<
        endl << "'Scan_Parse_Parameter_Type::fetch_handle_from_database' suc-
        ceeded, returning 0." << endl;
    handle.show("handle:");
    response.show("response:");
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
status = handle.add_value(mysql_ptr, response.hvt.idx, response.hvt.type, response.hvt.data_str, user_id,
&hvt);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::server_action_add_ha-
    ndle_value': " << endl << "'Handle_Type::add_handle_value' failed, returning " <<
    status << "." << endl << "Failed to add handle value to handle " << response.pid_str <<
    "\n." << endl << "Will send error message to client and continue." << endl;
unlock_cerr_mutex();
temp_strm.str("");
temp_strm << "ADD_HANDLE_VALUE_RESPONSE_1\n" << response.pid_str << "\n0\n\n\n";
strcpy(buffer, temp_strm.str().c_str());
temp_strm.str("");
++errors_occurred;
ret_val = 2;
goto FINISH;
} /* if (status ≠ 0) */
else {
#endif /* DEBUG_COMPILE */
if (DEBUG) {
    lock_cerr_mutex();

```

```

cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_add_handle_value':" <<
    endl << "'add_handle_value' succeeded, returning 0." << endl <<
    "Added handle value to handle\"" << response.pid.str << "\" successfully." << endl;
    hvt.show("hvt:");
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
temp_strm.str("");
temp_strm << "ADD_HANDLE_VALUE_RESPONSE_0\""
    << response.pid.str << "\" << hvt.idx <<
    "\" << hvt.type << "\"";
if (hvt.data_str[0] == '\x1e') temp_strm << "(binary)\\"";
else temp_strm << "\" << hvt.data_str << "\"";
strcpy(buffer, temp_strm.str().c_str());
#if DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "buffer==" << buffer << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
temp_strm.str("");
} /* else */
FINISH: status = send_to_peer(&buffer_ptr, 0);
if (status != 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::\n"
        "server_action_add_handle_value':" << endl <<
        "'Scan_Parse_Parameter_Type::send_to_peer' failed, returning"
        " " << endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    return 1;
} /* if (status != 0) */
#endif /* DEBUG_COMPILE */
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_add_handle_value':" <<
        endl << "'Scan_Parse_Parameter_Type::send_to_peer' succeeded, returning 0." << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

658.

```
< Scan_Parse_Parameter_Type :: server_action_add_handle_value definition 655 > +≡
    if (ret_val ≠ 0) {
#if DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ thread_str ≪ "In 'Scan_Parse_Parameter_Type::server_action_add_handle_value':"
            endl ≪ "'ret_val' == " ≪ ret_val ≪ endl ≪
            "Error or warning occurred. Exiting function with return value 'ret_val' == "
            "==" ≪ ret_val ≪ "." ≪ endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return ret_val;
} /* if (ret_val ≠ 0) */
```

659.

```
< Scan_Parse_Parameter_Type :: server_action_add_handle_value definition 655 > +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ thread_str ≪ "Exiting 'Scan_Parse_Parameter_Type::server_action_add_handl\
            e_value' successfully" ≪ "with return value 0." ≪ endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0;
/* End of Scan_Parse_Parameter_Type :: server_action_add_handle_value definition */
```

660. *server_action_delete_handle*. [LDF 2013.07.04.]

Log

[LDF 2013.07.04.] Added this function.

```
< Scan_Parse_Parameter_Type::server_action_delete_handle definition 660 > ≡
int Scan_Parse_Parameter_Type::server_action_delete_handle(Response_Type &response){ bool
    DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    char buffer[BUFFER_SIZE];
    char *buffer_ptr = buffer;
    memset(buffer, 0, BUFFER_SIZE);
    stringstream temp_strm;
    string thread_str;
    temp_strm << "[Thread_" << thread_ctr << "]_";
    thread_str = temp_strm.str();
    temp_strm.str("");
    Handle_Type handle;
    int ret_val = 0;
    string handle_database = (standalone_handle) ? "handlesystem_standalone" : "handlesystem";
    int handle_rows = 0;
    int created_by_user_id = 0;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering `Scan_Parse_Parameter_Type::se\
            rver_action_delete_handle':" << endl << "'response.type' == " <<
        Response_Type::typename_map[response.type] << "(" << response.type << ")" << endl;
        response.show("response:");
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
See also sections 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673, and 674.
This code is used in section 698.
```

661.

< Scan_Parse_Parameter_Type::server_action_delete_handle definition 660 > +≡

bool immediate = response.pid_options & 1U;

662.

```

⟨ Scan_Parse_Parameter_Type :: server_action_delete_handle definition 660 ⟩ +≡
  if (¬(privileges & SUPERUSER_PRIVILEGE ∨ privileges & DELETE_HANDLES_PRIVILEGE)) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "ERROR! In 'Scan_Parse_Parameter_Type::server_action_delete\
      _handle': " ≪ endl ≪ "User doesn't have the 'delete_handles' privilege." ≪ endl;
    if (immediate)
      cerr ≪ "Not marking handle " ≪ response.pid_str ≪ "' for immediate deletion." ≪ endl;
    else
      cerr ≪ "Not marking handle " ≪ response.pid_str ≪ "' for deletion." ≪ endl;
    cerr ≪ "Will send error message to client and continue." ≪ endl;
    unlock_cerr_mutex();
    temp_strm.str("");
    temp_strm ≪ "DELETE_HANDLE_RESPONSE_1\\"" ≪ response.pid_str ≪ "\\" " ≪
      "\\"" ERROR! User does not have 'delete_handles' privilege.\\"";
    if (immediate) temp_strm ≪ "Handle not marked for immediate deletion.\\"";
    else temp_strm ≪ "Handle not marked for deletion.\\"";
    strcpy(buffer, temp_strm.str().c_str());
    buffer_ptr = buffer;
    temp_strm.str("");
    ++errors_occurred;
    ret_val = 2;
    goto FINISH;
  } /* if */
}

```

663.

```

⟨ Scan_Parse_Parameter_Type :: server_action_delete_handle definition 660 ⟩ +≡
  if (response.pid_str.empty()) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "WARNING! In 'Scan_Parse_Parameter_Type::server_action_dele\
      te_handle': " ≪ endl ≪ "' Response_Type(response)' doesn't contain sufficient data to \
      add a handle value:" ≪ endl ≪ "response.pid_str== " ≪ response.pid_str ≪ endl ≪
      "Will send error message to client and continue." ≪ endl;
    unlock_cerr_mutex();
    temp_strm.str("");
    temp_strm ≪ "DELETE_HANDLE_RESPONSE_1\\"" ≪ response.pid_str ≪ "\\" " ≪
      "\\"" PID_string empty.\\"";
    if (immediate) temp_strm ≪ "Can't mark handle for immediate deletion\\"";
    else temp_strm ≪ "Can't mark handle for deletion\\"";
    strcpy(buffer, temp_strm.str().c_str());
    buffer_ptr = buffer;
    temp_strm.str("");
    ++errors_occurred;
    ret_val = 2;
    goto FINISH;
  } /* if */
}

```

664.

```

< Scan_Parse_Parameter_Type ::server_action_delete_handle definition 660 > +≡
status = fetch_handle_from_database(response.pid_str, handle);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "ERROR! In 'Scan_Parse_Parameter_Type::server_action_delete\
_handle': " ≪ endl ≪ "'Scan_Parse_Parameter_Type::fetch_handle\
e_from_database' failed, returning " ≪ status ≪ "." ≪ endl ≪
"Failed to fetch handle object for PID \" " ≪ response.pid_str ≪ "\" " ≪
"from database." ≪ endl ≪ "Will send error message to client and continue." ≪ endl;
unlock_cerr_mutex();
temp_strm.str("");
temp_strm ≪ "DELETE HANDLE RESPONSE 1 \" " ≪ response.pid_str ≪ "\" " ≪
"\Failed to fetch handle from database.\" ";
strcpy(buffer, temp_strm.str().c_str());
buffer_ptr = buffer;
temp_strm.str("");
++errors_occurred;
ret_val = 2;
goto FINISH;
} /* if (status ≠ 0) */
#endif DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "In 'Scan_Parse_Parameter_Type::server_action_delete_handle': " ≪
        endl ≪ "'Scan_Parse_Parameter_Type::fetch_handle_from_database' suc\
ceeded, returning 0." ≪ endl;
    handle.show("handle:");
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
created_by_user_id = handle.created_by_user_id();
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "created_by_user_id=" ≪ created_by_user_id ≪ endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

665.

```

< Scan_Parse_Parameter_Type ::server_action_delete_handle definition 660 > +≡
  if (privileges & SUPERUSER_PRIVILEGE) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_delete_handle':" <<
      endl << "User'" << username << "'(user_ID'" << user_id << ")" <<
      "has_superuser_privilege." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  } /* if (privileges & SUPERUSER_PRIVILEGE) */
  else if (user_id == created_by_user_id & privileges & DELETE_HANDLES_PRIVILEGE) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_delete_handle':" <<
      endl << "User'" << username << "'(user_ID'" << user_id << ")" <<
      "created_handle_and_has\"delete_handles\"privilege." << endl <<
      "Will_delete_handle." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  } /* else if (user_id == created_by_user_id & privileges & DELETE_HANDLES_PRIVILEGE) */
  else if (user_id != created_by_user_id) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::server_action_delete\
      _handle':" << endl << "User'" << username << "'(user_ID'" << user_id <<
      ") did not create handle" << "and does not have \"superuser\" privilege." <<
      endl << "'user_id' == " << user_id << endl << "'created_by_user_id' == " <<
      created_by_user_id << endl << "Will send error message to client and continue." << endl;
    unlock_cerr_mutex();
    temp_strm.str("");
    temp_strm << "DELETE_HANDLE_RESPONSE_1\"" << response.pid_str << "\" " <<
      "\"Authorization_failure: User not permitted to delete handle\"";
    strcpy(buffer, temp_strm.str().c_str());
    buffer_ptr = buffer;
    temp_strm.str("");
    ++errors_occurred;
    ret_val = 2;
    goto FINISH;
  } /* else if (user_id != created_by_user_id) */
else {
  lock_cerr_mutex();
  cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::server_action_delete\
    _handle':" << endl << "User'" << username << "'(user_ID'" << user_id <<
    ") does not have" << "\"delete handle\" privilege." << endl <<
    "Will send error message to client and continue." << endl;
  unlock_cerr_mutex();
  temp_strm.str("");
}

```

```

temp_strm << "DELETE_HANDLE_RESPONSE_1\"" << response.pid_str << "\" "
    "\Authorization_failure:\\User\\not\\permitted\\to\\delete\\handle\"";
strcpy(buffer, temp_strm.str().c_str());
buffer_ptr = buffer;
temp_strm.str("");
++errors_occurred;
ret_val = 2;
goto FINISH;
} /* else if (user_id != created_by_user_id) */

```

666. !! TODO: LDF 2013.07.17. Check *owner* and *group* fields in handle, once I've defined them. That is, they don't exist yet. [LDF 2013.07.17.]

⟨ Scan_Parse_Parameter_Type ::server_action_delete_handle definition 660 ⟩ +≡

667.

Log

[LDF 2013.08.21.] Added this section.

⟨ Scan_Parse_Parameter_Type ::server_action_delete_handle definition 660 ⟩ +≡

```

#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        handle.show("handle:");
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    if (handle.handle_value_map.size() == 0) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR!\\In\\`Scan_Parse_Parameter_Type::server_action_delete\\
            _handle':\\n" << endl << "Handle\\\" " << response.pid_str << "'\\has\\no\\handle\\values." <<
            endl << "Will\\send\\error\\message\\to\\client\\and\\continue." << endl;
        unlock_cerr_mutex();
        temp_strm.str("");
        temp_strm << "DELETE_HANDLE_RESPONSE_1\"" << response.pid_str << "\" "
            "\Handle\\has\\no\\handle\\values.\"";
        strcpy(buffer, temp_strm.str().c_str());
        buffer_ptr = buffer;
        temp_strm.str("");
        ++errors_occurred;
        ret_val = 2;
        goto FINISH;
    } /* if (handle.handle_value_map.size() == 0) */

```

668.

Log

[LDF 2013.08.21.] Added this section.

```
⟨ Scan_Parse_Parameter_Type :: server_action_delete_handle definition 660 ⟩ +≡
else
  if (handle.handle_value_map.begin()→second.marked_for_deletion ≡ true) {
    lock_cerr_mutex();
    cerr << thread_str << "WARNING! In 'Scan_Parse_Parameter_Type' "
      ::server_action_delete_handle' :" << endl << "Handle" <<
    response.pid_str << "' has already been marked for deletion." << endl <<
    "Will send warning message to client and continue." << endl;
    unlock_cerr_mutex();
    temp_strm.str("");
    temp_strm << "DELETE_HANDLE_RESPONSE_3\\"" << response.pid_str << "\""
      << "\Handle already marked for deletion.\\"";
    strcpy(buffer, temp_strm.str().c_str());
    buffer_ptr = buffer;
    temp_strm.str("");
    ++warnings_occurred;
    ret_val = 2;
    goto FINISH;
  } /* else if */
}
```

669.

```
⟨ Scan_Parse_Parameter_Type :: server_action_delete_handle definition 660 ⟩ +≡
status = handle.delete_from_database(mysql_ptr, user_id, response.pid_options, response.delay_value,
  &handle_rows, thread_ctr);
```

670.

```

⟨ Scan_Parse_Parameter_Type :: server_action_delete_handle definition 660 ⟩ +≡
  if (status ≡ 2) {
    lock_cerr_mutex();
    cerr << thread_str << "WARNING! In 'Scan_Parse_Parameter_Type::server_action_delete_handle':"
    << endl << "'Handle_Type::delete_from_database' returned 2:" << endl;
    if (immediate) cerr << "No rows for PID \" " << response.pid_str << "\" "
      << "marked for immediate deletion from database." << endl;
    else cerr << "No rows for PID \" " << response.pid_str << "\" "
      << "marked for deletion from database." << endl;
    cerr << "Will send message to client and continue." << endl;
    unlock_cerr_mutex();
    temp_strm.str("");
    temp_strm << "DELETE_HANDLE_RESPONSE_2\" " << response.pid_str << "\" ";
    if (immediate)
      temp_strm << "\"No handles marked for immediate deletion from database.\" ";
    else temp_strm << "\"No handles marked for deletion from database.\" ";
    strcpy(buffer, temp_strm.str().c_str());
    buffer_ptr = buffer;
    temp_strm.str("");
    ++warnings_occurred;
    ret_val = 0;
    goto FINISH;
  } /* if (status ≡ 2) */

```

671.

```

⟨ Scan_Parse_Parameter_Type :: server_action_delete_handle definition 660 ⟩ +≡
else
  if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "ERROR! In 'Scan_Parse_Parameter_Type::server_action_delete\
    _handle': " ≪ endl ≪ "'Handle_Type::delete_from_database' failed, returning " ≪
    status ≪ "." ≪ endl;
    if (immediate) cerr ≪ "Failed to mark handle object for PID \" " ≪ response.pid_str ≪
      "\" " ≪ "for immediate deletion from database." ≪ endl;
    else cerr ≪ "Failed to mark handle object for PID \" " ≪ response.pid_str ≪ "\" " ≪
      "for deletion from database." ≪ endl;
    cerr ≪ "Will send error message to client and continue." ≪ endl;
    unlock_cerr_mutex();
    temp_strm.str("");
    temp_strm ≪ "DELETE HANDLE RESPONSE 1 \" " ≪ response.pid_str ≪ "\" ";
    if (immediate)
      temp_strm ≪ "\" Failed to mark handle for immediate deletion from database.\" ";
    else temp_strm ≪ "\" Failed to mark handle for deletion from database.\" ";
    strcpy(buffer, temp_strm.str().c_str());
    buffer_ptr = buffer;
    temp_strm.str("");
    ++errors_occurred;
    ret_val = 2;
    goto FINISH;
  } /* if (status ≠ 0) */
#endif DEBUG_COMPILE
else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "In 'Scan_Parse_Parameter_Type::server_action_delete_handle': " ≪
      endl ≪ "'Handle_Type::delete_from_database' succeeded, returning 0." ≪ endl;
    if (immediate)
      cerr ≪ "'handle_rows' (number of rows marked for immediate deletion from " ≪ " " ≪
        handle_database ≪ ".handles' table) == ";
    else cerr ≪ "'handle_rows' (number of rows marked for deletion from " ≪ " " ≪
        handle_database ≪ ".handles' table) == ";
    cerr ≪ handle_rows ≪ endl ≪ endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
handle.clear();

```

672.

```
< Scan_Parse_Parameter_Type ::server_action_delete_handle definition 660 > +≡
temp_strm.str("");
temp_strm << "DELETE_HANDLE_RESPONSE\0" << response.pid_str << "\0";
if (immediate)
    temp_strm << "\"Marked\0" << handle_rows << "\0rows\0for\0immediate\0deletion\0from\0" << "\0" <<
        handle_database << ".handles'\0database\0table\"";
else temp_strm << "\"Marked\0" << handle_rows << "\0rows\0for\0deletion\0from\0" << "\0" <<
        handle_database << ".handles'\0database\0table\"";
strcpy(buffer, temp_strm.str().c_str());
buffer_ptr = buffer;
temp_strm.str("");
```

673.

```
< Scan_Parse_Parameter_Type ::server_action_delete_handle definition 660 > +≡
FINISH:
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "buffer\0==\0" << buffer << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    if (strlen(buffer) > 0) status = send_to_peer(&buffer_ptr, 0);
    else status = send_to_peer(0, 1);
    if (status != 0) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR!\0In\0`Scan_Parse_Parameter_Type::server_action_delete\
            _handle':\0" << endl << `Scan_Parse_Parameter_Type::send_to_peer'\0failed,\0returning\
            \0" << status << "." << endl << "Exiting\0function\0unsuccessfully\0with\0return\0value\01.\0" <<
            endl;
        unlock_cerr_mutex();
        ++errors_occurred;
        return 1;
    } /* if (status != 0) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In\0`Scan_Parse_Parameter_Type::server_action_delete_handle':\0" <<
            endl << `Scan_Parse_Parameter_Type::send_to_peer'\0succeeded,\0returning\00.\0" << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

674.

```
<Scan_Parse_Parameter_Type::server_action_delete_handle definition 660> +≡
  if (ret_val ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "In ‘Scan_Parse_Parameter_Type::server_action_delete_handle’:"
    endl ≪ "'ret_val' == " ≪ ret_val ≪ " != 0" ≪ endl ≪
    "Exiting function with return value " ≪ ret_val ≪ "." ≪ endl;
    unlock_cerr_mutex();
    return ret_val;
  }
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "Exiting ‘Scan_Parse_Parameter_Type::server_action_delete_ha\
ndle’ successfully" ≪ "with return value 0." ≪ endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
return 0; } /* End of Scan_Parse_Parameter_Type::server_action_delete_handle definition */
```

675. *server_action_undelete_handle*. [LDF 2013.08.21.]

Log

[LDF 2013.08.21.] Added this function.

```
< Scan_Parse_Parameter_Type :: server_action_undelete_handle definition 675 > ≡
int Scan_Parse_Parameter_Type :: server_action_undelete_handle(Response_Type &response){
    bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    char buffer[BUFFER_SIZE];
    char *buffer_ptr = buffer;
    memset(buffer, 0, BUFFER_SIZE);
    stringstream temp_strm;
    string thread_str;
    temp_strm << "[Thread_" << thread_ctr << "] ";
    thread_str = temp_strm.str();
    temp_strm.str("");
    Handle_Type handle;
    int ret_val = 0;
    string handle_database = (standalone_handle) ? "handlesystem_standalone" : "handlesystem";
    int handle_rows = 0;
    int created_by_user_id = 0;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering " 'Scan_Parse_Parameter_Type::se\
rver_action_undelete_handle': " << endl << "'response.type' == " <<
        Response_Type :: typename_map[response.type] << "(" << response.type << ")" << endl;
        response.show("response:");
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
See also sections 676, 677, 678, 679, 680, 681, 682, and 683.
This code is used in section 698.
```

676.

< Scan_Parse_Parameter_Type :: server_action_undelete_handle definition 675 > +≡

/* !! START HERE: LDF 2013.08.22. Check user's permissions. */

677.

```
< Scan_Parse_Parameter_Type :: server_action_undelete_handle definition 675 > +≡
if (response.pid_str.empty()) {
    lock_cerr_mutex();
    cerr << thread_str << "WARNING! In 'Scan_Parse_Parameter_Type' "
        ::server_action_undelete_handle' :" << endl <<
    "'Response_Type' response' doesn't contain sufficient data to unmark a handle" <<
    "for deletion:" << endl << "response.pid_str==" << response.pid_str << endl <<
    "Will send error message to client and continue." << endl;
unlock_cerr_mutex();
temp_strm.str("");
temp_strm << "UNDELETE_HANDLE_RESPONSE_1\" " << response.pid_str << "\" "
    "\"PID_string empty. Can't unmark handle for deletion\"";
strcpy(buffer, temp_strm.str().c_str());
buffer_ptr = buffer;
temp_strm.str("");
++warnings_occurred;
ret_val = 2;
goto FINISH;
} /* if */
```

678.

```

⟨ Scan_Parse_Parameter_Type :: server_action_undelete_handle definition 675 ⟩ +≡
status = fetch_handle_from_database(response.pid_str, handle);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::server_action_undelete_handle':"
        << endl << "'Scan_Parse_Parameter_Type::fetch_handle_from_database'" << status << "."
        << endl << "Failed to fetch handle object for PID \""
        << response.pid_str << "\" "
        << "from database." << endl << "Will send error message to client and continue."
        << endl;
unlock_cerr_mutex();
temp_strm.str("");
temp_strm << "UNDELETE_HANDLE_RESPONSE" << GW_HANDLE_NOT_FOUND << " "
<< "\" "
        << response.pid_str << "\" "
        << "\" Failed to fetch handle from database.\"";
strcpy(buffer, temp_strm.str().c_str());
buffer_ptr = buffer;
temp_strm.str("");
++errors_occurred;
ret_val = 2;
goto FINISH;
} /* if (status ≠ 0) */
#endif DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_undelete_handle':"
        << endl << "'Scan_Parse_Parameter_Type::fetch_handle_from_database' succeeded, returning 0."
        << endl;
    handle.show("handle:");
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
created_by_user_id = handle.created_by_user_id();
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "created_by_user_id=" << created_by_user_id << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */ /* !! START HERE: LDF 2013.08.22. Check user privileges. */

```

679.

```
< Scan_Parse_Parameter_Type ::server_action_undelete_handle definition 675 > +≡
status = handle.unmark_handle_for_deletion(mysql_ptr, thread_str);
if (status ≡ GW_HANDLE_NOT_MARKED_FOR_DELETION) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "WARNING! In 'Scan_Parse_Parameter_Type::server_action_undele\
    ate_handle': " ≪ endl ≪ "'Handle_Type::unmark_handle_for_deletion' returned " ≪
    GW_HANDLE_NOT_MARKED_FOR_DELETION ≪ " " ≪ "(GW_HANDLE_NOT_MARKED_FOR_DELETION)" ≪
    endl ≪ "Handle " ≪ response.pid_str ≪ "' not_marked_for_deletion." ≪ endl ≪
    "Can't unmark." ≪ endl ≪ "Will send warning message to client and continue." ≪ endl;
unlock_cerr_mutex();
temp_strm.str("");
temp_strm ≪ "UNDELETE_HANDLE_RESPONSE" ≪ GW_HANDLE_NOT_MARKED_FOR_DELETION ≪ " " ≪
    "\n" ≪ response.pid_str ≪ "\n" ≪ "\Handle not_marked_for_deletion. Can't unmark\n";
strcpy(buffer, temp_strm.str().c_str());
buffer_ptr = buffer;
temp_strm.str("");
++warnings_occurred;
ret_val = 2;
goto FINISH;
} /* if (status ≡ 2) */
```

680.

```
< Scan_Parse_Parameter_Type ::server_action_undelete_handle definition 675 > +≡
else
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "ERROR! In 'Scan_Parse_Parameter_Type::server_action_undele\
    ate_handle': " ≪ endl ≪ "'Handle_Type::unmark_handle_for_deletio\
    n' failed, returning " ≪ status ≪ ":" ≪ endl ≪ gwstrerror(status) ≪ endl ≪
    "Failed to unmark handle\n" ≪ response.pid_str ≪ "\n" ≪ "for deletion." ≪ endl ≪
    "Will send error message to client and continue." ≪ endl;
unlock_cerr_mutex();
temp_strm.str("");
temp_strm ≪ "UNDELETE_HANDLE_RESPONSE" ≪ status ≪ " " ≪ "\n" ≪ response.pid_str ≪
    "\n" ≪ "\Failed to unmark handle for deletion.\n";
strcpy(buffer, temp_strm.str().c_str());
buffer_ptr = buffer;
temp_strm.str("");
++errors_occurred;
ret_val = 2;
goto FINISH;
} /* else if (status ≠ 0) */
```

681.

```

⟨ Scan_Parse_Parameter_Type::server_action_undelete_handle definition 675 ⟩ +≡
else {
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_undelete_handle':" <<
            endl << "'Handle_Type::unmark_handle_for_deletion' succeeded, returning 0." <<
            endl << "Unmarked handle " << response.pid.str << " successfully." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
temp_strm.str("");
temp_strm << "UNDELETE_HANDLE_RESPONSE_0\" " << response.pid.str << "\"\"Success\"";
strcpy(buffer, temp_strm.str().c_str());
} /* else */

```

682.

```

⟨ Scan_Parse_Parameter_Type::server_action_undelete_handle definition 675 ⟩ +≡
FINISH:
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "buffer == " << buffer << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
if (strlen(buffer) > 0) status = send_to_peer(&buffer_ptr, 0);
else status = send_to_peer(0, 1);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::\n"
        server_action_undelete_handle':" << endl <<
        "'Scan_Parse_Parameter_Type::send_to_peer' failed, returning " <<
        status << "." << endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    return 1;
} /* if (status ≠ 0) */
#endif DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_undelete_handle':" <<
        endl << "'Scan_Parse_Parameter_Type::send_to_peer' succeeded, returning 0." << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

683.

```

⟨ Scan_Parse_Parameter_Type :: server_action_undelete_handle definition 675 ⟩ +≡
  if (ret_val ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "In ‘Scan_Parse_Parameter_Type::server_action_undelete_handle’:"
    endl ≪ "'ret_val' == " ≪ ret_val ≪ " != 0" ≪ endl ≪
    "Exiting function with return value " ≪ ret_val ≪ "." ≪ endl;
    unlock_cerr_mutex();
    return ret_val;
  }
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "Exiting ‘Scan_Parse_Parameter_Type::server_action_undelete_\
      handle’ successfully" ≪ "with return value 0." ≪ endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  return 0; } /* End of Scan_Parse_Parameter_Type::server_action_undelete_handle definition */

```

684. *server_action_delete_handle_value.* [LDF 2013.08.30.]

Log

[LDF 2013.08.30.] Added this function.

```

⟨ Scan_Parse_Parameter_Type :: server_action_delete_handle_value definition 684 ⟩ ≡
int Scan_Parse_Parameter_Type :: server_action_delete_handle_value(Response_Type &response){
    bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    char buffer[BUFFER_SIZE];
    char *buffer_ptr = buffer;
    memset(buffer, 0, BUFFER_SIZE);
    stringstream temp_strm;
    string thread_str;
    temp_strm << "[Thread_" << thread_ctr << "]_";
    thread_str = temp_strm.str();
    temp_strm.str("");
    int ret_val = 0;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering_`Scan_Parse_Parameter_Type::se\
rver_action_delete_handle_value':" << endl << `response.type' ==_"
        Response_Type :: typename_map[response.type] << "(" << response.type << ")" << endl;
        response.show("response:");
        show_privileges(privileges);
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

See also sections 685, 686, 687, and 688.

This code is used in section 698.

685.

```

⟨ Scan_Parse_Parameter_Type :: server_action_delete_handle_value definition 684 ⟩ +≡
if (¬(privileges & SUPERUSER_PRIVILEGE ∨ privileges & DELETE_HANDLE_VALUES_PRIVILEGE)) {
    lock_cerr_mutex();
    cerr << thread_str << "WARNING! In 'Scan_Parse_Parameter_Type::server_action_delete_handle_value':"
        << endl << "User" << username << ", ID" << user_id <<
        " " << "isn't doesn't have the 'DELETE_HANDLE_VALUES_PRIVILEGE'." << endl <<
        "Will send failure notice to client and continue." << endl;
    unlock_cerr_mutex();
    Response_Type new_response;
    new_response.type = Response_Type::COMMAND_ONLY_TYPE;
    temp_strm.str("");
    temp_strm << "DELETE_HANDLE_VALUE_RESPONSE" << GW_NO_PRIVILEGE_ERROR <<
        "\n" << response.string_val << "\-1\"(Type)\n" <<
        "\\"(Data)\n" << "WARNING! User" << username << ", ID" << user_id <<
        " " << "doesn't have the 'delete_handle_values' privilege.\n" <<
        (response.options & 1) << "\n" << response.delay_value << "UL" << purge_database_limit << "U";
    new_response.command = temp_strm.str();
    response_deque.push_back(new_response);
    ++warnings_occurred;
    ret_val = 2;
    goto FINISH;
} /* if */

```

686.

```

⟨ Scan_Parse_Parameter_Type :: server_action_delete_handle_value definition 684 ⟩ +≡
status = Handle_Value_Type::delete_handle_value(mysql_ptr, response.string_val, response_deque,
                                                user_id, username, privileges, response.options, response.delay_value, thread_str);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::Handle_Value_Type::delete_handle_value':"
        " failed, returning" << status << "." << endl;
    unlock_cerr_mutex();
    ret_val = 2;
    goto FINISH;
} /* if (status ≠ 0) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_delete_handle_value':"
            "'Handle_Value_Type::delete_handle_value' succeeded, returning 0."
            " Deleted handle value successfully." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

687.

```

⟨ Scan_Parse_Parameter_Type :: server_action_delete_handle_value definition 684 ⟩ +≡
FINISH:
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "buffer=="
        buffer << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    if (strlen(buffer) > 0) status = send_to_peer(&buffer_ptr, 0);
    else status = send_to_peer(0, 1);
    if (status ≠ 0) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::"
        server_action_delete_handle_value':"
        << endl <<
        "'Scan_Parse_Parameter_Type::send_to_peer' failed, returning"
        status << "."
        << endl << "Exiting function unsuccessfully with return value 1."
        << endl;
        unlock_cerr_mutex();
        ++errors_occurred;
    }
    return 1;
} /* if (status ≠ 0) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_delete_handle_"
        "value':"
        << endl << "'Scan_Parse_Parameter_Type::send_to_peer' succeeded, return"
        "ing 0."
        << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

688.

```
< Scan_Parse_Parameter_Type :: server_action_delete_handle_value definition 684 > +≡
  if (ret_val ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "In ‘Scan_Parse_Parameter_Type::server_action_delete_handle＼
      value’: " ≪ endl ≪ "'ret_val' == " ≪ ret_val ≪ " (!= 0)" ≪ endl ≪
      "Exiting function with return value " ≪ ret_val ≪ ". " ≪ endl;
    unlock_cerr_mutex();
    return ret_val;
  }
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "Exiting ‘Scan_Parse_Parameter_Type::server_action_delete_ha＼
      ndle_value’ " ≪ "successfully with return value 0." ≪ endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  return 0;
/* End of Scan_Parse_Parameter_Type :: server_action_delete_handle_value definition */
```

689. *server_action_undelete_handle_value.* [LDF 2013.08.30.]

Log

[LDF 2013.08.30.] Added this function.

```

⟨ Scan_Parse_Parameter_Type :: server_action_undelete_handle_value definition 689 ⟩ ≡
int Scan_Parse_Parameter_Type :: server_action_undelete_handle_value(Response_Type &response){
    bool DEBUG = true;      /* false */
    set_debug_level(DEBUG, 0, 0);
    int status;
    char buffer[BUFFER_SIZE];
    char *buffer_ptr = buffer;
    memset(buffer, 0, BUFFER_SIZE);
    stringstream temp_strm;
    string thread_str;
    temp_strm << "[Thread_" << thread_ctr << "] ";
    thread_str = temp_strm.str();
    temp_strm.str("");
    int ret_val = 0;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering 'Scan_Parse_Parameter_Type::se\
rver_action_undelete_handle_value': " << endl << "'response.type' == " \
Response_Type :: typename_map[response.type] << "(" << response.type << ")" << endl;
        response.show("response:");
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
See also sections 690, 691, 692, and 693.
This code is used in section 698.

```

690.

```

⟨ Scan_Parse_Parameter_Type :: server_action_undelete_handle_value definition 689 ⟩ +≡
if (¬(privileges & SUPERUSER_PRIVILEGE ∨ privileges & UNDELETE_HANDLE_VALUES_PRIVILEGE)) {
    lock_cerr_mutex();
    cerr << thread_str << "WARNING! In 'Scan_Parse_Parameter_Type::server_action_undelete_handle_value':"
        << endl << "User " << username << ", ID " << user_id <<
        " " << "isn't have the 'UNDELETE_HANDLE_VALUES_PRIVILEGE'." << endl <<
        "Will send failure notice to client and continue." << endl;
    unlock_cerr_mutex();
    Response_Type new_response;
    new_response.type = Response_Type::COMMAND_ONLY_TYPE;
    temp_strm.str("");
    temp_strm << "UNDELETE_HANDLE_VALUE_RESPONSE" << GW_NO_PRIVILEGE_ERROR <<
        " " << "\\" << response.string_val << "\"-1\"(Type)\" " <<
        "\\"(Data)\\" " << "WARNING! User " << username << ", ID " << user_id <<
        " " << "doesn't have the 'undelete_handle_values' privilege."";
    new_response.command = temp_strm.str();
    response_deque.push_back(new_response);
    ++warnings_occurred;
    ret_val = 2;
    goto FINISH;
} /* if */

```

691.

```

⟨ Scan_Parse_Parameter_Type :: server_action_undelete_handle_value definition 689 ⟩ +≡
status = Handle_Value_Type::unmark_handle_value_for_deletion(mysql_ptr, response.string_val,
    response_deque, user_id, username, privileges, thread_str);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::server_action_undelete_handle_value':"
        << endl << "'Handle_Value_Type::unmark_handle_value_for_deletion' failed, returning "
        status << endl;
    unlock_cerr_mutex();
    ret_val = 2;
} /* if (status ≠ 0) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_undelete_handle_value':"
            << endl << "'Handle_Value_Type::unmark_handle_value_for_deletion' succeeded, returning 0."
            << endl << "Undeleted handle value successfully." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

692.

```

⟨ Scan_Parse_Parameter_Type :: server_action_delete_handle_value definition 689 ⟩ +≡
FINISH:
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "buffer==" << buffer << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    if (strlen(buffer) > 0) status = send_to_peer(&buffer_ptr, 0);
    else status = send_to_peer(0, 1);
    if (status ≠ 0) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'Scan_Parse_Parameter_Type::"
            "server_action_delete_handle_value':" << endl <<
            "'Scan_Parse_Parameter_Type::send_to_peer' failed, returning"
            " " << endl << "Exiting function unsuccessfully with return value"
            " 1." << endl;
        unlock_cerr_mutex();
        ++errors_occurred;
        return 1;
    } /* if (status ≠ 0) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_a"
            "ction_delete_handle_value':" << endl <<
            "'Scan_Parse_Parameter_Type::send_to_peer' succeeded, returning"
            " 0." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

693.

```
< Scan_Parse_Parameter_Type :: server_action_undelete_handle_value definition 689 > +≡
  if (ret_val ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "In ‘Scan_Parse_Parameter_Type::server_action_undelete_handl\
e_value’: " ≪ endl ≪ "'ret_val' == " ≪ ret_val ≪ " (!= 0)" ≪ endl ≪
    "Exiting function with return value " ≪ ret_val ≪ ". " ≪ endl;
    unlock_cerr_mutex();
    return ret_val;
  }
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "Exiting ‘Scan_Parse_Parameter_Type::server_action_undelete_\\
handle_value’ " ≪ "successfully" ≪ "with return value 0." ≪ endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  return 0;
/* End of Scan_Parse_Parameter_Type :: server_action_undelete_handle_value definition */
```

694. *server_action_undelete_file*. [LDF 2013.08.15.]

Log

[LDF 2013.08.15.] Added this function.

```

⟨ Scan_Parse_Parameter_Type :: server_action_undelete_file definition 694 ⟩ ≡
int Scan_Parse_Parameter_Type :: server_action_undelete_file(Response_Type &response){ bool
    DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    char buffer[BUFFER_SIZE];
    char *buffer_ptr = buffer;
    memset(buffer, 0, BUFFER_SIZE);
    stringstream temp_strm;
    string thread_str;
    temp_strm << "[Thread_" << thread_ctr << "] ";
    thread_str = temp_strm.str();
    temp_strm.str("");
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering 'Scan_Parse_Parameter_Type::se\
rver_action_undelete_file': " << endl << "'response.type' == " <<
        Response_Type::typename_map[response.type] << "(" << response.type << ")" << endl;
        response.show("response:");
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

See also sections 695 and 696.

This code is used in section 698.

695.

```
< Scan_Parse_Parameter_Type :: server_action_undelete_file definition 694 > +≡
status = undelete_files(response, thread_str);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "ERROR! In Scan_Parse_Parameter_Type::server_action_undelet\
e_file:" ≪ endl ≪ "'Scan_Parse_Parameter_Type::undelete_files' failed, returning\
ng" ≪ status ≪ "." ≪ endl ≪ "Continuing." ≪ endl;
    unlock_cerr_mutex();
    ++errors_occurred;
} /* if (status ≠ 0) */
else {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ thread_str ≪ "In Scan_Parse_Parameter_Type::server_action_undelete_file:" ≪
            endl ≪ "'Scan_Parse_Parameter_Type::undelete_files' succeeded, returning 0." ≪
            endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
}
```

696.

```

< Scan_Parse_Parameter_Type::server_action_undelete_file definition 694 > +≡
status = send_to_peer(0, 1);      /* Scan_Parse_Parameter_Type::undelete_files takes care of creating
any responses to send to the client. [LDF 2013.08.16.] */
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "ERROR! In 'Scan_Parse_Parameter_Type::server_action_undelete_file':"
    ≪ endl ≪ "'Scan_Parse_Parameter_Type::send_to_peer' failed, returning "
    ≪ status ≪ ". " ≪ endl ≪ "Exiting function unsuccessfully with return value 1. "
    ≪ endl;
    unlock_cerr_mutex();
    ++errors_occurred;
    return 1;
}      /* if (status ≠ 0) */
#endif /* DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "In 'Scan_Parse_Parameter_Type::server_action_undelete_file':"
    ≪ endl ≪ "'Scan_Parse_Parameter_Type::send_to_peer' succeeded, returning 0. " ≪ endl;
    unlock_cerr_mutex();
}      /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
#endif /* DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "Exiting 'Scan_Parse_Parameter_Type::server_action_undelete_file':"
    ≪ endl ≪ "'Scan_Parse_Parameter_Type::server_action_undelete_file' successfully "
    ≪ "with return value 0. " ≪ endl;
    unlock_cerr_mutex();
}      /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
return 0; }      /* End of Scan_Parse_Parameter_Type::server_action_undelete_file definition */

```

697.

```

⟨ Scan_Parse_Parameter_Type :: server_action_unknown definition 697 ⟩ ≡
int Scan_Parse_Parameter_Type :: server_action_unknown(Response_Type &response)
{
    bool DEBUG = false;      /* true */
    set_debug_level(DEBUG, 0, 0);

    int status;
    char buffer[BUFFER_SIZE];
    char *buffer_ptr = buffer;
    memset(buffer, 0, BUFFER_SIZE);

    stringstream temp_strm;
    string thread_str;

    temp_strm << " [Thread_" << thread_ctr << "] ";
    thread_str = temp_strm.str();
    temp_strm.str("");

#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering 'Scan_Parse_Parameter_Type::server_action_unknown':"
            endl << "'response.type' == " << Response_Type::typename_map[response.type] <<
            "(" << response.type << ")" << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    lock_cerr_mutex();
    cerr << thread_str << "WARNING! In 'Scan_Parse_Parameter_Type\\"
        :: server_action_unknown': " << "'response.type' == "
        Response_Type::typename_map[response.type] << "(" << response.type << ")" << endl <<
        "This case hasn't been accounted for yet. Sending a single NULL byte to "
        "client (so it won't block) and continuing." << endl;
    unlock_cerr_mutex();
    ++warnings_occurred;
    memset(buffer, 0, BUFFER_SIZE);

    if (remote_connection == true) {
        status = gnutls_record_send(session, buffer, 1);
    }
    else {
        status = send(sock, buffer, 1, 0);
    }
    if (status == -1) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! " << "In 'Scan_Parse_Parameter_Type::server_a\\"
            "ction_unknown': " << "'send' failed, returning -1." << endl << "send_error: "
            strerror(errno) << endl << "Exiting function unsuccessfully with return value 1." <<
            endl;
        unlock_cerr_mutex();
        ++errors_occurred;
        return 1;
    } /* if (status == -1) */
#endif DEBUG_COMPILE
    else

```

```

if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Scan_Parse_Parameter_Type::server_action_unknown':"
    << "'send' succeeded, returning" << status << "."
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "Exiting 'Scan_Parse_Parameter_Type::server_action_unknown' "
    << "successfully" << "with return value 0."
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
return 0;
} /* End of Scan_Parse_Parameter_Type::server_action_unknown definition */

```

This code is used in section 698.

698.

```

< Include files 3>
using namespace std;
using namespace gwrdfpk;
< Scan_Parse_Parameter_Type::server_action_command_only definition 565 >
< Scan_Parse_Parameter_Type::server_action_send_file definition 568 >
< Scan_Parse_Parameter_Type::server_action_receive_put_file definition 571 >
< Scan_Parse_Parameter_Type::server_action_receive_metadata_file definition 572 >
< Scan_Parse_Parameter_Type::server_action_send_handle definition 573 >
< Scan_Parse_Parameter_Type::server_action_ls definition 592 >
< Scan_Parse_Parameter_Type::server_action_pwd definition 595 >
< Scan_Parse_Parameter_Type::server_action_cd definition 598 >
< Scan_Parse_Parameter_Type::server_action_mkdir definition 601 >
< Scan_Parse_Parameter_Type::server_action_get definition 607 >
< Scan_Parse_Parameter_Type::server_action_mark_irods_objects_for_deletion definition 604 >
< Scan_Parse_Parameter_Type::server_action_send_metadata definition 614 >
< Scan_Parse_Parameter_Type::server_action_end_server definition 618 >
< Scan_Parse_Parameter_Type::server_action_sleep definition 624 >
< Scan_Parse_Parameter_Type::server_action_show_certificate definition 631 >
< Scan_Parse_Parameter_Type::server_action_get_metadata definition 634 >
< Scan_Parse_Parameter_Type::server_action_get_handle definition 637 >
< Scan_Parse_Parameter_Type::server_action_send_tan_list definition 641 >
< Scan_Parse_Parameter_Type::server_action_process_pending definition 644 >
< Scan_Parse_Parameter_Type::server_action_get_user_info definition 649 >
< Scan_Parse_Parameter_Type::server_action_create_handle definition 652 >
< Scan_Parse_Parameter_Type::server_action_add_handle_value definition 655 >
< Scan_Parse_Parameter_Type::server_action_delete_handle definition 660 >
< Scan_Parse_Parameter_Type::server_action_undelete_handle definition 675 >
< Scan_Parse_Parameter_Type::server_action_delete_handle_value definition 684 >
< Scan_Parse_Parameter_Type::server_action_undelete_handle_value definition 689 >
< Scan_Parse_Parameter_Type::server_action_undelete_file definition 694 >
< Scan_Parse_Parameter_Type::server_action_unknown definition 697 >

```

699. This is what's written to the header file **srvractn.h**. [LDF 2013.05.29.]

```
< srvractn.h 699 >≡  
#ifndef SRVRACTN_H  
#define SRVRACTN_H 1      /* Empty */  
#endif
```

700. Types for X.509 Certificates (x509cert.web).

701. Include files.

```
<Include files 3> +≡
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <math.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include <gcrypt.h> /* for gcry-control */
#include <gnutls/gnutls.h>
#include <expat.h>
#include <iomanip>
#include <iostream>
#include <map>
#include <fstream>
#include <sstream>
#include <string>
#include <set>
#include <vector>
#include <deque>
#include <stack>
#include <pthread.h>
#if HAVE_CONFIG_H
#include <config.h>
#endif
#include <mysql.h>
#include "glblcnst.h++"
#include "glblvrbl.h++"
#include "excptntp.h++"
#include "utilfncts.h++"
#include "parser.h++"
#include "scanner.h++"
#include "hdlvltp.h++"
#include "rspnstp.h++"
#include "irdsavtp.h++"
#include "irdsobtp.h++"
#include "hdltype.h++"
#include "dcmtddtp.h++"
#ifndef _XOPEN_SOURCE
#define _XOPEN_SOURCE
#endif
```

702. **X509_Cert_Type** declaration. [LDF 2009.12.22.]

Log

[LDF 2009.12.22.] Added this **struct** declaration.

[LDF 2009.12.28.] Now conditionally compiling this **struct** declaration. This is necessary, because **struct X509_Cert_Type** must be known when **scprpmtp.h++** is included. This means that **x509cert.h++** must be included before **scprpmtp.h++** is and this **struct** declaration is compiled before this code is reached. If this code is then compiled, this causes an error because of the multiple declaration. The forward declaration of **X509_Cert_Type** in **gblvrb1.h++** does not suffice, because a **X509_Cert_Type** object is used in **Scan_Parse_Parameter_Type::get_database_username**. I don't like doing this in such a roundabout way, but I prefer it to putting the function declarations in separate files. I can't think of any better alternatives.

[LDF 2009.12.29.] Added **unsigned int user_id**.

[LDF 2012.02.09.] Changed **X509_Cert_Type** from a **struct** to a **class**. Added **friend** declarations. All data members and member functions are **private**. !! TODO: Add functions for accessing data members and see if I can get rid of some of the **friend** declarations.

[LDF 2013.05.08.] Added **friend** declaration for *extract_dn_fields*.

[LDF 2013.05.15.] Added data members **unsigned int certificate_id**, **unsigned int issuer_cert_id**, **bool is_ca** and **bool is_proxy**.

[LDF 2013.05.22.] Added **friend** declaration for *get_user_info_func*.

```
< class X509_Cert_Type declaration 702 >≡
class X509_Cert_Type {
    friend class Scan_Parse_Parameter_Type;
    friend class Distinguished_Name_Type;
    friend class User_Info_Type;
    friend int verify_certificate(gnutls_session_t session, X509_Cert_Type *x509_cert_ptr, const char
        *hostname);
    friend int extract_dn_fields(gnutls_x509_crt_t & cert, X509_Cert_Type *x509_cert, bool
        subject, Scan_Parse_Parameter_Type *param);
    friend int yyparse(yyscan_t);
    friend int get_user_info_func(Scan_Parse_Parameter_Type *, const char *);
    string organization;
    string organizationalUnitName;
    string commonName;
    string countryName;
    string localityName;
    string stateOrProvinceName;
    unsigned long int serialNumber;
    time_t Validity_notBefore;
    time_t Validity_notAfter;
    X509_Cert_Type *issuer_cert;
    string user_name;
    unsigned int user_id;
    unsigned int certificate_id;
    unsigned int issuer_cert_id;
    bool is_ca;
    bool is_proxy;
public: < X509_Cert_Type function declarations 705 >
};
```

This code is used in sections 748 and 749.

703. **X509_Cert_Type** functions. [LDF 2009.12.22.]

Log

[LDF 2009.12.22.] Added this section.

704. Constructors and Setting Functions. [LDF 2009.12.22.]

Log

[LDF 2009.12.22.] Added this section.

705. Default constructor. [LDF 2009.12.22.]

Log

[LDF 2009.12.22.] Added this function.

⟨X509_Cert_Type function declarations 705⟩ ≡**X509_Cert_Type(void);**

See also sections 707, 709, 711, 713, 730, 732, 734, 743, and 745.

This code is used in section 702.

706.**⟨X509_Cert_Type constructor definitions 706⟩ ≡****X509_Cert_Type::X509_Cert_Type(void)**

{

```

    issuer_cert = 0;
    serialNumber = 0;
    certificate_id = issuer_cert_id = user_id = 0;
    user_name = "";
    Validity_notBefore = 0;
    Validity_notAfter = 0;
    is_ca = is_proxy = false;
    return;
}
```

/* End of **X509_Cert_Type** default constructor definition */

See also sections 708 and 710.

This code is used in section 748.

707. Copy constructor. [LDF 2013.05.15.]

Log

[LDF 2013.05.15.] Added this function.

⟨X509_Cert_Type function declarations 705⟩ +≡**X509_Cert_Type(const X509_Cert_Type &cert);**

708.

```
<X509_Cert_Type constructor definitions 706 > +≡
X509_Cert_Type::X509_Cert_Type(const X509_Cert_Type &cert)
{
    organization = cert.organization;
    organizationalUnitName = cert.organizationalUnitName;
    commonName = cert.commonName;
    countryName = cert.countryName;
    localityName = cert.localityName;
    stateOrProvinceName = cert.stateOrProvinceName;
    serialNumber = cert.serialNumber;
    Validity_notBefore = cert.Validity_notBefore;
    Validity_notAfter = cert.Validity_notAfter;
    user_name = cert.user_name;
    user_id = cert.user_id;
    is_ca = cert.is_ca;
    is_proxy = cert.is_proxy;
    issuer_cert = cert.issuer_cert;
    certificate_id = cert.certificate_id;
    issuer_cert_id = cert.issuer_cert_id;
    return;
} /* End of X509_Cert_Type copy constructor definition */
```

709. Constructor with arguments. [LDF 2009.12.22.]

Log

[LDF 2009.12.22.] Added this function.

```
<X509_Cert_Type function declarations 705 > +≡
X509_Cert_Type(unsigned long int sserialNumber, X509_Cert_Type *iissuer_cert = 0, string
organization = "", string oorganizationalUnitName = "", string ccommonName = "", string
ccountryName = "", string llocalityName = "", string sstateOrProvinceName = "", unsigned
int uuser_id = 0, string uuser_name = "", time_t VValidity_notBefore = 0, time_t
VValidity_notAfter = 0, bool iis_ca = false, bool iis_proxy = false, unsigned int
ccertificate_id = 0, unsigned int iissuer_cert_id = 0);
```

710.

```
<X509_Cert_Type constructor definitions 706 > +≡
X509_Cert_Type::X509_Cert_Type(unsigned long int sserialNumber, X509_Cert_Type
*iissuer_cert, string oorganization, string oorganizationalUnitName, string ccommonName,
string ccountryName, string llocalityName, string sstateOrProvinceName, unsigned
int uuser_id, string uuser_name, time_t VValidity_notBefore, time_t VValidity_notAfter, bool
iis_ca, bool iis_proxy, unsigned int ccertificate_id, unsigned int iissuer_cert_id)
: serialNumber(sserialNumber), issuer_cert(iissuer_cert),
organization(oorganization), organizationalUnitName(oorganizationalUnitName),
commonName(ccommonName), countryName(ccountryName), localityName(llocalityName),
stateOrProvinceName(sstateOrProvinceName), user_id(uuser_id), user_name(uuser_name),
Validity_notBefore(VValidity_notBefore), Validity_notAfter(VValidity_notAfter), is_ca(iis_ca),
is_proxy(iis_proxy), certificate_id(ccertificate_id), issuer_cert_id(iissuer_cert_id) {
return;
} /* End of X509_Cert_Type non-default constructor definition */
```

711. Setting function with multiple arguments. [LDF 2009.12.22.]

Log

[LDF 2009.12.22.] Added this function.

`(X509_Cert_Type function declarations 705) +≡`

```
int set(unsigned long int sserialNumber, X509_Cert_Type *iissuer_cert = 0, string
       oorganization = "", string oorganizationalUnitName = "", string ccommonName = "", string
       ccountryName = "", string llocalityName = "", string sstateOrProvinceName = "", unsigned
       int uuser_id = 0, string uuser_name = "", time_t VValidity_notBefore = 0, time_t
       VValidity_notAfter = 0, bool iis_ca = false, bool iis_proxy = false, unsigned int
       ccertificate_id = 0, unsigned int iissuer_cert_id = 0);
```

712.

`(X509_Cert_Type::set definitions 712) ≡`

```
int X509_Cert_Type::set(unsigned long int sserialNumber, X509_Cert_Type *iissuer_cert, string
                        oorganization, string oorganizationalUnitName, string ccommonName, string
                        ccountryName, string llocalityName, string sstateOrProvinceName, unsigned int
                        uuser_id, string uuser_name, time_t VValidity_notBefore, time_t VValidity_notAfter, bool
                        iis_ca, bool iis_proxy, unsigned int ccertificate_id, unsigned int iissuer_cert_id)
{
    serialNumber = sserialNumber;
    issuer_cert = iissuer_cert;
    organization = oorganization;
    organizationalUnitName = oorganizationalUnitName;
    commonName = ccommonName;
    countryName = ccountryName;
    localityName = llocalityName;
    stateOrProvinceName = sstateOrProvinceName;
    user_id = uuser_id;
    user_name = uuser_name;
    Validity_notBefore = VValidity_notBefore;
    Validity_notAfter = VValidity_notAfter;
    is_ca = iis_ca;
    is_proxy = iis_proxy;
    certificate_id = ccertificate_id;
    issuer_cert_id = iissuer_cert_id;
    return 0;
} /* End of X509_Cert_Type::set definition */
```

See also sections 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, and 729.

This code is used in section 748.

713. Setting function with MYSQL_ROW & argument. [LDF 2013.05.15.]

Log

[LDF 2013.05.15.] Added this function.

`(X509_Cert_Type function declarations 705) +≡`

```
int set(MYSQL_ROW & row, string thread_ctr_str = "");
```

714.

```
<X509_Cert_Type::set definitions 712> +≡
int X509_Cert_Type::set(MYSQL_ROW &row, string thread_ctr_str){ bool DEBUG = false;
    /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    unsigned long int temp_val = 0;
    time_t temp_time_val = 0;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "Entering 'X509_Cert_Type::set'." << endl;
        unlock_cerr_mutex();
    }      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */
    int field_ctr = 0;
```

715. *certificate_id*.

```

⟨X509_Cert_Type::set definitions 712⟩ +≡
  if (row[field_ctr] ∧ strlen(row[field_ctr]) > 0) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'X509_Cert_Type::set':" << endl << "'row[" << field_ctr <<
      "]' == " << row[field_ctr] << "(certificate_id)" << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  errno = 0;
  temp_val = strtoul(row[field_ctr], 0, 10);
  if (temp_val ≡ ULONG_MAX) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'X509_Cert_Type::set':" << endl <<
      "'strtoul' failed, returning 'ULONG_MAX': " << endl << strerror(errno) <<
      endl << "Failed to retrieve 'certificate_id'." << endl <<
      "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    return 1;
  } /* if (temp_val ≡ ULONG_MAX) */
  certificate_id = temp_val;
#endif DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'X509_Cert_Type::set':" << endl << "'certificate_id'" == " <<
      certificate_id << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (row[field_ctr] ∧ strlen(row[field_ctr]) > 0) */
else /* row[field_ctr] ≡ 0 ∨ strlen(row[field_ctr]) ≡ 0 */
{
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'X509_Cert_Type::set':" << endl << "'row[" << field_ctr <<
      "]' == 0" << " or 'strlen(row[field_ctr])' == 0 (certificate_id)" << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  lock_cerr_mutex();
  cerr << thread_ctr_str << "ERROR! In 'X509_Cert_Type::set':" << endl << "'row[" << field_ctr <<
    "]' == 0" << " or 'strlen(row[field_ctr])' == 0 (certificate_id)" <<
    endl << "Failed to retrieve 'certificate_id'." << endl <<
    "Exiting function unsuccessfully with return value 1." << endl;
  unlock_cerr_mutex();
  return 1;
} /* else (row[field_ctr] ≡ 0 ∨ strlen(row[field_ctr]) ≡ 0) */

```

716. *user_id*.

```

⟨X509_Cert_Type::set definitions 712⟩ +≡
++field_ctr;
  if (row[field_ctr] ∧ strlen(row[field_ctr]) > 0) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'X509_Cert_Type::set':" << endl << "'row[" << field_ctr <<
      "]' " << row[field_ctr] << "(user_id)" << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  errno = 0;
  temp_val = strtoul(row[field_ctr], 0, 10);
  if (temp_val ≡ ULONG_MAX) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'X509_Cert_Type::set':" <<
      endl << "'strtoul' failed, returning 'ULONG_MAX':" << endl <<
      strerror(errno) << endl << "Failed to retrieve 'user_id'." << endl <<
      "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    return 1;
  } /* if (temp_val ≡ ULONG_MAX) */
  user_id = temp_val;
#endif DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'X509_Cert_Type::set':" << endl << "'user_id' " << user_id <<
      endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (row[field_ctr] ∧ strlen(row[field_ctr]) > 0) */
else /* row[field_ctr] ≡ 0 ∨ strlen(row[field_ctr]) ≡ 0 */
{
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'X509_Cert_Type::set':" << endl << "'row[" << field_ctr <<
      "]' " << 0 << "'strlen(row[field_ctr])' " << 0 << "(user_id)" << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  lock_cerr_mutex();
  cerr << thread_ctr_str << "ERROR! In 'X509_Cert_Type::set':" << endl << "'row[" <<
    field_ctr << "]' " << 0 << "'strlen(row[field_ctr])' " << 0 << "(user_id)" <<
    endl << "Failed to retrieve 'user_id'." << endl <<
    "Exiting function unsuccessfully with return value 1." << endl;
  unlock_cerr_mutex();
  return 1;
} /* else (row[field_ctr] ≡ 0 ∨ strlen(row[field_ctr]) ≡ 0) */

```

717. *user_name*.

```

⟨X509_Cert_Type::set definitions 712⟩ +≡
++field_ctr;
if (row[field_ctr] ∧ strlen(row[field_ctr]) > 0) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "In 'X509_Cert_Type::set':" << endl << "row[" << field_ctr <<
        "]' == " << row[field_ctr] << "(user_name)" << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    user_name = row[field_ctr];
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'X509_Cert_Type::set':" << endl << "user_name'" == " <<
    user_name << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (row[field_ctr] ∧ strlen(row[field_ctr]) > 0) */
else /* row[field_ctr] ≡ 0 ∨ strlen(row[field_ctr]) ≡ 0 */
{
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "In 'X509_Cert_Type::set':" << endl << "row[" << field_ctr <<
        "]' == 0 or 'strlen(row[field_ctr])' == 0 (user_name)" << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'X509_Cert_Type::set':" << endl << "row[" <<
    field_ctr << "]' == 0 or 'strlen(row[field_ctr])' == 0 (user_name)" <<
    endl << "Failed to retrieve 'user_name'." << endl <<
    "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    return 1;
} /* else (row[field_ctr] ≡ 0 ∨ strlen(row[field_ctr]) ≡ 0) */
}

```

718. *issuer_cert_id*.

```

⟨X509_Cert_Type::set definitions 712⟩ +≡
++field_ctr;
  if (row[field_ctr] ∧ strlen(row[field_ctr]) > 0) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'X509_Cert_Type::set':" << endl << "row[" << field_ctr <<
      "]' == " << row[field_ctr] << "(issuer_cert_id)" << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  errno = 0;
  temp_val = strtoul(row[field_ctr], 0, 10);
  if (temp_val ≡ ULONG_MAX) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'X509_Cert_Type::set':" << endl <<
      "'strtoul' failed, returning 'ULONG_MAX': " << endl << strerror(errno) <<
      endl << "Failed to retrieve 'issuer_cert_id'." << endl <<
      "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    return 1;
  } /* if (temp_val ≡ ULONG_MAX) */
  issuer_cert_id = temp_val;
#endif DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'X509_Cert_Type::set':" << endl << "issuer_cert_id" == " <<
      issuer_cert_id << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (row[field_ctr] ∧ strlen(row[field_ctr]) > 0) */
else /* row[field_ctr] ≡ 0 ∨ strlen(row[field_ctr]) ≡ 0 */
{
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'X509_Cert_Type::set':" << endl << "row[" << field_ctr <<
      "]' == 0" << " or 'strlen(row[field_ctr])' == 0 (issuer_cert_id)" << endl <<
      "Setting 'issuer_cert_id' to 0." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  issuer_cert_id = 0;
} /* else (row[field_ctr] ≡ 0 ∨ strlen(row[field_ctr]) ≡ 0) */

```

719. *is_ca*.

```

⟨X509_Cert_Type::set definitions 712⟩ +≡
++field_ctr;
  if (row[field_ctr] ∧ strlen(row[field_ctr]) > 0) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'X509_Cert_Type::set':" << endl << "row[" << field_ctr <<
    "]' == " << row[field_ctr] << "(is_ca)" << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  errno = 0;
  temp_val = strtoul(row[field_ctr], 0, 10);
  if (temp_val ≡ ULONG_MAX) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'X509_Cert_Type::set':" <<
      endl << "'strtoul' failed, returning 'ULONG_MAX':" << endl <<
      strerror(errno) << endl << "Failed to retrieve 'is_ca'." << endl <<
      "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    return 1;
  } /* if (temp_val ≡ ULONG_MAX) */
  is_ca = temp_val;
#endif DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'X509_Cert_Type::set':" << endl << "'is_ca' == " << is_ca << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (row[field_ctr] ∧ strlen(row[field_ctr]) > 0) */
else /* row[field_ctr] ≡ 0 ∨ strlen(row[field_ctr]) ≡ 0 */
{
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'X509_Cert_Type::set':" << endl << "row[" << field_ctr <<
      "]' == 0 or 'strlen(row[field_ctr])' == 0 (is_ca)" << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  is_ca = false;
} /* else (row[field_ctr] ≡ 0 ∨ strlen(row[field_ctr]) ≡ 0) */

```

720. *is_proxy*.

```

⟨X509_Cert_Type::set definitions 712⟩ +≡
++field_ctr;
if (row[field_ctr] ∧ strlen(row[field_ctr]) > 0) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "In 'X509_Cert_Type::set':" << endl << "row[" << field_ctr <<
            "]' == " << row[field_ctr] << "(is_proxy)" << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    errno = 0;
    temp_val = strtoul(row[field_ctr], 0, 10);
    if (temp_val ≡ ULONG_MAX) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "ERROR! In 'X509_Cert_Type::set':" <<
            endl << "'strtoul' failed, returning 'ULONG_MAX'" << endl <<
            strerror(errno) << endl << "Failed to retrieve 'is_proxy'." << endl <<
            "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        return 1;
    } /* if (temp_val ≡ ULONG_MAX) */
    is_proxy = temp_val;
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "In 'X509_Cert_Type::set':" << endl << "'is_proxy' == " <<
            is_proxy << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (row[field_ctr] ∧ strlen(row[field_ctr]) > 0) */
else /* row[field_ctr] ≡ 0 ∨ strlen(row[field_ctr]) ≡ 0 */
{
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "In 'X509_Cert_Type::set':" << endl << "row[" << field_ctr <<
            "]' == 0 or 'strlen(row[field_ctr])' == 0 (is_proxy)" << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    is_proxy = false;
} /* else (row[field_ctr] ≡ 0 ∨ strlen(row[field_ctr]) ≡ 0) */

```

721. *serialNumber*.

```

⟨X509_Cert_Type::set definitions 712⟩ +≡
++field_ctr;
if (row[field_ctr] ∧ strlen(row[field_ctr]) > 0) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "In 'X509_Cert_Type::set':" << endl << "'row[" << field_ctr <<
            "]' " << row[field_ctr] << "(serialNumber)" << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    errno = 0;
    temp_val = strtoul(row[field_ctr], 0, 10);
    if (temp_val ≡ ULONG_MAX) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "ERROR! In 'X509_Cert_Type::set':" <<
            endl << "'strtoul' failed, returning 'ULONG_MAX':" << endl <<
            strerror(errno) << endl << "Failed to retrieve 'serialNumber'." << endl <<
            "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        return 1;
    } /* if (temp_val ≡ ULONG_MAX) */
    serialNumber = temp_val;
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "In 'X509_Cert_Type::set':" << endl << "'serialNumber'" << endl <<
            serialNumber << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (row[field_ctr] ∧ strlen(row[field_ctr]) > 0) */
else /* row[field_ctr] ≡ 0 ∨ strlen(row[field_ctr]) ≡ 0 */
{
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "In 'X509_Cert_Type::set':" << endl << "'row[" << field_ctr <<
            "]' " << "or 'strlen(row[field_ctr])' " << "0" << "(serialNumber)" << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'X509_Cert_Type::set':" << endl << "'row[" <<
        field_ctr << "]' " << "or 'strlen(row[field_ctr])' " << "0" << "(serialNumber)" <<
        endl << "Failed to retrieve 'serialNumber'." << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    return 1;
} /* else (row[field_ctr] ≡ 0 ∨ strlen(row[field_ctr]) ≡ 0) */

```

722. *organization.*

```
< X509_Cert_Type::set definitions 712 > +≡
  ++field_ctr;
  if (row[field_ctr] ∧ strlen(row[field_ctr]) > 0) {
#if DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ thread_ctr_str ≪ "In 'X509_Cert_Type::set':" ≪ endl ≪ "'row[" ≪ field_ctr ≪
      "]' " ≪ row[field_ctr] ≪ "(organization)" ≪ endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  organization = row[field_ctr];
} /* if (row[field_ctr] ∧ strlen(row[field_ctr]) > 0) */
else /* row[field_ctr] ≡ 0 ∨ strlen(row[field_ctr]) ≡ 0 */
{
#if DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ thread_ctr_str ≪ "In 'X509_Cert_Type::set':" ≪ endl ≪ "'row[" ≪ field_ctr ≪
      "]' " ≪ "or 'strlen(row[field_ctr])' " ≪ "0" ≪ "(organization)" ≪ endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  lock_cerr_mutex();
  cerr ≪ thread_ctr_str ≪ "ERROR! In 'X509_Cert_Type::set':" ≪ endl ≪ "'row[" ≪
    field_ctr ≪ "]' " ≪ "0" ≪ " or 'strlen(row[field_ctr])' " ≪ "0" ≪ "(organization)" ≪
    endl ≪ "Failed to retrieve 'organization'." ≪ endl ≪
    "Exiting function unsuccessfully with return value 1." ≪ endl;
  unlock_cerr_mutex();
  return 1;
} /* else (row[field_ctr] ≡ 0 ∨ strlen(row[field_ctr]) ≡ 0) */
```

723. *organizationalUnitName*.

```

⟨X509_Cert_Type::set definitions 712⟩ +≡
  ++field_ctr;
  if (row[field_ctr] ∧ strlen(row[field_ctr]) > 0) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'X509_Cert_Type::set':" << endl << "'row[" << field_ctr <<
      "]' " << row[field_ctr] << "(organizationalUnitName)" << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  organizationalUnitName = row[field_ctr];
} /* if (row[field_ctr] ∧ strlen(row[field_ctr]) > 0) */
else /* row[field_ctr] ≡ 0 ∨ strlen(row[field_ctr]) ≡ 0 */
{
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'X509_Cert_Type::set':" << endl << "'row[" << field_ctr <<
      "]' " << 0 << " or " << strlen(row[field_ctr]) << 0 << "(organizationalUnitName)" << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  organizationalUnitName = ""; /* This is allowed. [LDF 2013.05.15.] */
} /* else (row[field_ctr] ≡ 0 ∨ strlen(row[field_ctr]) ≡ 0) */

```

724. *commonName*.

```
< X509_Cert_Type::set definitions 712 > +≡
  ++field_ctr;
  if (row[field_ctr] ∧ strlen(row[field_ctr]) > 0) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ thread_ctr_str ≪ "In 'X509_Cert_Type::set':" ≪ endl ≪ "'row[" ≪ field_ctr ≪
      "]' " ≪ row[field_ctr] ≪ "(commonName)" ≪ endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  commonName = row[field_ctr];
} /* if (row[field_ctr] ∧ strlen(row[field_ctr]) > 0) */
else /* row[field_ctr] ≡ 0 ∨ strlen(row[field_ctr]) ≡ 0 */
{
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ thread_ctr_str ≪ "In 'X509_Cert_Type::set':" ≪ endl ≪ "'row[" ≪ field_ctr ≪
      "]' " ≪ 0 ≪ "or " ≪ strlen(row[field_ctr]) ≪ 0 ≪ "(commonName)" ≪ endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  lock_cerr_mutex();
  cerr ≪ thread_ctr_str ≪ "ERROR! In 'X509_Cert_Type::set':" ≪ endl ≪ "'row[" ≪
    field_ctr ≪ "]' " ≪ 0 ≪ "or " ≪ strlen(row[field_ctr]) ≪ 0 ≪ "(commonName)" ≪
    endl ≪ "Failed to retrieve 'commonName'." ≪ endl ≪
    "Exiting function unsuccessfully with return value 1." ≪ endl;
  unlock_cerr_mutex();
  return 1;
} /* else (row[field_ctr] ≡ 0 ∨ strlen(row[field_ctr]) ≡ 0) */
```

725. *countryName*.

```

⟨X509_Cert_Type::set definitions 712⟩ +≡
++field_ctr;
if (row[field_ctr] ∧ strlen(row[field_ctr]) > 0) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "In 'X509_Cert_Type::set':" << endl << "'row[" << field_ctr <<
        "]' " << row[field_ctr] << "(countryName)" << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    countryName = row[field_ctr];
} /* if (row[field_ctr] ∧ strlen(row[field_ctr]) > 0) */
else /* row[field_ctr] ≡ 0 ∨ strlen(row[field_ctr]) ≡ 0 */
{
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "In 'X509_Cert_Type::set':" << endl << "'row[" << field_ctr <<
        "]' " << 0 || 'strlen(row[field_ctr])' << 0 << "(countryName)" << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'X509_Cert_Type::set':" << endl << "'row[" <<
    field_ctr << "]' " << 0 || 'strlen(row[field_ctr])' << 0 << "(countryName)" <<
    endl << "Failed to retrieve 'countryName'." << endl <<
    "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    return 1;
} /* else (row[field_ctr] ≡ 0 ∨ strlen(row[field_ctr]) ≡ 0) */

```

726. *localityName*.

```
< X509_Cert_Type::set definitions 712 > +≡
  ++field_ctr;
  if (row[field_ctr] ∧ strlen(row[field_ctr]) > 0) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In `X509_Cert_Type::set':"
      << endl << "'row[" << field_ctr <<
      "]' " << row[field_ctr] << "(localityName)" << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  localityName = row[field_ctr];
} /* if (row[field_ctr] ∧ strlen(row[field_ctr]) > 0) */
else /* row[field_ctr] ≡ 0 ∨ strlen(row[field_ctr]) ≡ 0 */
{
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In `X509_Cert_Type::set':"
      << endl << "'row[" << field_ctr <<
      "]' " << 0 << endl << "strlen(row[field_ctr])' " << 0 <<
      "(localityName)" << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  localityName = ""; /* This is allowed. [LDF 2013.05.15.] */
} /* else (row[field_ctr] ≡ 0 ∨ strlen(row[field_ctr]) ≡ 0) */
```

727. *stateOrProvinceName*.

```

⟨X509_Cert_Type::set definitions 712⟩ +≡
  ++field_ctr;
  if (row[field_ctr] ∧ strlen(row[field_ctr]) > 0) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'X509_Cert_Type::set':" << endl << "'row[" << field_ctr <<
      "]' " << row[field_ctr] << "(stateOrProvinceName)" << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  stateOrProvinceName = row[field_ctr];
} /* if (row[field_ctr] ∧ strlen(row[field_ctr]) > 0) */
else /* row[field_ctr] ≡ 0 ∨ strlen(row[field_ctr]) ≡ 0 */
{
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'X509_Cert_Type::set':" << endl << "'row[" << field_ctr <<
      "]' " << 0 << "or " << strlen(row[field_ctr]) << 0 << "(stateOrProvinceName)" << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  stateOrProvinceName = ""; /* This is allowed. [LDF 2013.05.15.] */
} /* else (row[field_ctr] ≡ 0 ∨ strlen(row[field_ctr]) ≡ 0) */

```

728. *Validity_notBefore*.

```

⟨X509_Cert_Type::set definitions 712⟩ +≡
  ++field_ctr;
  if (row[field_ctr] ∧ strlen(row[field_ctr]) > 0) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'X509_Cert_Type::set':" << endl << "row[" << field_ctr <<
    "]' == " << row[field_ctr] << "(Validity_notBefore)" << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  status = get_seconds_since_epoch(row[field_ctr], temp_time_val);
  if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'X509_Cert_Type::set':" << endl <<
      "'get_seconds_since_epoch' failed, returning " << status <<
      "." << endl << "Failed to retrieve 'Validity_notBefore'." << endl <<
      "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    return 1;
  } /* if (status ≠ 0) */
  Validity_notBefore = temp_time_val;
#endif DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'X509_Cert_Type::set':" << endl <<
      "'Validity_notBefore' == " << Validity_notBefore << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (row[field_ctr] ∧ strlen(row[field_ctr]) > 0) */
else /* row[field_ctr] ≡ 0 ∨ strlen(row[field_ctr]) ≡ 0 */
{
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'X509_Cert_Type::set':" << endl << "row[" << field_ctr <<
      "]' == 0 or 'strlen(row[field_ctr])' == 0" << "(Validity_notBefore)" << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  lock_cerr_mutex();
  cerr << thread_ctr_str << "ERROR! In 'X509_Cert_Type::set':" << endl << "row[" << field_ctr <<
    "]' == 0 or " << "'strlen(row[field_ctr])' == 0 (Validity_notBefore)" << endl <<
    endl << "Failed to retrieve 'Validity_notBefore'." << endl <<
    "Exiting function unsuccessfully with return value 1." << endl;
  unlock_cerr_mutex();
  return 1;
} /* else (row[field_ctr] ≡ 0 ∨ strlen(row[field_ctr]) ≡ 0) */

```

729. *Validity_notAfter*.

```

⟨X509_Cert_Type::set definitions 712⟩ +≡
  ++field_ctr;
  if (row[field_ctr] ∧ strlen(row[field_ctr]) > 0) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'X509_Cert_Type::set':" << endl << "'row[" << field_ctr <<
    "]' == " << row[field_ctr] << "(Validity_notAfter)" << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  status = get_seconds_since_epoch(row[field_ctr], temp_time_val);
  if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'X509_Cert_Type::set':" << endl <<
      "'get_seconds_since_epoch' failed, returning " << status <<
      "." << endl << "Failed to retrieve 'Validity_notAfter'." << endl <<
      "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    return 1;
  } /* if (status ≠ 0) */
  Validity_notAfter = temp_time_val;
#endif DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'X509_Cert_Type::set':" << endl << "'Validity_notAfter'" ==
      "Validity_notAfter" << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (row[field_ctr] ∧ strlen(row[field_ctr]) > 0) */
else /* row[field_ctr] ≡ 0 ∨ strlen(row[field_ctr]) ≡ 0 */
{
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'X509_Cert_Type::set':" << endl << "'row[" << field_ctr <<
      "]' == 0 or 'strlen(row[field_ctr])' == 0" << "(Validity_notAfter)" << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  lock_cerr_mutex();
  cerr << thread_ctr_str << "ERROR! In 'X509_Cert_Type::set':" << endl << "'row[" << field_ctr <<
    "]' == 0 or " << "'strlen(row[field_ctr])' == 0 (Validity_notAfter)" << endl <<
    "Failed to retrieve 'Validity_notAfter'." << endl <<
    "Exiting function unsuccessfully with return value 1." << endl;
  unlock_cerr_mutex();
  return 1;
} /* else (row[field_ctr] ≡ 0 ∨ strlen(row[field_ctr]) ≡ 0) */
#ifndef DEBUG_COMPILE
  if (DEBUG) {

```

```

lock_cerr_mutex();
cerr << thread_ctr_str << "Exiting 'X509_Cert_Type::set' successfully with return value\n"
    e_0. " << endl;
unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
return 0; } /* End of X509_Cert_Type::set definition */

```

730. Assignment operator. [LDF 2013.05.15.]

Log

[LDF 2013.05.15.] Added this function.

⟨ X509_Cert_Type function declarations 705 ⟩ +≡
void operator=(const X509_Cert_Type &cert);

731.

⟨ X509_Cert_Type::operator= definition 731 ⟩ ≡
void X509_Cert_Type::operator=(const X509_Cert_Type &cert)
{
 organization = cert.organization;
 organizationalUnitName = cert.organizationalUnitName;
 commonName = cert.commonName;
 countryName = cert.countryName;
 localityName = cert.localityName;
 stateOrProvinceName = cert.stateOrProvinceName;
 serialNumber = cert.serialNumber;
 Validity_notBefore = cert.Validity_notBefore;
 Validity_notAfter = cert.Validity_notAfter;
 user_name = cert.user_name;
 user_id = cert.user_id;
 is_ca = cert.is_ca;
 is_proxy = cert.is_proxy;
 issuer_cert = cert.issuer_cert;
 certificate_id = cert.certificate_id;
 issuer_cert_id = cert.issuer_cert_id;
return;
} /* End of X509_Cert_Type assignment operator definition */

This code is used in section 748.

732. Clear. [LDF 2013.05.15.]

⟨ X509_Cert_Type function declarations 705 ⟩ +≡
void clear(void);

733.

```
< X509_Cert_Type :: clear definition 733 > ≡
void X509_Cert_Type :: clear(void)
{
    user_id = 0U;
    organization = organizationalUnitName = commonName = countryName = user_name =
        localityName = stateOrProvinceName = "";
    serialNumber = 0UL;
    Validity_notBefore = Validity_notAfter = static_cast<time_t>(0);
    issuer_cert = 0;
    is_ca = is_proxy = false;
    certificate_id = issuer_cert_id = 0;
    return;
} /* End of X509_Cert_Type :: clear definition */
```

This code is used in section 748.

734. Get from database. [LDF 2013.05.17.]**Log**

[LDF 2013.05.17.] Added this function.

```
< X509_Cert_Type function declarations 705 > +≡
int get_from_database(MYSQL *mysql_ptr, unsigned int uuser_id, string thread_ctr_str = "");
```

735.

```
< X509_Cert_Type :: get_from_database definition 735 > ≡
int X509_Cert_Type :: get_from_database(MYSQL *mysql_ptr, unsigned int uuser_id, string
    thread_ctr_str) { bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status = 0;
    unsigned long temp_val = 0UL;
    user_id = uuser_id;
    stringstream sql strm;
    MYSQL_RES * result = 0;
    unsigned int row_ctr;
    unsigned int field_ctr;
    MYSQL_ROW curr_row;
#if DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ thread_ctr_str ≪ "Entering 'X509_Cert_Type::get_from_database'." ≪ endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
}
```

See also sections 736, 737, 738, 739, 740, 741, and 742.

This code is used in section 748.

736.

```

⟨ X509_Cert_Type::get_from_database definition 735 ⟩ +≡
sqlstrm << "select C.certificate_id, C.user_id, U.username, C.issuer_cert_id, C.is_ca, "
    "C.is_proxy, C.serialNumber, C.organization, C.organizationalUnitName, "
    "C.commonName, C.countryName, C.localityName, C.stateOrProvinceName, "
    "C.Validity_notBefore, C.Validity_notAfter" << "from gwiridsif.Certificates
    as C, Users as U" << "where C.user_id=U.user_id and C.user_id=u"
    "user_id << " "order by C.certificate_id desc";
#endif /* DEBUG_COMPILE */
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'X509_Cert_Type::get_from_database':" << endl <<
        "'sql.strm.str()' == " << sqlstrm.str() << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
status = submit_mysql_query(sqlstrm.str(), result, mysql_ptr, &row_ctr, &field_ctr, 0, thread_ctr_str);
if (status != 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'X509_Cert_Type::get_from_database':" <<
        endl << "'submit_mysql_query' failed, returning" << status << "."
    endl << "Failed to retrieve X.509 certificate data from database table"
    "'gwiridsif.Certificates' for user" << user_id << "."
    "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    return 1;
} /* if (status != 0) */

```

737.

```

⟨ X509_Cert_Type::get_from_database definition 735 ⟩ +≡
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "In 'X509_Cert_Type::get_from_database':" <<
            endl << "'submit_mysql_query' succeeded, returning 0."
        endl << "Retrieved X.509 certificate data from database table"
        "'gwiridsif.Certificates' for user" << user_id << " successfully."
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

738.

```
< X509_Cert_Type::get_from_database definition 735 > +≡
if (row_ctr == 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'X509_Cert_Type::get_from_database':" <<
        endl << "'row_ctr'==0." << endl << "Failed to retrieve X.509 certificate da\
ta from database table" << "'gwiridsif.Certificates' for user" << user_id << "." <<
        endl << "Exiting function unsuccessfully with return value 3." << endl;
unlock_cerr_mutex();
if (result) mysql_free_result(result);
return 3;
} /* if (row_ctr == 0) */
```

739.

```
< X509_Cert_Type::get_from_database definition 735 > +≡
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "In 'X509_Cert_Type::get_from_database':" << endl <<
            "'row_ctr'==0" << row_ctr << endl << "'field_ctr'==0" << field_ctr << endl;
    if (row_ctr > 1)
        cerr << "'row_ctr'==0" << row_ctr << " Will retrieve the certificate data" <<
            "with the highest value for 'gwiridsif.Certificates.certificate_id'." << endl;
    unlock_cerr_mutex();
}
/* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

740.

```
< X509_Cert_Type::get_from_database definition 735 > +≡
if ((curr_row = mysql_fetch_row(result)) == 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'X509_Cert_Type::get_from_database':" <<
        endl << "'mysql_fetch_row' failed:" << endl << mysql_error(mysql_ptr) << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    return 1;
} /* if (curr_row = mysql_fetch_row(result) == 0) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "In 'X509_Cert_Type::get_from_database':" << endl <<
            "'mysql_fetch_row' succeeded." << endl;
        unlock_cerr_mutex();
}
/* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

741.

```

⟨ X509_Cert_Type::get_from_database definition 735 ⟩ +≡
status = set(curr_row, thread_ctr_str);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'X509_Cert_Type::get_from_database':"
    endl << "'X509_Cert_Type::set' failed, returning" << status << "."
    endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    return 1;
} /* if (status ≠ 0) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "In 'X509_Cert_Type::get_from_database':"
        endl << "'X509_Cert_Type::set' succeeded, returning 0."
        endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

742.

```

⟨ X509_Cert_Type::get_from_database definition 735 ⟩ +≡
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "Exiting 'X509_Cert_Type::get_from_database' successfully wi\
        th return value 0." << endl;
    show(*this);
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
if (result) mysql_free_result(result);
return 0; /* End of X509_Cert_Type::get_from_database definition */

```

743. Show. [LDF 2009.12.22.]

Log

[LDF 2009.12.22.] Added this function.

[LDF 2009.12.29.] Added optional argument **bool show_issuer**.

⟨ X509_Cert_Type function declarations 705 ⟩ +≡

void show(string s = "", stringstream *strm = 0, bool show_issuer = false) const;

744.

```

⟨ X509_Cert_Type::show definition 744 ⟩ ≡
void X509_Cert_Type::show(string s, stringstream *strm, bool show_issuer) const
{
    if (s == "") s = "X509_Cert_Type:";

    stringstream temp_strm;
    temp_strm << setfill(' ') << s << endl << setw(34) << std::left << "serialNumber:" <<
        hex << serialNumber << " (hexadecimal)" << dec << endl << setw(34) <<
        std::left << "organization:" << organization << endl << setw(34) <<
        std::left << "organizationalUnitName:" << organizationalUnitName <<
        endl << setw(34) << std::left << "commonName:" << commonName << endl <<
        setw(34) << std::left << "countryName:" << countryName << endl << setw(34) <<
        std::left << "localityName:" << localityName << endl << setw(34) << std::left <<
        "stateOrProvinceName:" << stateOrProvinceName << endl << setw(34) << std::left <<
        "user_id:" << user_id << endl << setw(34) << std::left << "certificate_id:" <<
        certificate_id << endl << setw(34) << std::left << "issuer_cert_id:" << issuer_cert_id <<
        endl << setw(34) << std::left << "user_name:" << user_name << endl << setw(34) <<
        std::left << "is_ca:" << is_ca << endl << setw(34) << std::left << "is_proxy:" <<
        is_proxy << endl;

    char outstr[200];
    struct tm tmp;
    struct tm *tmp_ptr = &tmp;

    temp_strm << setw(34) << std::left << "Activation_time:";
    if (Validity_notBefore > 0) gmtime_r(&Validity_notBefore, tmp_ptr);
    else tmp_ptr = 0;
    if (!tmp_ptr) {
        temp_strm << outstr << endl;
        else temp_strm << "Unknown" << endl;
    }
    temp_strm << setw(34) << std::left << "Expiration_time:";
    if (Validity_notAfter > 0) gmtime_r(&Validity_notAfter, tmp_ptr);
    else tmp_ptr = 0;
    if (!tmp_ptr) {
        temp_strm << outstr << endl;
        else temp_strm << "Unknown" << endl;
    }
    if (show_issuer) {
        if (issuer_cert) issuer_cert->show("issuer:", strm, false);
        else temp_strm << "'issuer' is null." << endl;
    }
    /* if (show_issuer) */
    temp_strm << setfill(' ');
    if (strm) *strm << temp_strm.str();
    else cerr << temp_strm.str();
    return;
} /* End of X509_Cert_Type::show definition */

```

This code is used in section 748.

745. Output. [LDF 2013.05.15.]

This function outputs information about the **X509_Cert_Type** object in the format needed for sending it to the client program **gwirdcli**. [LDF 2013.05.15.]

Log

[LDF 2013.05.15.] Added this function.
 [LDF 2013.05.19.] Made this function **const**.

< X509_Cert_Type function declarations 705 > +≡

string output(void) const;

746.

< X509_Cert_Type :: output definition 746 > ≡

```
string X509_Cert_Type :: output(void) const
{
    stringstream temp_strm;
    temp_strm << "CERTIFICATE_ID" << certificate_id << "USER_ID" << user_id << "USER_NAME"
    << user_name << "\u" << "ISSUER_CERT_ID" << issuer_cert_id << "IS_CA"
    << is_ca << "\u" << "IS_PROXY" << is_proxy << "\u" << "ORGANIZATION"
    << organization << "\u" << "ORGANIZATIONAL_UNIT_NAME" << organizationalUnitName <<
    "COMMON_NAME" << commonName << "\u" << "COUNTRY_NAME" << countryName << "\u" <<
    "LOCALITY_NAME" << localityName << "\u" << "STATE_OR_PROVINCE_NAME" << stateOrProvinceName << "\u" <<
    "SERIAL_NUMBER" << serialNumber << "UL" << "VALIDITY_NOT_BEFORE" << Validity_notBefore << "UL"
    << "VALIDITY_NOT_AFTER" << Validity_notAfter << "UL";
    return temp_strm.str();
}
```

/* End of X509_Cert_Type :: output definition */

This code is used in section 748.

747.

< Garbage 303 > +≡

748. Putting Types for X.509 Certificates together.

```
using namespace std;
< Include files 3 >

using namespace gwrdifpk;
typedef void *yyscan_t;

< class X509_Cert_Type declaration 702 >
< X509_Cert_Type constructor definitions 706 >
< X509_Cert_Type :: set definitions 712 >
< X509_Cert_Type :: operator= definition 731 >
< X509_Cert_Type :: clear definition 733 >
< X509_Cert_Type :: get_from_database definition 735 >
< X509_Cert_Type :: output definition 746 >
< X509_Cert_Type :: show definition 744 >
#endif /* 1 */
< Garbage 303 >
#endif
```

749.

```
<x509cert.h 749>≡  
#ifndef X509CERT_H  
#define X509CERT_H 1  
    typedef void *yytoken_t;  
    class Scan_Parse_Parameter_Type;  
    < class X509_Cert_Type declaration 702 >  
#endif
```

750. Handle_Value_Type (hdlvltp.web).

Log

[LDF 2013.02.28.] Added this file. It contains code for **class Handle_Value_Type** removed from hndltype.web. █

751. Include files.

```
<Include files 3> +≡
#ifndef _XOPEN_SOURCE
#define _XOPEN_SOURCE
#endif
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <unistd.h>
#include <limits.h>
#if 0
#include <sys/stat.h>
#include <sys/time.h>
#endif
#include <time.h>
#include <sys/types.h>
#if 0
#include <dirent.h>
#endif
#include <string.h>
#if 0
#include <pwd.h>
#endif
#include <errno.h>
#include <pthread.h>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <string>
#include <sstream>
#include <deque>
#include <map>
#include <stack>
#include <vector>
#include <gcrypt.h> /* for gcry-control */
#include <gnutls/gnutls.h>
#include <gnutls/x509.h>
#include <expat.h>
#include <mysql.h>
#if HAVE_CONFIG_H
#include <config.h>
#endif
#undef NAME_LEN
#undef LOCAL_HOST
#include "glblcnst.h++"
#include "glblvrb1.h++"
#include "rspercds.h++"
#include "excptntp.h++"
#include "utilfncs.h++"
#include "hndlvltp.h++"
#include "rspnstp.h++"
```

```
#include "parser.h++"
#include "scanner.h++"
#include "grouptp.h++"
#include "irdsavtp.h++"
#include "irdsobtp.h++"
#include "dcmtddtp.h++"
#include "dstngnmt.h++"
#include "x509cert.h++"
#include "usrinftp.h++"
#include "scprpmtp.h++"
```

752. class Handle_Type.

Log

[LDF 2012.10.12.] Added this **class** declaration.

[LDF 2012.11.22.] Added **string** *filename*.

[LDF 2013.01.11.] Added **bool** *deleted*. It's set to *false* by default.

[LDF 2013.01.14.] Added **friend** declaration for *generate_pids*.

[LDF 2013.01.31.] Added **static const unsigned int** data members for use in the *idx* field of handles (i.e., PIDs).

[LDF 2013.02.05.] Added **static const unsigned int** OTHER_HANDLE_VALUE_TYPE_INDEX. Changed the values of the other constants so that ranges can be used. A handle can have multiple values of the same type, so that it must be possible to use different indices for a given type.

[LDF 2013.02.07.] Added **string** *created_str* and **string** *last_modified_str*.

[LDF 2013.02.08.] Add **int** *created_by_user_id*.

[LDF 2013.02.22.] Added **static map<string, unsigned int>** *type_idx_map*.

[LDF 2013.02.22.] Added the **static const unsigned int** data members RESERVED_0_INDEX, RESERVED_1_INDEX and RESERVED_2_INDEX.

[LDF 2013.02.22.] Added **friend** declaration for *main*. This is needed for the *main* function of the program genpids.

[LDF 2013.02.27.] Renamed this class. Old name: **class Handle_Type**. New name: **class Handle_Value_Type**.
Added a new **class Handle_Type** which contains a **map < Handle_Value_Type**. See the declaration of the latter class below.

Added **friend** declaration for **class Handle_Type**.

[LDF 2013.02.28.] Moved this class declaration and the member function definitions from **hdltype.web** to this file (**hndlvtlp.web**).

[LDF 2013.02.28.] Added **string** *created_by_user_name*. See comment below.

[LDF 2013.03.07.] Added the **static const unsigned int** data members DC_METADATA_IRODS_OBJECT_INDEX, DC_METADATA_IRODS_OBJECT_PID_INDEX and DC_METADATA_IRODS_OBJECT_REF_INDEX.

[LDF 2013.07.03.] Added the **static const unsigned int** data members CREATOR_INDEX and OWNER_INDEX.

[LDF 2013.07.04.] Added **static const unsigned int** IRODS_OBJECT_DELETED_INDEX.

[LDF 2013.07.25.] Added **static const unsigned int** HANDLE_MARKED_FOR_DELETION_INDEX.

[LDF 2013.08.09.] Added **static const unsigned int** IRODS_OBJECT_DELETED_FROM_GWIRDSIF_DB_INDEX, static const unsigned int DC_METADATA_IRODS_OBJECT_DELETED_FROM_ARCHIVE_INDEX and static const unsigned int DC_METADATA_IRODS_OBJECT_DELETED_FROM_GWIRDSIF_DB_INDEX.

[LDF 2013.08.12.] Added **static const unsigned int** IRODS_OBJECT_MARKED_FOR_DELETION_FROM_ARCHIVE_INDEX, static const unsigned int IRODS_OBJECT_MARKED_FOR_DELETION_FROM_GWIRDSIF_DB_INDEX, static const unsigned int DC_METADATA_IRODS_OBJECT_MARKED_FOR_DELETION_FROM_ARCHIVE_INDEX and static const unsigned int DC_METADATA_IRODS_OBJECT_MARKED_FOR_DELETION_FROM_GWIRDSIF_DB_INDEX.

[LDF 2013.08.15.] Added the **static unsigned int** constants IRODS_OBJECT_REF_DELETED_FROM_ARCHIVE_INDEX, IRODS_OBJECT_REF_DELETED_FROM_GWIRDSIF_DB_INDEX, DC_METADATA_IRODS_OBJECT_REF_DELETED_FROM_ARCHIVE_INDEX and DC_METADATA_IRODS_OBJECT_REF_DELETED_FROM_GWIRDSIF_DB_INDEX.

[LDF 2013.08.26.] Added **time_t** *delete_from_database_timestamp*, **string** *delete_from_database_timestamp_str*. and **unsigned long** *irods_object_id*.

```
< Declare class Handle_Value_Type 752 > ≡
class Handle_Value_Type {
    friend class Scan_Parse_Parameter_Type;
    friend class Irods_Object_Type;
    friend class Handle_Type;
    friend int exchange_data_with_client(Scan_Parse_Parameter_Type &param);
    friend int zzparse(yyscan_t);
    friend int main(int argc, char *argv[]);
```

```

friend int generate_pids(MYSQL *mysql_ptr, string prefix_str, string &pid_str, vector<string>
    *pid_vector_ptr, unsigned int number_of_pids, vector<unsigned long int> *handle_id_vector_ptr,
    vector<unsigned long int> *handle_value_id_vector_ptr, bool standalone_hs, string
    institute_str, string suffix_str, vector<Handle_Type> *handle_vector, string fifo_pathname, long
    int user_id, string username);

string filename;
string handle;
int idx;
string type;
char *data;
unsigned int data_length;
int ttl_type;
int ttl;
time_t timestamp;
char *refs;
unsigned int refs_length;
bool admin_read;
bool admin_write;
bool pub_read;
bool pub_write;
unsigned long int handle_id;
unsigned long int handle_value_id;
int created_by_user_id;
unsigned long irods_object_id;
string created_by_user_name; /* As of this date (2013.02.28.), created_by_user_name is only used
    in Scan_Parse_Parameter_Type::get_handle and zzparse. Otherwise it's not needed, since the
    username is available on the server-side. There is no corresponding field in the gwirdsif.Users
    database table. */
bool marked_for_deletion;
time_t created;
time_t last_modified;
time_t delete_from_database_timestamp;
string created_str;
string last_modified_str;
string delete_from_database_timestamp_str;

public: <Handle_Value_Type function declarations 754>

static const unsigned int NULL_HANDLE_VALUE_TYPE_INDEX;
static const unsigned int IRODS_OBJECT_INDEX;
static const unsigned int IRODS_OBJECT_PID_INDEX;
static const unsigned int IRODS_OBJECT_REF_INDEX;
static const unsigned int IRODS_OBJECT_DELETED_FROM_ARCHIVE_INDEX;
static const unsigned int IRODS_OBJECT_DELETED_FROM_GWIRDSIF_DB_INDEX;
static const unsigned int IRODS_OBJECT_REF_DELETED_FROM_ARCHIVE_INDEX;
static const unsigned int IRODS_OBJECT_REF_DELETED_FROM_GWIRDSIF_DB_INDEX;
static const unsigned int IRODS_OBJECT_MARKED_FOR_DELETION_FROM_ARCHIVE_INDEX;
static const unsigned int IRODS_OBJECT_MARKED_FOR_DELETION_FROM_GWIRDSIF_DB_INDEX;
static const unsigned int DC_METADATA_INDEX;
static const unsigned int DC_METADATA_PID_INDEX;
static const unsigned int DC_METADATA_REF_INDEX;
static const unsigned int DC_METADATA_IRODS_OBJECT_INDEX;
static const unsigned int DC_METADATA_IRODS_OBJECT_PID_INDEX;
static const unsigned int DC_METADATA_IRODS_OBJECT_REF_INDEX;

```

```
static const unsigned int DC_METADATA_IRODS_OBJECT_DELETED_FROM_ARCHIVE_INDEX;
static const unsigned int DC_METADATA_IRODS_OBJECT_DELETED_FROM_GWIRDSIF_DB_INDEX;
static const unsigned int DC_METADATA_IRODS_OBJECT_REF_DELETED_FROM_ARCHIVE_INDEX;
static const unsigned int DC_METADATA_IRODS_OBJECT_REF_DELETED_FROM_GWIRDSIF_DB_INDEX;
static const unsigned int DC_METADATA_IRODS_OBJECT_MARKED_FOR_DELETION_FROM_ARCHIVE_INDEX;
static const unsigned int
    DC_METADATA_IRODS_OBJECT_MARKED_FOR_DELETION_FROM_GWIRDSIF_DB_INDEX;
static const unsigned int CREATOR_INDEX;
static const unsigned int OWNER_INDEX;
static const unsigned int HANDLE_MARKED_FOR_DELETION_INDEX;
static const unsigned int RESERVED_0_INDEX;
static const unsigned int RESERVED_1_INDEX;
static const unsigned int RESERVED_2_INDEX;
static const unsigned int OTHER_HANDLE_VALUE_TYPE_INDEX;
static map<int, string> idx_type_map;
static map<string, unsigned int> type_idx_map;
};
```

This code is used in section 884.

753.

Log

[LDF 2013.01.31.] Added this section.

```

⟨ Initialize static Handle_Value_Type data members 753 ⟩ ≡
const unsigned int Handle_Value_Type::NULL_HANDLE_VALUE_TYPE_INDEX = 0;
const unsigned int Handle_Value_Type::IRODS_OBJECT_INDEX = 1;
const unsigned int Handle_Value_Type::IRODS_OBJECT_PID_INDEX = 11;
const unsigned int Handle_Value_Type::IRODS_OBJECT_REF_INDEX = 21;
const unsigned int Handle_Value_Type::IRODS_OBJECT_DELETED_FROM_ARCHIVE_INDEX = 31;
const unsigned int Handle_Value_Type::IRODS_OBJECT_DELETED_FROM_GWIRDSIF_DB_INDEX = 41;
const unsigned int Handle_Value_Type::IRODS_OBJECT_REF_DELETED_FROM_ARCHIVE_INDEX = 51;
const unsigned int Handle_Value_Type::IRODS_OBJECT_REF_DELETED_FROM_GWIRDSIF_DB_INDEX =
    61;
const unsigned int
    Handle_Value_Type::IRODS_OBJECT_MARKED_FOR_DELETION_FROM_ARCHIVE_INDEX = 71;
const unsigned int
    Handle_Value_Type::IRODS_OBJECT_MARKED_FOR_DELETION_FROM_GWIRDSIF_DB_INDEX = 81;
const unsigned int Handle_Value_Type::DC_METADATA_INDEX = 91;
const unsigned int Handle_Value_Type::DC_METADATA_PID_INDEX = 101;
const unsigned int Handle_Value_Type::DC_METADATA_REF_INDEX = 111;
const unsigned int Handle_Value_Type::DC_METADATA_IRODS_OBJECT_INDEX = 121;
const unsigned int Handle_Value_Type::DC_METADATA_IRODS_OBJECT_PID_INDEX = 131;
const unsigned int Handle_Value_Type::DC_METADATA_IRODS_OBJECT_REF_INDEX = 141;
const unsigned int
    Handle_Value_Type::DC_METADATA_IRODS_OBJECT_DELETED_FROM_ARCHIVE_INDEX = 151;
const unsigned int
    Handle_Value_Type::DC_METADATA_IRODS_OBJECT_DELETED_FROM_GWIRDSIF_DB_INDEX = 161;
const unsigned int
    Handle_Value_Type::DC_METADATA_IRODS_OBJECT_REF_DELETED_FROM_ARCHIVE_INDEX = 171;
const unsigned int
    Handle_Value_Type::DC_METADATA_IRODS_OBJECT_REF_DELETED_FROM_GWIRDSIF_DB_INDEX =
        181;
const unsigned int
    Handle_Value_Type::DC_METADATA_IRODS_OBJECT_MARKED_FOR_DELETION_FROM_ARCHIVE_INDEX =
        191;
const unsigned int Handle_Value_Type::DC_METADATA_IRODS_OBJECT_MARKED_FOR_DELETION_FROM_GWIRDSIF_D
    201;
const unsigned int Handle_Value_Type::CREATOR_INDEX = 211;
const unsigned int Handle_Value_Type::OWNER_INDEX = 221;
const unsigned int Handle_Value_Type::HANDLE_MARKED_FOR_DELETION_INDEX = 231;
const unsigned int Handle_Value_Type::RESERVED_0_INDEX = 301;
const unsigned int Handle_Value_Type::RESERVED_1_INDEX = 401;
const unsigned int Handle_Value_Type::RESERVED_2_INDEX = 501;
const unsigned int Handle_Value_Type::OTHER_HANDLE_VALUE_TYPE_INDEX = 601;
map<int, string> Handle_Value_Type::idx_type_map;
map<string, unsigned int> Handle_Value_Type::type_idx_map;

```

This code is used in section 883.

754. Default constructor. [LDF 2012.10.12.]

Log

[LDF 2012.10.12.] Added this function.

⟨ **Handle_Value_Type** function declarations 754 ⟩ ≡
Handle_Value_Type(void);

See also sections 756, 758, 760, 762, 764, 789, 796, 830, 855, 869, 871, and 873.

This code is used in section 752.

755.

⟨ **Handle_Value_Type** constructor definitions 755 ⟩ ≡

```
Handle_Value_Type::Handle_Value_Type(void)
{
    idx = 0;
    data = 0;
    data_length = 0;
    ttl_type = 0;
    ttl = 0;
    timestamp = 0;
    refs = 0;
    refs_length = 0;
    handle_id = 0UL;
    handle_value_id = 0UL;
    created_by_user_id = 0;
    irods_object_id = 0UL;
    marked_for_deletion = false;
    created = 0;
    last_modified = 0;
    delete_from_database_timestamp = 0;
    return;
} /* End of Handle_Value_Type default constructor definition */
```

See also section 757.

This code is used in section 883.

756. Copy constructor. [LDF 2013.01.15.]

Log

[LDF 2013.01.15.] Added this function.

⟨ **Handle_Value_Type** function declarations 754 ⟩ +≡
Handle_Value_Type(const **Handle_Value_Type** &h);

757.

```

⟨Handle_Value_Type constructor definitions 755⟩ +≡
Handle_Value_Type::Handle_Value_Type(const Handle_Value_Type &h)
{
    bool DEBUG = false;      /* true */
    set_debug_level(DEBUG, 0, 0);
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Entering 'Handle_Value_Type' copy constructor." << endl;
        unlock_cerr_mutex();
    }      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */
    data = 0;
    refs = 0;
    operator=(h);
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Exiting 'Handle_Value_Type' copy constructor." << endl;
        unlock_cerr_mutex();
    }      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */
    return;
}      /* End of Handle_Value_Type copy constructor */

```

758. Assignment operator. [LDF 2013.02.27.]

Log

[LDF 2013.02.27.] Added this function.

```

⟨Handle_Value_Type function declarations 754⟩ +≡
void operator=(const Handle_Value_Type &h);

```

759.

```
<Handle_Value_Type::operator= definition 759> ≡
void Handle_Value_Type::operator=(const Handle_Value_Type &h)
{
    bool DEBUG = false;      /* true */
    set_debug_level(DEBUG, 0, 0);
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Entering 'Handle_Value_Type' assignment operator." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    filename = h.filename;
    handle = h.handle;
    idx = h.idx;
    type = h.type;
    if (data) {
        delete[] data;
        data = 0;
    }
    if (refs) {
        delete[] refs;
        refs = 0;
    }
    data_length = h.data_length;
    ttl_type = h.ttl_type;
    ttl = h.ttl;
    timestamp = h.timestamp;
    refs_length = h.refs_length;
    admin_read = h.admin_read;
    admin_write = h.admin_write;
    pub_read = h.pub_read;
    pub_write = h.pub_write;
    handle_id = h.handle_id;
    handle_value_id = h.handle_value_id;
    marked_for_deletion = h.marked_for_deletion;
    created = h.created;
    last_modified = h.last_modified;
    delete_from_database_timestamp = h.delete_from_database_timestamp;
    created_str = h.created_str;
    last_modified_str = h.last_modified_str;
    delete_from_database_timestamp_str = h.delete_from_database_timestamp_str;
    created_by_user_id = h.created_by_user_id;
    irods_object_id = h.irods_object_id;
    created_by_user_name = h.created_by_user_name;
    if (h.data) {
        data = new char[data_length];
        memcpy(data, h.data, data_length);
    }
    else data = 0;
    if (h.refs) {
```

```
    refs = new char[refs_length];
    memcpy(refs, h.refs, refs_length);
}
else refs = 0;
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "Exiting 'Handle_Value_Type' assignment operator." << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
return;
} /* End of Handle_Value_Type assignment operator definition */
```

This code is used in section 883.

760. Destructor. [LDF 2012.10.12.]

Log

[LDF 2012.10.12.] Added this function.

⟨Handle_Value_Type function declarations 754⟩ +≡
~Handle_Value_Type(void);

761.

```

⟨ Handle_Value_Type destructor definition 761 ⟩ ≡
Handle_Value_Type::~Handle_Value_Type(void)
{
    bool DEBUG = false;      /* true */
    set_debug_level(DEBUG, 0, 0);
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Entering 'Handle_Value_Type' destructor." << endl;
        unlock_cerr_mutex();
    }      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */
    if (data) {
        delete[] data;
        data = 0;
    }
    if (refs) {
        delete[] refs;
        refs = 0;
    }
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Exiting 'Handle_Value_Type' destructor successfully with no return value." <<
            endl;
        unlock_cerr_mutex();
    }      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */
    return;
}      /* End of Handle_Value_Type destructor definition */

```

This code is used in section 883.

762. Initialize *idx_type_map* and *type_idx_map*. [LDF 2013.02.05.]**Log**

[LDF 2013.02.05.] Added this function.
 [LDF 2013.02.22.] Changed the name of this function from *initialize_idx_type_map* to *initialize_maps*.
 [LDF 2013.02.22.] Added code for initializing *type_idx_map*.
 [LDF 2013.03.07.] Added code to account for the new **static const unsigned int** data members DC_METADATA_IRODS_O
 DC_METADATA_IRODS_OBJECT_PID_INDEX and DC_METADATA_IRODS_OBJECT_REF_INDEX.
 No significant changes. Added code to account for the new **static const unsigned int** data members
 CREATOR_INDEX and OWNER_INDEX.

```

⟨ Handle_Value_Type function declarations 754 ⟩ +≡
static int initialize_maps(void);

```

763.

```

⟨ Handle_Value_Type :: initialize_maps definition 763 ⟩ ≡
int Handle_Value_Type :: initialize_maps(void)
{
    bool DEBUG = false;      /* true */
    set_debug_level(DEBUG, 0, 0);
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Entering Handle_Value_Type::initialize_maps." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    idx_type_map[NULL_HANDLE_VALUE_TYPE_INDEX] = "NULL_HANDLE_VALUE_TYPE";
    idx_type_map[IRODS_OBJECT_INDEX] = "IRODS_OBJECT";
    idx_type_map[IRODS_OBJECT_PID_INDEX] = "IRODS_OBJECT_PID";
    idx_type_map[IRODS_OBJECT_REF_INDEX] = "IRODS_OBJECT_REF";
    idx_type_map[IRODS_OBJECT_DELETED_FROM_ARCHIVE_INDEX] =
        "IRODS_OBJECT_DELETED_FROM_ARCHIVE";
    idx_type_map[IRODS_OBJECT_DELETED_FROM_GWIRDSIF_DB_INDEX] =
        "IRODS_OBJECT_DELETED_FROM_GWIRDSIF_DB";
    idx_type_map[IRODS_OBJECT_REF_DELETED_FROM_ARCHIVE_INDEX] =
        "IRODS_OBJECT_REF_DELETED_FROM_ARCHIVE";
    idx_type_map[IRODS_OBJECT_REF_DELETED_FROM_GWIRDSIF_DB_INDEX] =
        "IRODS_OBJECT_REF_DELETED_FROM_GWIRDSIF_DB";
    idx_type_map[IRODS_OBJECT_MARKED_FOR_DELETION_FROM_ARCHIVE_INDEX] =
        "IRODS_OBJECT_MARKED_FOR_DELETION_FROM_ARCHIVE";
    idx_type_map[IRODS_OBJECT_MARKED_FOR_DELETION_FROM_GWIRDSIF_DB_INDEX] =
        "IRODS_OBJECT_MARKED_FOR_DELETION_FROM_GWIRDSIF_DB";
    idx_type_map[DC_METADATA_INDEX] = "DC_METADATA";
    idx_type_map[DC_METADATA_PID_INDEX] = "DC_METADATA_PID";
    idx_type_map[DC_METADATA_REF_INDEX] = "DC_METADATA_REF";
    idx_type_map[DC_METADATA_IRODS_OBJECT_INDEX] = "DC_METADATA_IRODS_OBJECT";
    idx_type_map[DC_METADATA_IRODS_OBJECT_PID_INDEX] = "DC_METADATA_IRODS_OBJECT_PID";
    idx_type_map[DC_METADATA_IRODS_OBJECT_REF_INDEX] = "DC_METADATA_IRODS_OBJECT_REF";
    idx_type_map[DC_METADATA_IRODS_OBJECT_DELETED_FROM_ARCHIVE_INDEX] =
        "DC_METADATA_IRODS_OBJECT_DELETED_FROM_ARCHIVE";
    idx_type_map[DC_METADATA_IRODS_OBJECT_DELETED_FROM_GWIRDSIF_DB_INDEX] =
        "DC_METADATA_IRODS_OBJECT_DELETED_FROM_GWIRDSIF_DB";
    idx_type_map[DC_METADATA_IRODS_OBJECT_REF_DELETED_FROM_ARCHIVE_INDEX] =
        "DC_METADATA_IRODS_OBJECT_REF_DELETED_FROM_ARCHIVE";
    idx_type_map[DC_METADATA_IRODS_OBJECT_REF_DELETED_FROM_GWIRDSIF_DB_INDEX] =
        "DC_METADATA_IRODS_OBJECT_REF_DELETED_FROM_GWIRDSIF_DB";
    idx_type_map[DC_METADATA_IRODS_OBJECT_MARKED_FOR_DELETION_FROM_ARCHIVE_INDEX] =
        "DC_METADATA_IRODS_OBJECT_MARKED_FOR_DELETION_FROM_ARCHIVE";
    idx_type_map[DC_METADATA_IRODS_OBJECT_MARKED_FOR_DELETION_FROM_GWIRDSIF_DB_INDEX] =
        "DC_METADATA_IRODS_OBJECT_MARKED_FOR_DELETION_FROM_GWIRDSIF_DB";
    idx_type_map[CREATOR_INDEX] = "CREATOR";
    idx_type_map[OWNER_INDEX] = "OWNER";
    idx_type_map[HANDLE_MARKED_FOR_DELETION_INDEX] = "HANDLE_MARKED_FOR_DELETION";
    idx_type_map[RESERVED_0_INDEX] = "RESERVED_0";
    idx_type_map[RESERVED_1_INDEX] = "RESERVED_1";

```

```

idx_type_map[RESERVED_2_INDEX] = "RESERVED_2";
idx_type_map[OTHER_HANDLE_VALUE_TYPE_INDEX] = "OTHER_HANDLE_VALUE_TYPE";
type_idx_map["NULL_HANDLE_VALUE_TYPE"] = NULL_HANDLE_VALUE_TYPE_INDEX;
type_idx_map["IRODS_OBJECT"] = IRODS_OBJECT_INDEX;
type_idx_map["IRODS_OBJECT_PID"] = IRODS_OBJECT_PID_INDEX;
type_idx_map["IRODS_OBJECT_REF"] = IRODS_OBJECT_REF_INDEX;
type_idx_map["IRODS_OBJECT_DELETED_FROM_ARCHIVE"] =
    IRODS_OBJECT_DELETED_FROM_ARCHIVE_INDEX;
type_idx_map["IRODS_OBJECT_DELETED_FROM_GWIRDSIF_DB"] =
    IRODS_OBJECT_DELETED_FROM_GWIRDSIF_DB_INDEX;
type_idx_map["IRODS_OBJECT_REF_DELETED_FROM_ARCHIVE"] =
    IRODS_OBJECT_REF_DELETED_FROM_ARCHIVE_INDEX;
type_idx_map["IRODS_OBJECT_REF_DELETED_FROM_GWIRDSIF_DB"] =
    IRODS_OBJECT_REF_DELETED_FROM_GWIRDSIF_DB_INDEX;
type_idx_map["IRODS_OBJECT_MARKED_FOR_DELETION_FROM_ARCHIVE"] =
    IRODS_OBJECT_MARKED_FOR_DELETION_FROM_ARCHIVE_INDEX;
type_idx_map["IRODS_OBJECT_MARKED_FOR_DELETION_FROM_GWIRDSIF_DB"] =
    IRODS_OBJECT_MARKED_FOR_DELETION_FROM_GWIRDSIF_DB_INDEX;
type_idx_map["DC_METADATA"] = DC_METADATA_INDEX;
type_idx_map["DC_METADATA_PID"] = DC_METADATA_PID_INDEX;
type_idx_map["DC_METADATA_REF"] = DC_METADATA_REF_INDEX;
type_idx_map["DC_METADATA_IRODS_OBJECT"] = DC_METADATA_IRODS_OBJECT_INDEX;
type_idx_map["DC_METADATA_IRODS_OBJECT_PID"] = DC_METADATA_IRODS_OBJECT_PID_INDEX;
type_idx_map["DC_METADATA_IRODS_OBJECT_REF"] = DC_METADATA_IRODS_OBJECT_REF_INDEX;
type_idx_map["DC_METADATA_IRODS_OBJECT_DELETED_FROM_ARCHIVE"] =
    DC_METADATA_IRODS_OBJECT_DELETED_FROM_ARCHIVE_INDEX;
type_idx_map["DC_METADATA_IRODS_OBJECT_DELETED_FROM_GWIRDSIF_DB"] =
    DC_METADATA_IRODS_OBJECT_DELETED_FROM_GWIRDSIF_DB_INDEX;
type_idx_map["DC_METADATA_IRODS_OBJECT_REF_DELETED_FROM_ARCHIVE"] =
    DC_METADATA_IRODS_OBJECT_REF_DELETED_FROM_ARCHIVE_INDEX;
type_idx_map["DC_METADATA_IRODS_OBJECT_REF_DELETED_FROM_GWIRDSIF_DB"] =
    DC_METADATA_IRODS_OBJECT_REF_DELETED_FROM_GWIRDSIF_DB_INDEX;
type_idx_map["DC_METADATA_IRODS_OBJECT_MARKED_FOR_DELETION_FROM_ARCHIVE"] =
    DC_METADATA_IRODS_OBJECT_MARKED_FOR_DELETION_FROM_ARCHIVE_INDEX;
type_idx_map["DC_METADATA_IRODS_OBJECT_MARKED_FOR_DELETION_FROM_GWIRDSIF_DB"] =
    DC_METADATA_IRODS_OBJECT_MARKED_FOR_DELETION_FROM_GWIRDSIF_DB_INDEX;
type_idx_map["CREATOR"] = CREATOR_INDEX;
type_idx_map["OWNER"] = OWNER_INDEX;
type_idx_map["HANDLE_MARKED_FOR_DELETION"] = HANDLE_MARKED_FOR_DELETION_INDEX;
type_idx_map["RESERVED_0"] = RESERVED_0_INDEX;
type_idx_map["RESERVED_1"] = RESERVED_1_INDEX;
type_idx_map["RESERVED_2"] = RESERVED_2_INDEX;
type_idx_map["OTHER_HANDLE_VALUE_TYPE"] = OTHER_HANDLE_VALUE_TYPE_INDEX;

#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "idx_type_map:" << endl;
    for (map<int, string>::const_iterator iter = idx_type_map.begin(); iter != idx_type_map.end();
         ++iter) {
        cerr << setw(3) << iter->first << ", " << iter->second << endl;
    } /* for */
    unlock_cerr_mutex();
}

```

```

lock_cerr_mutex();
cerr << "Exiting 'Handle_Value_Type::initialize_maps' successfully\n"
     ith_return_value_0." << endl;
unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
return 0;
} /* End of Handle_Value_Type::initialize_maps definition */

```

This code is used in section 883.

764. Set with MYSQL_ROW & argument. [LDF 2012.10.12.]

As of this date (i.e., 2013.02.27.), this function is only called by **Scan_Parse_Parameter_Type::get_handle**, which is defined in **sptfnc1.web**. [LDF 2013.02.27.]

Log

[LDF 2012.10.12.] Added this function.

[LDF 2013.01.14.] BUG FIX: Now converting *created* and *last_modified* to **time_t** values.

[LDF 2013.02.07.] Added argument **unsigned int field_ctr**. Made **string hhandle** optional with the empty string as its default value.

⟨Handle_Value_Type function declarations 754⟩ +≡

```
int set(MYSQL_ROW & curr_row, unsigned int field_ctr, string hhandle = "", int thread_ctr = -1);
```

765.

⟨Handle_Value_Type::set definitions 765⟩ ≡

```

int Handle_Value_Type::set(MYSQL_ROW & curr_row, unsigned int field_ctr, string hhandle, int
                           thread_ctr){ bool DEBUG = false; /* true */
                           set_debug_level(DEBUG, 0, 0);
                           int status;
                           string thread_ctr_str;
                           stringstream temp_strm;
                           struct tm tmp_tm;
                           char *temp_char_ptr = 0;
                           char temp_buffer[32];
                           unsigned long temp_val = 0;
                           memset(temp_buffer, 0, 32);
                           if (thread_ctr > 0) {
                               temp_strm << "[Thread]" << thread_ctr << "]";
                           }
#ifndef DEBUG_COMPILE
                           if (DEBUG) {
                               lock_cerr_mutex();
                               cerr << thread_ctr_str << "Entering 'Handle_Value_Type::set'." << endl <<
                                   "'field_ctr' == " << field_ctr << endl;
                               unlock_cerr_mutex();
                           } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
                           clear();

```

See also sections 766, 767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786, 787, 788, 790, 791, 792, 793, 794, and 795.

This code is used in section 883.

766.

```
⟨Handle_Value_Type::set definitions 765⟩ +≡  
  if (field_ctr ≤ 18) handle = hhandle;  
  /* !! TODO: Add error-handling and debugging output. LDF 2012.10.15. */
```

767. handle_id.

```
⟨Handle_Value_Type::set definitions 765⟩ +≡  
  if (curr_row[0] ∧ strlen(curr_row[0])) {  
    handle_id = atoi(curr_row[0]);  
  }  
  else {}
```

768. handle_value_id.

```
⟨Handle_Value_Type::set definitions 765⟩ +≡  
  if (curr_row[1] ∧ strlen(curr_row[1])) {  
    handle_value_id = atoi(curr_row[1]);  
  }  
  else {}
```

769. idx.

```
⟨Handle_Value_Type::set definitions 765⟩ +≡  
  if (curr_row[2] ∧ strlen(curr_row[2])) {  
    idx = atoi(curr_row[2]);  
  }  
  else {}
```

770. type.

```
⟨Handle_Value_Type::set definitions 765⟩ +≡  
  if (curr_row[3] ∧ strlen(curr_row[3])) {  
    type = curr_row[3];  
  }  
  else {}
```

771. *length*(*data*) and *data*.

```
<Handle_Value_Type::set definitions 765> +≡
  data_length = 0;
  data = 0;
  if (curr_row[4] ∧ strlen(curr_row[4])) {
    data_length = atoi(curr_row[4]);
    if (data_length > 0 ∧ curr_row[5]) {
      data = new char[data_length];
      memset(data, 0, data_length);
      memcpy(data, curr_row[5], data_length);
    }
  #if DEBUG_COMPILE
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "In 'Handle_Value_Type::set': " << "data_length==" << data_length << endl <<
        "data==" << endl;
      fwrite(data, 1, data_length, stderr);
      cerr << endl << "data[data_length-1]==" << data[data_length - 1] <<
        "numerical_value" << static_cast<unsigned int>(data[data_length - 1]) << endl;
      unlock_cerr_mutex();
    } /* if (DEBUG) */
  #endif /* DEBUG_COMPILE */
  } /* if (data_length > 0 ∧ curr_row[5]) */
  else { }
```

772. *ttl_type*.

```
<Handle_Value_Type::set definitions 765> +≡
  if (curr_row[6] ∧ strlen(curr_row[6])) {
    ttl_type = atoi(curr_row[6]);
  }
  else { }
```

773. *ttl*.

```
<Handle_Value_Type::set definitions 765> +≡
  if (curr_row[7] ∧ strlen(curr_row[7])) {
    ttl = atoi(curr_row[7]);
  }
  else { }
```

774. *timestamp*.

```
<Handle_Value_Type::set definitions 765> +≡
  if (curr_row[8] ∧ strlen(curr_row[8])) {
    timestamp = strtoul(curr_row[8], 0, 10);
  }
  else { }
```

775. *length(refs)* and *refs*.

```
<Handle_Value_Type::set definitions 765> +≡
  refs_length = 0;
  refs = 0;
  if (curr_row[9] ∧ strlen(curr_row[9])) {
    refs_length = atoi(curr_row[9]);
    if (refs_length > 0 ∧ curr_row[10]) {
      refs = new char[refs_length];
      memset(refs, 0, refs_length);
      memcpy(refs, curr_row[10], refs_length);
    }
  }
  else { }
```

776. *admin_read*.

```
<Handle_Value_Type::set definitions 765> +≡
  if (curr_row[11] ∧ strlen(curr_row[11])) {
    admin_read = atoi(curr_row[11]);
  }
  else { }
```

777. *admin_write*.

```
<Handle_Value_Type::set definitions 765> +≡
  if (curr_row[12] ∧ strlen(curr_row[12])) {
    admin_write = atoi(curr_row[12]);
  }
  else { }
```

778. *pub_read*.

```
<Handle_Value_Type::set definitions 765> +≡
  if (curr_row[13] ∧ strlen(curr_row[13])) {
    pub_read = atoi(curr_row[13]);
  }
  else { }
```

779. *pub_write*.

```
<Handle_Value_Type::set definitions 765> +≡
  if (curr_row[14] ∧ strlen(curr_row[14])) {
    pub_write = atoi(curr_row[14]);
  }
  else { }
```

780. *marked_for_deletion.*

Log

[LDF 2013.01.11.] Added this section.

```
<Handle_Value_Type::set definitions 765> +≡  
if (curr_row[15] ∧ strlen(curr_row[15])) {  
    marked_for_deletion = strtoul(curr_row[15], 0, 10);  
}  
else {}
```

781. *created.*

Log

[LDF 2013.02.07.] Added code for catching the case that *created* ≡ 0.

```

⟨Handle_Value_Type::set definitions 765⟩ +≡
  if (curr_row[16] ∧ strlen(curr_row[16])) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "curr_row[16] == " << curr_row[16] << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  created_str = curr_row[16];
  if (strcmp(curr_row[16], "0000-00-00 00:00:00") ≡ 0) {
    created = 0;
  }
  else {
    temp_char_ptr = 0;
    temp_char_ptr = strptime(curr_row[16], "%Y-%m-%d %H:%M:%S", &tmp_tm);
    if (temp_char_ptr ≡ 0) {
      lock_cerr_mutex();
      cerr << "WARNING! In 'Handle_Value_Type::set': 'strptime' failed, "
          "returning NULL." << endl << "Setting 'created' to 0. Will try to continue." <<
          endl;
      unlock_cerr_mutex();
      created = 0;
    } /* if (temp_char_ptr ≡ 0) */
    else {
      memset(temp_buffer, 0, 32);
      status = strftime(temp_buffer, 32, "%s", &tmp_tm);
      if (status ≡ 0) {
        lock_cerr_mutex();
        cerr << "WARNING! In 'Handle_Value_Type::set': 'strftime' failed, "
            "returning NULL." << endl << "Setting 'created' to 0. Will try to continue." <<
            endl;
        unlock_cerr_mutex();
        created = 0;
      } /* if (status ≡ 0) */
      else {
        errno = 0;
        temp_val = strtoul(temp_buffer, 0, 10);
        if (temp_val ≡ ULONG_MAX) {
          lock_cerr_mutex();
          cerr << "WARNING! In 'Handle_Value_Type::set': "
              "'strtoul' failed, returning 'ULONG_MAX': " << endl << strerror(errno) << endl <<
              "Setting 'created' to 0. Will try to continue." << endl;
          unlock_cerr_mutex();
          created = 0;
        } /* if (temp_val ≡ ULONG_MAX) */
        else {

```

```
        created = temp_val;
    }
}
/* else */
}
/* else */
}
/* if (curr_row[16] & strlen(curr_row[16])) */
else {
    created = 0;
    created_str = "";
}
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << 'c' << "reated" << endl << 'c' << "reated_str" << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

782. *last-modified*.

Log

[LDF 2013.02.07.] Added code for catching the case that *last-modified* ≡ 0.

```

⟨Handle_Value_Type::set definitions 765⟩ +≡
  if (curr_row[17] ∧ strlen(curr_row[17])) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "curr_row[17] == " << curr_row[17] << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  last_modified_str = curr_row[17];
  if (strcmp(curr_row[17], "0000-00-00 00:00:00") ≡ 0) {
    last_modified = 0;
  }
  else {
    temp_char_ptr = 0;
    temp_char_ptr = strptime(curr_row[17], "%Y-%m-%d %H:%M:%S", &tmp_tm);
    if (temp_char_ptr ≡ 0) {
      lock_cerr_mutex();
      cerr << "WARNING! In 'Handle_Value_Type::set': 'strptime' failed, " <<
        "returning NULL." << endl << "Setting 'last_modified' to 0. Will try \
        to continue." << endl;
      unlock_cerr_mutex();
      last_modified = 0;
    } /* if (temp_char_ptr ≡ 0) */
    else {
      memset(temp_buffer, 0, 32);
      status = strftime(temp_buffer, 32, "%s", &tmp_tm);
      if (status ≡ 0) {
        lock_cerr_mutex();
        cerr << "WARNING! In 'Handle_Value_Type::set': 'strftime' failed, " <<
          "returning NULL." << endl << "Setting 'last_modified' to 0. Will try \
          to continue." << endl;
        unlock_cerr_mutex();
        last_modified = 0;
      } /* if (status ≡ 0) */
      else {
        errno = 0;
        temp_val = strtoul(temp_buffer, 0, 10);
        if (temp_val ≡ ULONG_MAX) {
          lock_cerr_mutex();
          cerr << "WARNING! In 'Handle_Value_Type::set': " <<
            "'strtoul' failed, returning 'ULONG_MAX'" << endl << strerror(errno) << endl <<
            "Setting 'last_modified' to 0. Will try to continue." << endl;
          unlock_cerr_mutex();
          last_modified = 0;
        } /* if (temp_val ≡ ULONG_MAX) */
        else {

```

```

        last_modified = temp_val;
    }
}
/* else */
}
/* else */
}
/* if (curr_row[17] & strlen(curr_row[17])) */
else {
    last_modified = 0;
    last_modified_str = "";
}
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "last_modified" << last_modified << endl << "last_modified_str" <<
        last_modified_str << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

783. handle. [LDF 2013.02.07.]

Log

[LDF 2013.02.07.] Added this section.

```

⟨Handle_Value_Type::set definitions 765⟩ +≡
    if (field_ctr > 18) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "In 'Handle_Value_Type::set': field_ctr" <<
            field_ctr <<
            ">18." << endl << "Setting handle" << curr_row[18] <<
            curr_row[18] <<
            "." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    handle = curr_row[18];
} /* if (field_ctr > 18) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "In 'Handle_Value_Type::set': field_ctr" <<
            field_ctr <<
            "<=18." << endl << "'handle' should have been set to" <<
            "hhandle" <<
            "above." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

784. *created_by_user_id*. [LDF 2013.02.07.]

Log

[LDF 2013.02.07.] Added this section.

```

⟨Handle_Value_Type::set definitions 765⟩ +≡
  if (field_ctr > 19) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'Handle_Value_Type::set': field_ctr' == " << field_ctr <<
      "(>19)." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  if (curr_row[19] & strlen(curr_row[19]) > 0) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'Handle_Value_Type::set': curr_row[19]' == " << curr_row[19] << curr_row[19] << endl << "Setting 'created_by_user_id' to " << curr_row[19] << "."
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  errno = 0;
  created_by_user_id = strtol(curr_row[19], 0, 10);
  if (created_by_user_id == LONG_MAX || created_by_user_id == LONG_MIN) {
    lock_cerr_mutex();
    cerr << "WARNING! In 'Handle_Value_Type::set': 'strtol' failed, returning ";
    if (created_by_user_id == LONG_MAX) cerr << "'LONG_MAX'." << endl;
    else cerr << "'LONG_MIN'." << endl;
    cerr << endl << strerror(errno) << endl << "Setting 'created_by_user_id' to 0. Con\"
      tinuing." << endl;
    unlock_cerr_mutex();
    created_by_user_id = 0;
  } /* if */
#endif DEBUG_COMPILE
  else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'Handle_Value_Type::set': "
      "'strtol' succeeded. Set 'created_by_user_id' to " << created_by_user_id <<
      " ." << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  } /* if (curr_row[19] & strlen(curr_row[19]) > 0) */
  else {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();

```

```
    cerr << thread_ctr_str << "In 'Handle_Value_Type::set': "
    << "'curr_row[19]' is NULL or empty." << endl <<
    "Setting 'created_by_user_id' to 0." << endl;
    unlock_cerr_mutex();
}
/* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    created_by_user_id = 0;
}
/* else */
}
/* if (field_ctr > 19) */
#endif DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'Handle_Value_Type::set': "
    << "'field_ctr' == "
    << field_ctr << endl << "Not setting 'created_by_user_id'." << endl;
    unlock_cerr_mutex();
}
/* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

785. *irods_object_id*. [LDF 2013.08.26.]

Log

[LDF 2013.08.26.] Added this section.

```

⟨Handle_Value_Type::set definitions 765⟩ +≡
  if (field_ctr > 20) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'Handle_Value_Type::set': field_ctr" <=
         << field_ctr <<
         "(>20)." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  if (curr_row[20] & strlen(curr_row[20]) > 0) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'Handle_Value_Type::set': curr_row[20]" <=
         << curr_row[20] << endl << "Setting 'irods_object_id' to" << curr_row[20] << "."
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  errno = 0;
  irods_object_id = strtoul(curr_row[20], 0, 10);
  if (irods_object_id == ULONG_MAX) {
    lock_cerr_mutex();
    cerr << "WARNING! In 'Handle_Value_Type::set': "
         << "'strtoul' failed, returning 'ULONG_MAX'" << endl << strerror(errno) <<
         endl << "Setting 'irods_object_id' to 0UL. Continuing." << endl;
    unlock_cerr_mutex();
    irods_object_id = 0UL;
  } /* if */
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << thread_ctr_str << "In 'Handle_Value_Type::set': "
           << "'strtoul' succeeded. Set 'irods_object_id' to" << irods_object_id << "."
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  } /* if (curr_row[20] & strlen(curr_row[20]) > 0) */
  else {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << thread_ctr_str << "In 'Handle_Value_Type::set': "
           << "'curr_row[20]' is NULL or empty." << endl <<
           "Setting 'irods_object_id' to 0UL."
      unlock_cerr_mutex();
    }

```

```
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    irods_object_id = 0UL;
} /* else */
} /* if (field_ctr > 20) */
#endif DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'Handle_Value_Type::set': field_ctr" <=
        " == " << field_ctr <<
        endl << "Not setting irods_object_id." << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

786. *delete_from_database_timestamp.* [LDF 2013.08.26.]

Log

[LDF 2013.08.26.] Added this section.

```

⟨Handle_Value_Type::set definitions 765⟩ +≡
  if (field_ctr > 21 ∧ curr_row[21] ∧ strlen(curr_row[21])) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "curr_row[21] == " ≪ curr_row[21] ≪ endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  delete_from_database_timestamp_str = curr_row[21];
  if (strcmp(curr_row[21], "0000-00-00 00:00:00") == 0) {
    delete_from_database_timestamp = 0;
  }
  else {
    temp_char_ptr = 0;
    temp_char_ptr = strptime(curr_row[21], "%Y-%m-%d %H:%M:%S", &tmp_tm);
    if (temp_char_ptr == 0) {
      lock_cerr_mutex();
      cerr ≪ "WARNING! In 'Handle_Value_Type::set': 'strptime' failed, "
             ≪ "returning NULL." ≪ endl ≪ "Setting 'delete_from_database_timestamp' "
             ≪ "to 0. Will try to continue." ≪ endl;
      unlock_cerr_mutex();
      delete_from_database_timestamp = 0;
    } /* if (temp_char_ptr == 0) */
    else {
      memset(temp_buffer, 0, 32);
      status = strftime(temp_buffer, 32, "%s", &tmp_tm);
      if (status == 0) {
        lock_cerr_mutex();
        cerr ≪ "WARNING! In 'Handle_Value_Type::set': 'strftime' failed, "
              ≪ "returning NULL." ≪ endl ≪ "Setting 'delete_from_database_timestamp' "
              ≪ "to 0. Will try to continue." ≪ endl;
        unlock_cerr_mutex();
        delete_from_database_timestamp = 0;
      } /* if (status == 0) */
      else {
        errno = 0;
        temp_val = strtoul(temp_buffer, 0, 10);
        if (temp_val == ULONG_MAX) {
          lock_cerr_mutex();
          cerr ≪ "WARNING! In 'Handle_Value_Type::set': "
                ≪ "'strtoul' failed, returning 'ULONG_MAX': " ≪ endl ≪ strerror(errno) ≪
                endl ≪ "Setting 'delete_from_database_timestamp' to 0. "
                ≪ "Will try to continue." ≪ endl;
          unlock_cerr_mutex();
          delete_from_database_timestamp = 0;
        } /* if (temp_val == ULONG_MAX) */
      }
    }
  }
}

```

```

        else {
            delete_from_database_timestamp = temp_val;
        }
    } /* else */
} /* else */
} /* if (field_ctr > 21 & curr_row[21] & strlen(curr_row[21])) */
#endif DEBUG_COMPILE
else
if (field_ctr ≤ 21 & DEBUG) {
lock_cerr_mutex();
cerr << thread_ctr_str << "In 'Handle_Value_Type::set': field_ctr" <=
field_ctr <<
" (≤ 21)." << endl << "Not setting 'delete_from_database_timestamp' and"
" 'delete_from_database_timestamp_str'." << endl;
unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

787.

Log

[LDF 2013.03.01.] Added this section.

```

⟨Handle_Value_Type::set definitions 765⟩ +≡
if (type ≡ "IRODS_OBJECT" & data ≠ 0 & data_length > 0) {
    filename.assign(data, data_length);
}

```

788.

```

⟨Handle_Value_Type::set definitions 765⟩ +≡
#endif DEBUG_COMPILE
if (DEBUG) {
lock_cerr_mutex();
cerr << thread_ctr_str << "Exiting 'Handle_Value_Type::set'" <=
"successfully with return value 0." << endl;
unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
return 0; } /* End of Handle_Value_Type::set */

```

789. Set with arguments for the individual data members. [LDF 2013.01.11.]

Log

[LDF 2013.01.11.] Added this function.

[LDF 2013.02.27.] Added argument **long int ccreated_by_user_id**.

⟨Handle_Value_Type function declarations 754⟩ +≡

```
int set(string hhandle, int iidx, string ttype, char *ddata, unsigned int ddata_length, int ttl_type, int
       ttl, time_t timestamp, char *rrefs, unsigned int rrefs_length, bool aadmin_read, bool
       aadmin_write, bool ppub_read, bool ppub_write, long int ccreated_by_user_id, unsigned long
       int hhandle_id, unsigned long int hhandle_value_id, bool mmarked_for_deletion, time_t
       ccreated, time_t llast_modified, string ccreated_str = "", string llast_modified_str = "", string
       filename = "");
```

790.

⟨Handle_Value_Type::set definitions 765⟩ +≡

```
int Handle_Value_Type::set(string hhandle, int iidx, string ttype, char *ddata, unsigned
                           int ddata_length, int ttl_type, int ttl, time_t timestamp, char *rrefs, unsigned int
                           rrefs_length, bool aadmin_read, bool aadmin_write, bool ppub_read, bool ppub_write, long int
                           ccreated_by_user_id, unsigned long int hhandle_id, unsigned long int hhandle_value_id, bool
                           mmarked_for_deletion, time_t ccreated, time_t llast_modified, string ccreated_str, string
                           llast_modified_str, string filename){ bool DEBUG = false; /* true */
   set_debug_level(DEBUG, 0, 0);
   int status = 0;
#ifndef DEBUG_COMPILE
   if (DEBUG) {
      lock_cerr_mutex();
      cerr << "Entering 'Handle_Value_Type::set'." << endl;
      unlock_cerr_mutex();
   } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

791.

Log

[LDF 2013.01.30.] BUG FIX: Now setting *handle* to *hhandle*.

```

⟨Handle_Value_Type::set definitions 765⟩ +≡
  handle = hhandle;
  idx = iidx;
  type = ttype;
  data_length = ddata_length;
  ttl_type = ttl_type;
  ttl = ttl;
  timestamp = ttimestamp;
  refs_length = rrefs_length;
  admin_read = aadmin_read;
  admin_write = aadmin_write;
  pub_read = ppub_read;
  pub_write = ppub_write;
  handle_id = hhandle_id;
  created_by_user_id = ccreated_by_user_id;
  handle_value_id = hhandle_value_id;
  marked_for_deletion = mmarked_for_deletion;
  created = ccreated;
  last_modified = llast_modified;
  created_str = ccreated_str;
  last_modified_str = llast_modified_str;
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "admin_data_length=" << admin_data_length << endl;
    fwrite(admin_data, 1, admin_data_length, stderr);
    cerr << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  if (ddata ∧ data_length > 0) {
    /* data_length has been set to ddata_length above. [LDF 2013.09.14.] */
    data = new char[data_length];
    memcpy(data, ddata, data_length);
  }
  else data = 0;
  if (rrefs ∧ refs_length > 0) { /* refs_length has been set to rrefs_length above. [LDF 2013.09.14.] */
    refs = new char[refs_length];
    memcpy(refs, rrefs, refs_length);
  }
  else refs = 0;

```

792.

Log

[LDF 2013.02.07.] Added this section.

```

⟨Handle_Value_Type::set definitions 765⟩ +≡
  struct tm tmp;
  char buffer[64];
  memset(buffer, 0, 64);
  if (created > 0 ∧ created_str ≡ "") {
    if (gmtime_r(&created, &tmp) ≡ 0) {
      lock_cerr_mutex();
      cerr << "WARNING! In 'Handle_Value_Type::set':"
          << endl <<
          "'gmtime_r' failed, returning NULL."
          << endl <<
          "Failed to set 'created_str'. Will set it to the empty string."
          << endl << "Continuing."
          << endl;
      unlock_cerr_mutex();
      created_str = "";
    }
  #if DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "In 'Handle_Value_Type::set':"
          << endl << "'gmtime_r' succeeded."
          << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
  #endif /* DEBUG_COMPILE */
  status = strftime(buffer, 64, "%Y-%m-%d %H:%M:%S %Z", &tmp);
  if (status ≡ 0) {
    lock_cerr_mutex();
    cerr << "WARNING! In 'Handle_Value_Type::set':"
        << endl <<
        "'strftime' failed, returning NULL."
        << endl <<
        "Failed to set 'created_str'. Will set it to the empty string."
        << endl << "Continuing."
        << endl;
    unlock_cerr_mutex();
    created_str = "";
  } /* if (status ≡ 0) */
  else {
  #if DEBUG_COMPILE
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "In 'Handle_Value_Type::set':"
          << endl << "'strftime' succeeded, returning"
          << endl <<
          "status << '.' << endl << "'buffer' =="
          << buffer << '.' << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
  #endif /* DEBUG_COMPILE */
    created_str = buffer;
  } /* else */
} /* if (created > 0 ∧ created_str ≡ "") */

```

793.

Log

[LDF 2013.02.07.] Added this section.

```

⟨Handle_Value_Type::set definitions 765⟩ +≡
memset(buffer, 0, 64);
if (last_modified > 0 ∧ last_modified_str ≡ "") {
    if (gmtime_r(&last_modified, &tmp) ≡ 0) {
        lock_cerr_mutex();
        cerr << "WARNING! In 'Handle_Value_Type::set':"
            << endl <<
            "'gmtime_r' failed, returning NULL." << endl <<
            "Failed to set 'last_modified_str'. Will set it to the empty string." << endl <<
            "Continuing." << endl;
        unlock_cerr_mutex();
        last_modified_str = "";
    }
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'Handle_Value_Type::set':"
            << endl << "'gmtime_r' succeeded."
            << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
status = strftime(buffer, 64, "%Y-%m-%d %H:%M:%S %Z", &tmp);
if (status ≡ 0) {
    lock_cerr_mutex();
    cerr << "WARNING! In 'Handle_Value_Type::set':"
        << endl <<
        "'strftime' failed, returning NULL." << endl <<
        "Failed to set 'last_modified_str'. Will set it to the empty string." << endl <<
        "Continuing." << endl;
    unlock_cerr_mutex();
    last_modified_str = "";
} /* if (status ≡ 0) */
else {
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'Handle_Value_Type::set':"
            << endl << "'strftime' succeeded, returning"
            << status << endl << "'buffer' == "
            << buffer << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    last_modified_str = buffer;
} /* else */
} /* if (last_modified > 0 ∧ last_modified_str ≡ "") */

```

794.**Log**

[LDF 2013.03.01.] Added this section.

```
<Handle_Value_Type::set definitions 765> +≡
  if (filename.empty()) filename = filename;
  else if (type ≡ "IRODS_OBJECT" ∧ data ≠ 0 ∧ data_length > 0) {
    filename.assign(data, data_length);
  }
```

795.

```
<Handle_Value_Type::set definitions 765> +≡
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "Exiting 'Handle_Value_Type::set' successfully with return value 0." ≪ endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  return 0; } /* End of Handle_Value_Type::set definition */
```

796. Delete handle value (*delete_handle_value*). [LDF 2013.08.30.]**Log**

[LDF 2013.08.30.] Added this function.

[LDF 2013.09.10.] Added argument **MYSQL** & **mysql_ptr*.

[LDF 2013.09.11.] Added required argument *deque* < **Response_Type** > & *response_deque* and optional arguments **unsigned int** *options* = 0_U and **unsigned long int** *delay_value* = 0.

[LDF 2013.09.12.] Added required arguments **int** *user_id*, **string** *username* and **unsigned int** *privileges*.

[LDF 2013.09.12.] Removed the arguments **char** **buffer_ptr* and **size_t** *buffer_size*.

```
<Handle_Value_Type function declarations 754> +≡
```

```
static int delete_handle_value (MYSQL *&mysql_ptr, string hv_str,
                               deque < Response_Type > &response_deque, int user_id, string
                               username, unsigned int privileges, unsigned int options = 0U,
                               unsigned long int delay_value = 0, string thread_str = "" );
```

797.

```

⟨ Handle_Value_Type :: delete_handle_value definition 797 ⟩ ≡
int Handle_Value_Type :: delete_handle_value (MYSQL * &mysql_ptr, string hv_str, deque <
    Response_Type > &response_deque, int user_id, string username,
    unsigned int privileges, unsigned int options, unsigned long int
    delay_value, string thread_str ) { bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    stringstream temp_strm;
    int status;

#if DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering 'Handle_Value_Type::delete_han\
            dle_value'." << endl << "'options' == " << oct <<
            options << "(octal)" << dec << endl << "'delay_value' == " <<
            delay_value << endl << "'user_id' == " << user_id <<
            endl << "'username' == " << username << endl <<
            "'privileges' == " << oct << privileges << "(octal)" << dec <<
            endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

See also sections 798, 799, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818, 819, 820, 821, 822, 823, 824, 825, 826, 827, 828, and 829.

This code is used in section 883.

798.

```
⟨ Handle_Value_Type :: delete_handle_value definition 797 ⟩ +≡
  stringstream sql_strm;
  MYSQL_RES * result = 0;
  MYSQL_ROW curr_row;
  unsigned int row_ctr = 0;
  unsigned int field_ctr = 0;
  long affected_rows = 0;
  string prefix;
  string sub_handle;
  string handle;
  int index = -1;
  string type;
  Response_Type response;
  response.type = Response_Type::COMMAND_ONLY_TYPE;
  bool immediate = (options & 1U) ? true : false;
  errno = 0;
  time_t curr_time = time(0);
  if (curr_time ≡ static_cast<time_t>(-1)) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Handle_Value_Type::delete_handle_value':"
      << endl << "'time' failed, returning '(time_t)-1':"
      << endl << strerror(errno) << endl <<
    "Failed to get current time." << endl << "Will send failure notice to client."
      << endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    temp_strm.str("");
    temp_strm << "DELETE_HANDLE_VALUE_RESPONSE"
      << GW_ERROR << "\\" << hv_str <<
      "\\-1\\"(Type) "\\"
      << "\\"(Data) "\\"(Failure) "\\"
      << immediate << "\\"
      << delay_value <<
      "UL"
      << purge_database_limit << "U";
    response.command = temp_strm.str();
    response_deque.push_back(response);
  }
  return 1;
} /* if (curr_time ≡ static_cast<time_t>(-1)) */
```

799.

```

⟨Handle_Value_Type::delete_handle_value definition 797⟩ +≡
status = extract(hv_str, prefix, sub_handle, handle, index, type, thread_str);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "ERROR! In 'Handle_Value_Type::delete_handle_value':"
    endl ≪ "'extract' failed, returning" ≪ status ≪ "."
    endl ≪ "Failed to extract fields from handle specifier."
    endl ≪ "Will send failure notice to client."
    endl ≪ "Exiting function unsuccessfully with return value 1."
    endl ≪ endl;
    unlock_cerr_mutex();
    temp_strm.str("");
    temp_strm ≪ "DELETE_HANDLE_VALUE_RESPONSE" ≪ GW_ERROR ≪ " "
    ≪ "\n" ≪ hv_str ≪ "\n-1\"(Type)\n" ≪ "\n(Data)\n\"(Failure)\n" ≪ immediate ≪ " "
    ≪ (curr_time + delay_value) ≪ "UL" ≪ purge_database_limit ≪ "U";
    response.command = temp_strm.str();
    response_deque.push_back(response);
    return 1;
} /* if (status ≠ 0) */

```

800.

```

⟨Handle_Value_Type::delete_handle_value definition 797⟩ +≡
else
if (prefix.empty() ∨ sub_handle.empty() ∨ (index ≤ 0 ∧ type.empty())) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "ERROR! In 'Handle_Value_Type::delete_handle_value':"
    endl ≪ "'extract' failed to extract prefix, non-prefix part of handle, index and/or type."
    endl ≪ "Not enough information to query database for handle."
    endl ≪ "Will send failure notice to client."
    endl ≪ "Exiting function unsuccessfully with return value 1."
    endl ≪ endl;
    unlock_cerr_mutex();
    temp_strm.str("");
    temp_strm ≪ "DELETE_HANDLE_VALUE_RESPONSE" ≪ GW_INVALID_HANDLE_VALUE_SPECIFIER ≪
    " "
    ≪ "\n" ≪ hv_str ≪ "\n-1\"(Type)\n" ≪ "\n(Data)\n\"(Failure)\n" ≪
    immediate ≪ " "
    ≪ (curr_time + delay_value) ≪ "UL" ≪ purge_database_limit ≪ "U";
    response.command = temp_strm.str();
    response_deque.push_back(response);
    return 1;
} /* else if */

```

801.

```

⟨Handle_Value_Type::delete_handle_value definition 797⟩ +≡
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_str << "In 'Handle_Value_Type::delete_handle_value':" <<
                endl << "'extract'_succeeded,_returning_0." << endl <<
                "Extracted_fields_from_handle_specifier_successfully:" << endl <<
                "'prefix'_____==" << prefix << endl << "'sub_handle'____==" << sub_handle << endl <<
                "'index'_____==" << index << endl << "'handle'_____==" << handle << endl <<
                "'type'_____==" << type << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

802.

```

⟨Handle_Value_Type::delete_handle_value definition 797⟩ +≡
if (type ≡ "HS_ADMIN" ∧ ¬(privileges & Scan_Parse_Parameter_Type::SUPERUSER_PRIVILEGE ∨
    privileges & Scan_Parse_Parameter_Type::DELETE_HS_ADMIN_HANDLE_VALUES_PRIVILEGE)) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Handle_Value_Type::delete_handle_value':" <<
        endl << "User_" << username << ",_ID_" << user_id <<
        ",_doesn't_have_the_" << "'delete_hs_admin_handle_value'_privilege." <<
        endl << "Will_send_failure_notice_to_client." << endl <<
        "Exiting_function_unsuccessfully_with_return_value_1." << endl;
    unlock_cerr_mutex();
    temp_strm.str("");
    temp_strm << "DELETE_HANDLE_VALUE_RESPONSE_" << GW_NO_PRIVILEGE_ERROR <<
        "_" << "\"" << hv_str << "\"_" << index << "_" << "\"HS_ADMIN\"_" <<
        "\"(Data)\\"_" << "User_" << username << ",_ID_" << user_id << ",_" <<
        "doesn't_have_the_'delete_hs_admin_handle_value'_privilege.\"_" << immediate <<
        "_" << "OUL_" << purge_database_limit << "U";
    response.command = temp_strm.str();
    response_deque.push_back(response);
    return 1;
} /* if */

```

803.

```

⟨ Handle_Value_Type :: delete_handle_value definition 797 ⟩ +≡
  string handle_database = (standalone_handle) ? "handlesystem_standalone" : "handlesystem";
  sql_strm << "select handle_id, handle_value_id, idx, type, length(data), data, "
  << "ttl_type, ttl, timestamp, length(refs), refs, admin_read, "
  << "admin_write, pub_read, pub_write, marked_for_deletion, created, "
  << "last_modified, handle, created_by_user_id, irods_object_id, "
  << "delete_from_database_timestamp" << "from" << handle_database << ".handles" <<
  "where handle=" << prefix << "/" << sub_handle << ',';
  if (index > 0) sql_strm << "and idx=" << index << ",";
  if (!type.empty()) sql_strm << "and type=" << type << ",";
  sql_strm << "order by handle, type, idx";
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Handle_Value_Type::delete_handle_value':" << endl <<
      "'sql_strm.str()' == " << sql_strm.str() << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

804.

```

⟨ Handle_Value_Type :: delete_handle_value definition 797 ⟩ +≡
  status = submit_mysql_query(sql_strm.str(), result, mysql_ptr, &row_ctr, &field_ctr, 0, thread_str);
  if (status != 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Handle_Value_Type::delete_handle_value':" <<
      endl << "'submit_mysql_query' failed, returning" << status << "." << endl <<
      mysql_error(mysql_ptr) << endl << "Will send failure notice to client." << endl <<
      "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    temp_strm.str("");
    temp_strm << "DELETE_HANDLE_VALUE_RESPONSE" << GW_SERVER_SIDE_DATABASE_ERROR << " "
      << "\" << hv_str << "\"-1\"(Type)\\" << "\"(Data)\\"\"(Failure)\\" << immediate <<
      "\" << (curr_time + delay_value) << "UL" << purge_database_limit << "U";
    response.command = temp_strm.str();
    response_deque.push_back(response);
    return 1;
  } /* if (status != 0) */

```

805.

```

⟨ Handle_Value_Type :: delete_handle_value definition 797 ⟩ +≡
else
  if (row_ctr ≡ 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Handle_Value_Type::delete_handle_value':"
    endl << "'submit_mysql_query' returned 0 rows." << endl <<
    "No matching handle values found in "
    ".handles_database_table." << endl << "Will send failure notice to client."
    endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    temp_strm.str("");
    temp_strm << "DELETE_HANDLE_VALUE_RESPONSE"
    << GW_HANDLE_VALUE_NOT_FOUND << " "
    "\n" << hv_str << "\n";
    if (index > 0) temp_strm << index << " ";
    else temp_strm << "-1 ";
    if (!type.empty()) temp_strm << "\n" << type << "\n";
    else temp_strm << "\n"(Type)\n";
    temp_strm << "\n"(Data)\n\No matching handle value\n"
    << immediate << "\n"
    (curr_time + delay_value) << "UL"
    << purge_database_limit << "U";
    response.command = temp_strm.str();
    response_deque.push_back(response);
    return 1;
  } /* else if (row_ctr ≡ 0) */

```

806.

```

⟨ Handle_Value_Type :: delete_handle_value definition 797 ⟩ +≡
#ifndef DEBUG_COMPILE
else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Handle_Value_Type::delete_handle_value':"
    endl << "'submit_mysql_query' succeeded, returning 0."
    << endl << "'row_ctr' == "
    row_ctr << endl << "'field_ctr' == "
    << field_ctr << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

807.

```

⟨ Handle_Value_Type :: delete_handle_value definition 797 ⟩ +≡
Handle_Value_Type hv;
vector<Handle_Value_Type> hv_vector;
bool hs_admin_found = false; for (int i = 0; i < row_ctr; ++i) { curr_row = mysql_fetch_row(result);
if (curr_row == 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Handle_Value_Type::delete_handle_value':" <<
        endl << "'mysql_fetch_row' failed:" << endl << mysql_error(mysql_ptr) << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
unlock_cerr_mutex();
mysql_free_result(result);
temp_strm.str("");
temp_strm << "DELETE_HANDLE_VALUE_RESPONSE" << GW_SERVER_SIDE_DATABASE_ERROR << " " <<
    "\" " << hv_str << "\" ";
if (index > 0) temp_strm << index << " ";
else temp_strm << "-1 ";
if (!type.empty()) temp_strm << "\" " << type << "\" ";
else temp_strm << "\"(Type)" << "";
temp_strm << "("Data)" " "Server-side database error" " << immediate << " " <<
    (curr_time + delay_value) << "UL" << purge_database_limit << "U";
response.command = temp_strm.str();
response_deque.push_back(response);
return 1;
} /* if (curr_row == 0) */
#endif /* DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Handle_Value_Type::delete_handle_value':" << endl;
    if (curr_row[0]) cerr << "'curr_row[0]' == " << curr_row[0] << endl;
    else cerr << "'curr_row[0]' is NULL." << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
hv.clear();
status = hv.set(curr_row, field_ctr, handle);
if (status != 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Handle_Value_Type::delete_handle_value':" <<
        endl << "'Handle_Value_Type::set' failed, returning " << status <<
        "." << endl << "Failed to set 'Handle_Value_Type' object." << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
unlock_cerr_mutex();
mysql_free_result(result);
temp_strm.str("");
temp_strm << "DELETE_HANDLE_VALUE_RESPONSE" << GW_HANDLE_VALUE_ERROR << " " << "\" " <<
    hv_str << "\" ";
if (index > 0) temp_strm << index << " ";
else temp_strm << "-1 ";
if (!type.empty()) temp_strm << "\" " << type << "\" ";
else temp_strm << "\"(Type)" << "";

```

```
temp_strm << "(Data)\\"_\"(Failure)\\" " << immediate << "U" << (curr_time + delay_value) <<
    "ULU" << purge_database_limit << "U";
response.command = temp_strm.str();
response_deque.push_back(response);
return 1;
} /* if (status != 0) */
```

808.

```
<Handle_Value_Type::delete_handle_value definition 797> +=  
#if DEBUG_COMPILE  
else  
    if (DEBUG) {  
        lock_cerr_mutex();  
        cerr << thread_str << "In 'Handle_Value_Type::delete_handle_value': " << endl <<  
            "'Handle_Value_Type::set' succeeded, returning 0: " << endl;  
#if 0  
    hv.show("hv:");  
#endif  
    unlock_cerr_mutex();  
} /* else if (DEBUG) */  
#endif /* DEBUG_COMPILE */
```

809.

Log

[LDF 2013.09.12.] Added this section.

```

⟨Handle_Value_Type::delete_handle_value definition 797⟩ +≡
if (hv.created_by_user_id ≠ user_id ∧ ¬(privileges &
    Scan_Parse_Parameter_Type::SUPERUSER_PRIVILEGE)) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "WARNING! In 'Handle_Value_Type::delete_handle_value':"
    ≪ endl ≪ "'hv.created_by_user_id' != 'user_id':"
    ≪ endl ≪ "'hv.created_by_user_id' == "
    hv.created_by_user_id ≪ endl ≪ "'user_id' == "
    user_id ≪ endl ≪ "User"
    username ≪ ", ID"
    user_id ≪ ","
    " didn't create this handle value and doesn't have the 'superuser' privilege."
    ≪ endl ≪ "Not pushing 'hv' onto 'hv_vector'."
    ≪ endl ≪ "Continuing."
    ≪ endl;
    unlock_cerr_mutex();
    temp_strm.str("");
    temp_strm ≪ "DELETE HANDLE_VALUE_RESPONSE"
    ≪ GW_NO_PRIVILEGE_ERROR ≪ " "
    ≪ "\"
    hv_str ≪ "\";
    if (hv.idx > 0) temp_strm ≪ hv.idx ≪ " ";
    else temp_strm ≪ "-1";
    temp_strm ≪ "\"
    ≪ hv.type ≪ "\"
    ≪ "(Data)\"
    ≪ "WARNING: "
    ≪ "user"
    username ≪ ","
    ID ≪ user_id ≪ ","
    " didn't create this handle value"
    ≪ endl ≪ "and doesn't have the 'superuser' privilege."
    ≪ endl ≪
    "Handle value not marked for deletion.\"
    ≪ immediate ≪ "
    (curr_time + delay_value) ≪ "UL"
    ≪ purge_database_limit ≪ "U";
    response.command = temp_strm.str();
    response_deque.push_back(response);
    response.command = "";
    temp_strm.str("");
    if (row_ctr ≡ 1) {
        lock_cerr_mutex();
        cerr ≪ thread_str ≪ "In 'Handle_Value_Type::delete_handle_value':"
        ≪ endl ≪
        "Exiting function with return value 3."
        ≪ endl;
        unlock_cerr_mutex();
        mysql_free_result(result);
        return 3;
    }
    else continue;
} /* if */

```

810.

Log

[LDF 2013.09.12.] Added this section.

```

⟨Handle_Value_Type::delete_handle_value definition 797⟩ +≡
if (hv.type ≡ "HS_ADMIN" ∧ ¬(privileges & Scan_Parse_Parameter_Type::SUPERUSER_PRIVILEGE ∨
    (privileges & Scan_Parse_Parameter_Type::DELETE_HS_ADMIN_HANDLE_VALUES_PRIVILEGE)))
{
    lock_cerr_mutex();
    cerr << thread_str << "WARNING! In 'Handle_Value_Type::delete_handle_value':"
    endl << " hv.type == \"HS_ADMIN\" and user " << username << " "
    " doesn't have the 'delete_hs_admin_handle_values' privilege." << endl <<
    " Not pushing 'hv' onto 'hv_vector'." << endl << "Continuing." << endl;
    unlock_cerr_mutex();
    temp_strm.str("");
    temp_strm << "DELETE_HANDLE_VALUE_RESPONSE" << GW_NO_PRIVILEGE_ERROR << " "
    << hv_str << "\"";
    if (hv.idx > 0) temp_strm << hv.idx << " ";
    else temp_strm << "-1";
    temp_strm << "\\"HS_ADMIN\\" << "\\"(Data)\\" << "WARNING: Handle value is "
    of 'type' == 'HS_ADMIN', " << endl << "but user " << username << ", "
    << "ID " << user_id << ", "
    " doesn't have the 'delete_hs_admin_handle_values' privilege." <<
    endl << "Handle value not marked for deletion.\\" << immediate << " "
    (curr_time + delay_value) << "UL" << purge_database_limit << "U";
    response.command = temp_strm.str();
    response_deque.push_back(response);
    response.command = "";
    temp_strm.str("");
    if (row_ctr ≡ 1)
    {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Handle_Value_Type::delete_handle_value':"
        << endl <<
        "Exiting function with return value 3." << endl;
        unlock_cerr_mutex();
        mysql_free_result(result);
        return 3;
    }
    else continue;
} /* if */

```

811.

```

⟨Handle_Value_Type::delete_handle_value definition 797⟩ +≡
hv_vector.push_back(hv); } /* for */
mysql_free_result(result);
result = 0;
sql_strm.str("");

```

812.**Log**

[LDF 2013.09.12.] Added this section.

```
<Handle_Value_Type::delete_handle_value definition 797> +≡
if (hv_vector.size() ≡ 0) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "WARNING! In 'Handle_Value_Type::delete_handle_value':"
    endl ≪ 'hv_vector.size()' ≡ 0 : No handle values found." ≪ endl ≪
    "Exiting function unsuccessfully with return value 3." ≪ endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    temp_strm.str("");
    temp_strm ≪ "DELETE_HANDLE_VALUE_RESPONSE" ≪ GW_HANDLE_VALUE_NOT_FOUND ≪ " "
    " \" ≪ hv_str ≪ "\" ";
    if (index > 0) temp_strm ≪ index ≪ " ";
    else temp_strm ≪ "-1 ";
    if (!type.empty()) temp_strm ≪ " " ≪ type ≪ " ";
    else temp_strm ≪ "(Type)\" ";
    temp_strm ≪ "(Data)\" " ≪ "No handle values found\" "
    ≪ immediate ≪ " "
    (curr_time + delay_value) ≪ "UL" ≪ purge_database_limit ≪ "U";
    response.command = temp_strm.str();
    response_deque.push_back(response);
    return 3;
} /* if (hv_vector.size() ≡ 0) */
```

813.**Log**

[LDF 2013.09.12.] Added this section.

```
<Handle_Value_Type::delete_handle_value definition 797> +≡
else
  if (index > 0 ∧ hv_vector.size() > 1) {
    lock_cerr_mutex();
    cerr << thread_str << "WARNING! In 'Handle_Value_Type::delete_handle_value':"
    endl << "'index' == " << index << " (>0) and 'hv_vector.size()' > 1:"
    endl << "Multiple handle values found for a single index."
    endl << "This shouldn't be possible." << endl <<
      "Exiting function unsuccessfully with return value 3." << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    temp_strm.str("");
    temp_strm << "DELETE HANDLE_VALUE RESPONSE" << GW_HANDLE_VALUE_ERROR << " "
    << hv_str << "\ " << index << " ";
    if (!type.empty()) temp_strm << "\ " << type << " ";
    else temp_strm << "\ "(Type)\ ";
    temp_strm << "\ "(Data) << "\ Multiple handle values found for a single index.\ "
    << "This shouldn't be possible.\ " << immediate << " "
    << (curr_time + delay_value) << "UL "
    << purge_database_limit << "U";
    response.command = temp_strm.str();
    response_deque.push_back(response);
    return 3;
} /* else if */
```

814. In this case, HS_ADMIN will have been specified in the handle value specifier, so all of the handle values of type HS_ADMIN will be on *hv_vector*. Therefore, the user must either be a superuser or have the Scan_Parse_Parameter_Type::DELETE_LAST_HS_ADMIN_HANDLE_VALUE_PRIVILEGE privilege. [LDF 2013.09.12.] ■

Log

[LDF 2013.09.12.] Added this section.

```

⟨ Handle_Value_Type :: delete_handle_value definition 797 ⟩ +≡
if (type ≡ "HS_ADMIN" ∧ index ≤ 0 ∧ ¬(privileges &
    Scan_Parse_Parameter_Type :: SUPERUSER_PRIVILEGE ∨ privileges &
    Scan_Parse_Parameter_Type :: DELETE_LAST_HS_ADMIN_HANDLE_VALUE_PRIVILEGE)) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "WARNING! In 'Handle_Value_Type::delete_handle_value':"
    endl ≪ "'type' == \"HS_ADMIN\" and user " ≪ username ≪ " "
    "doesn't have the 'delete_last_hs_admin_handle_value' privilege."
    endl ≪ "Not marking handle value(s) for deletion."
    endl ≪ "Exiting function with return value 3."
    unlock_cerr_mutex();
    temp_strm.str("");
    temp_strm ≪ "DELETE_HANDLE_VALUE_RESPONSE" ≪ GW_NO_PRIVILEGE_ERROR ≪ " "
    endl ≪ " "
    hv_str ≪ " ";
    if (hv_vector.size() ≡ 1) temp_strm ≪ hv_vector.back().idx ≪ " ";
    else temp_strm ≪ "-1";
    temp_strm ≪ "\\" HS_ADMIN "\\ " ≪ "(Data) \\ \"WARNIN\\"
        G:\\Handle\\value\\type\\'HS_ADMIN' was specified," ≪ endl ≪
        "so all handle values of this type would be deleted." ≪ endl ≪
        "However, user " ≪ username ≪ ", "
        "ID" ≪ user_id ≪ ", "
        "doesn't have the 'delete_last_hs_admin_handle_value' privilege."
        endl ≪ "Handle value(s) not marked for deletion.\\" "
        "immediate ≪ " "
        (curr_time + delay_value) ≪ "UL" ≪ purge_database_limit ≪ "U";
    response.command = temp_strm.str();
    response_deque.push_back(response);
    return 3;
} /* if */

```

815. *index* was specified and *type* \equiv "HS_ADMIN". Check for other handle values of *type* \equiv "HS_ADMIN". In this case, there should only be one **Handle_Value_Type** object on *hv_vector*, so there might be more handle values of type HS_ADMIN for the handle in question. Since the *idx* value was specified, they were not retrieved by the SELECT query above, so we have to query for them here. [LDF 2013.09.12.]

Log

[LDF 2013.09.12.] Added this section.

```

⟨ Handle_Value_Type :: delete_handle_value definition 797 ⟩ +≡
if (index > 0 ∧ hv_vector.back().type ≡ "HS_ADMIN") {
    sql_strm.str("");
    sql_strm ≪ "select_handle_value_id, idx from " ≪ handle_database ≪
        ".handles" ≪ "where handle_id = " ≪ hv_vector.back().handle_id ≪ " " ≪
        "and type = 'HS_ADMIN' and idx <>" ≪ index;
#endif /* DEBUG_COMPILE */
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "sql_strm.str() == " ≪ sql_strm.str() ≪ endl;
    unlock_cerr_mutex();
}
/* if (DEBUG) */
#endif /* DEBUG_COMPILE */
status = submit_mysql_query(sql_strm.str(), result, mysql_ptr, &row_ctr, &field_ctr, 0);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "ERROR! In 'Handle_Value_Type::delete_handle_value':"
        endl ≪ "'submit_mysql_query' failed, returning " ≪ status ≪ "." ≪ endl ≪
        mysql_error(mysql_ptr) ≪ endl ≪ "Will send failure notice to client." ≪ endl ≪
        "Exiting function unsuccessfully with return value 1." ≪ endl;
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    temp_strm.str("");
    temp_strm ≪ "DELETE_HANDLE_VALUE_RESPONSE" ≪ GW_SERVER_SIDE_DATABASE_ERROR ≪
        " " ≪ "\"" ≪ hv_str ≪ "\"" ≪ index ≪ "\"HS_ADMIN\" " ≪
        "\"(Data)\" \"Server-side database error: Query failed.\" " ≪ immediate ≪ " "
        (curr_time + delay_value) ≪ "UL" ≪ purge_database_limit ≪ "U";
    response.command = temp_strm.str();
    response_deque.push_back(response);
    return 1;
} /* if (status ≠ 0) */
#endif /* DEBUG_COMPILE */
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "In 'Handle_Value_Type::delete_handle_value':"
        endl ≪ "'submit_mysql_query' succeeded, returning 0." ≪ endl ≪ "'row_ctr' == "
        row_ctr ≪ endl;
    unlock_cerr_mutex();
}
/* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
if (row_ctr ≡ 0 ∧ ¬(privileges & Scan_Parse_Parameter_Type :: SUPERUSER_PRIVILEGE ∨ privileges &
    Scan_Parse_Parameter_Type :: DELETE_LAST_HS_ADMIN_HANDLE_VALUE_PRIVILEGE)) {
    lock_cerr_mutex();
}

```

```

cerr << thread_str << "WARNING! In 'Handle_Value_Type::delete_handle_value':"
    endl << "'type' == \"HS_ADMIN\", there are no other 'HS_ADMIN' handle values" <<
    "for handle" << hv_vector.back().handle << ',' << endl << "and user" << username <<
    "' doesn't have the 'delete_last_hs_admin_handle_value' privilege." << endl <<
    "Not marking handle value for deletion." << endl <<
    "Exiting function with return value 3." << endl;
unlock_cerr_mutex();
temp_strm.str("");
temp_strm << "DELETE_HANDLE_VALUE_RESPONSE" << GW_NO_PRIVILEGE_ERROR <<
    "\n" << "\n" << hv_str << "\n" << index << "\n\"HS_ADMIN\""
    "\n\"Data\"\n\"WARNING: Handle value with index" << index << "\n"
    "has type 'HS_ADMIN',\n" << endl << "there are no other 'HS_ADMIN' handle values" <<
    "for handle" << hv_vector.back().handle << ',' << endl <<
    "and user" << username << ',', << "ID" << user_id << ",\n"
    "doesn't have the 'delete_last_hs_admin_handle_value' privilege.\n" << endl <<
    "Handle value not marked for deletion.\n" << immediate << "\n"
    (curr_time + delay_value) << "UL" << purge_database_limit << "U";
response.command = temp_strm.str();
response_deque.push_back(response);
return 3;
} /* if */
mysql_free_result(result);
result = 0;
sql_strm.str("");
} /* if (index > 0 & hv_vector.back().type == "HS_ADMIN") */

```

816. Currently, there's no way to distinguish between the `DELAY` option not being used at all or `DELAY 0`. It would make sense to use the latter as a synonym for `IMMEDIATE`, but I can't just change this because the `DELAY` option is used in other rules, too. !! TODO: Look into this. [LDF 2013.09.11.]

```

⟨Handle_Value_Type::delete_handle_value definition 797⟩ +≡
    bool found = false;
    string comma_str;
    sql_strm << "update" << handle_database << ".handles" <<
        "set last_modified=from_unixtime(" << curr_time << "),"
        << "marked_for_deletion=1," << "delete_from_database_timestamp=";
    if (immediate)
        sql_strm << "timestampadd(second,-31622400,from_unixtime(" << curr_time << "))";
    else if (delay_value == 0 || delay_value == 1) { /* See TEx text, above. [LDF 2013.09.11.] */
        sql_strm << "from_unixtime(" << curr_time << ")";
    }
    else sql_strm << "from_unixtime(" << (curr_time + delay_value) << ")";
    sql_strm << "where handle_value_id in(";
    for (vector<Handle_Value_Type>::iterator
        iter = hv_vector.begin(); iter != hv_vector.end(); ++iter) {
#if DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        iter->show("*iter:");
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

817.

```

⟨Handle_Value_Type::delete_handle_value definition 797⟩ +≡
  if (iter→marked_for_deletion) {
    lock_cerr_mutex();
    cerr << thread_str << "WARNING! In 'Handle_Value_Type::delete_handle_value':"
    endl << 'Handle_Value_Type' object already marked for deletion." << endl <<
    "Not marking again." << endl << "Will send response to client and continue." << endl;
    unlock_cerr_mutex();
    temp_strm.str("");
    temp_strm << "DELETE_HANDLE_VALUE_RESPONSE"
    GW_HANDLE_VALUE_ALREADY_MARKED_FOR_DELETION << " "
    << "\n" << hv_str << "\n";
    if (iter→idx > 0) temp_strm << iter→idx << " ";
    else temp_strm << "-1 ";
    if (¬iter→type.empty()) temp_strm << "\n" << iter→type << "\n";
    else temp_strm << "\n"(Type)\n";
    temp_strm << "\n"(Data)\n" Handle_value already marked for deletion.\n"
    "Not marking again.\n" << immediate << " "
    << (curr_time + delay_value) << "UL "
    << purge_database_limit << "U";
    response.command = temp_strm.str();
    response_deque.push_back(response);
    response.command = "";
    continue;
  } /* if (iter→marked_for_deletion) */

```

818.

```

⟨Handle_Value_Type::delete_handle_value definition 797⟩ +≡
  else {
    found = true;
    sql_strm << comma_str << iter→handle_value_id;
    comma_str = ",";
  }
  /* for */
  sql_strm << ")";
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Handle_Value_Type::delete_handle_value':"
    endl << "'sql_strm.str()' == "
    << sql_strm.str() << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

819.

```

⟨Handle_Value_Type::delete_handle_value definition 797⟩ +≡
if (found) { status = submit_mysql_query(sql_strm.str(), result, mysql_ptr, 0, 0, &affected_rows, thread_str);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "ERROR! In 'Handle_Value_Type::delete_handle_value':"
    endl ≪ "'submit_mysql_query' failed, returning" ≪ status ≪ "."
    mysql_error(mysql_ptr) ≪ endl ≪ "Will send failure notice to client."
    endl ≪ "Exiting function unsuccessfully with return value 1."
    endl;
unlock_cerr_mutex();
if (result) mysql_free_result(result);
temp_strm.str("");
temp_strm ≪ "DELETE_HANDLE_VALUE_RESPONSE" ≪ GW_SERVER_SIDE_DATABASE_ERROR ≪ " "
    " \"" ≪ hv_str ≪ "\" ";
if (index > 0) temp_strm ≪ index ≪ " ";
else temp_strm ≪ "-1 ";
if (!type.empty()) temp_strm ≪ " " ≪ type ≪ " ";
else temp_strm ≪ "(Type) ";
temp_strm ≪ "(Data)" ≪ "(Failure)" ≪ immediate ≪ " "
    ≪ (curr_time + delay_value) ≪
    "UL" ≪ purge_database_limit ≪ "U";
response.command = temp_strm.str();
response_deque.push_back(response);
return 1;
} /* if (status ≠ 0) */

```

820.

```

⟨Handle_Value_Type::delete_handle_value definition 797⟩ +≡
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ thread_str ≪ "In 'Handle_Value_Type::delete_handle_value':"
        endl ≪ "'submit_mysql_query' succeeded, returning 0."
        endl ≪ "'affected_rows' == "
        affected_rows ≪ endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
mysql_free_result(result);
result = 0; /* if (found) */

```

821.

```

⟨Handle_Value_Type::delete_handle_value definition 797⟩ +≡
bool binary_data = false;
string temp_str; if (found) { for (vector<Handle_Value_Type>::iterator iter = hv_vector.begin();
    iter ≠ hv_vector.end(); ++iter) {

```

822. These are the handle values that were already marked for deletion and therefore not marked again. Responses were already pushed onto *response deque* for them above. [LDF 2013.09.11.]

```
<Handle_Value_Type::delete_handle_value definition 797> +≡
  if (iter→marked_for_deletion) {
    continue;
  }
```

823.

```
<Handle_Value_Type::delete_handle_value definition 797> +≡
  temp_str = "";
  binary_data = false;
  temp_strm.str("");
  temp_strm ≪ "DELETE_HANDLE_VALUE_RESPONSE" ≪ GW_SUCCESS ≪ "\\" ≪ hv_str ≪ "\" ≪
    iter→idx ≪ "\\" ≪ iter→type ≪ "\" ≪
  if (iter→data_length ≡ 0) temp_strm ≪ "\"(No_data)\\";
```

824. Currently, the size of *string_value* in the (identical) *unions* declared for *yyparse* in *parser.web* and for *zzparse* in *prsrlnt.web* is 1024. This value is used literally in the **union** declaration, i.e., a preprocessor macro is *not* used. (It must certainly be a constant, so a variable cannot be used). Therefore, making the maximum data length here 512 is very conservative. However, I'm not sure at this time whether it's really necessary to send the contents of the data fields to the client at all, so I'm only sending them if they're reasonably short. !! TODO: Maybe define a preprocessor macro for use in the **union** declarations. [LDF 2013.09.11.]

```
<Handle_Value_Type::delete_handle_value definition 797> +≡
  else
    if (iter→data_length > 512) {
      temp_strm ≪ "\"Data:\\" ≪ iter→data_length ≪ "\bytes:\\" ≪
        "Exceeds_maximum_length_for_this_purpose(512_bytes).\\" ≪ "Not_sent.\\"";
    }
```

825.

```

⟨Handle_Value_Type::delete_handle_value definition 797⟩ +≡
else {
    for (int i = 0; i < iter->data_length; ++i) {
        if (!isprint(iter->data[i])) {
            binary_data = true;
            break;
        }
    } /* for */
    if (binary_data == true) temp_strm << "\"Binary\u0020data.\u0020Not\u0020sent.\"";
    else {
        temp_str.append(iter->data, iter->data_length);
        temp_strm << "\\" << temp_str << "\\";
    }
} /* else */
temp_strm << "\"Success\"" << immediate << "\\";
if (immediate) temp_strm << "OUL";
else temp_strm << (curr_time + delay_value) << "UL";
temp_strm << purge_database_limit << "U";
response.command = temp_strm.str();
response_deque.push_back(response);
temp_strm.str(""); } /* for */
} /* if (found) */

```

826.

```

⟨Handle_Value_Type::delete_handle_value definition 797⟩ +≡
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "WARNING! In 'Handle_Value_Type::delete_handle_value':"
        endl << "'found' == 'false': No handle values marked for deletion."
        endl << "Continuing." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

827.

```

⟨Handle_Value_Type::delete_handle_value definition 797⟩ +≡
  if (found ∧ immediate ∧ purge_database_interval > 0) {
    pthread_mutex_lock(&purge_server_database_mutex);
    status = pthread_cond_signal(&purge_server_database_cond);
    if (status ≠ 0) {
      lock_cerr_mutex();
      cerr ≪ thread_str ≪ "ERROR! In 'Handle_Value_Type::delete_handle_value':"
      ≪ endl ≪ "'pthread_cond_signal' failed, returning" ≪ status ≪ ":" ≪ endl ≪
      strerror(status) ≪ endl ≪ "Exiting function unsuccessfully with return value 3."
      ≪ endl;
      unlock_cerr_mutex();
      pthread_mutex_unlock(&purge_server_database_mutex);
      return 3;
    } /* if (status ≠ 0) */
  #if DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr ≪ thread_str ≪ "In 'Handle_Value_Type::delete_handle_value':"
      ≪ endl ≪ "'pthread_cond_signal' succeeded, returning 0." ≪ endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
  #endif /* DEBUG_COMPILE */
  pthread_mutex_unlock(&purge_server_database_mutex);
} /* if (found ∧ immediate ∧ purge_database_interval > 0) */

```

828.

```

⟨Handle_Value_Type::delete_handle_value definition 797⟩ +≡
  #if DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr ≪ thread_str ≪ "In 'Handle_Value_Type::delete_handle_value':"
      ≪ endl ≪ "'found' == 'false' and/or 'immediate' == 'false'" ≪
      "and/or 'purge_database_interval' == 0:" ≪ endl ≪
      "'found' == " ≪ found ≪ endl ≪
      "'immediate' == " ≪ immediate ≪ endl ≪
      "'purge_database_interval' == " ≪ purge_database_interval ≪ endl ≪
      "Not waking up 'purge_server_database' thread." ≪ endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
  #endif /* DEBUG_COMPILE */

```

829.

```
<Handle_Value_Type::delete_handle_value definition 797> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Exiting 'Handle_Value_Type::delete_handle_value' <" <<
            "successfully with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; } /* End of Handle_Value_Type::delete_handle_value definition */
```

830. Unmark handle value for deletion (*unmark_handle_value_for_deletion*). [LDF 2013.09.11.]

Log

[LDF 2013.09.11.] Added this function.

[LDF 2013.09.12.] Removed arguments **char *buffer_ptr** and **size_t buffer_size**.

```
<Handle_Value_Type function declarations 754> +≡
static int unmark_handle_value_for_deletion (MYSQL *&mysql_ptr, string hv_str, deque <
    Response_Type > &response_deque, int user_id, string username, unsigned
    int privileges, string thread_str = "" );
```

831.

```
<Handle_Value_Type::unmark_handle_value_for_deletion definition 831> ≡
int Handle_Value_Type::unmark_handle_value_for_deletion (MYSQL *&mysql_ptr, string hv_str,
    deque <Response_Type> &response_deque, int user_id, string username,
    unsigned int privileges, string thread_str ) { bool DEBUG = false;
    /* true */
    set_debug_level(DEBUG, 0, 0);
    stringstream temp_strm;
    int status;
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering 'Handle_Value_Type::unmark_han<
            dle_value_for_deletion'." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

See also sections 832, 833, 834, 835, 836, 837, 838, 839, 840, 841, 842, 843, 844, 845, 846, 847, 848, 849, 850, 851, 852, 853, and 854.

This code is used in section 883.

832.

```
<Handle_Value_Type::unmark_handle_value_for_deletion definition 831> +≡
  stringstream sql_strm;
  MYSQL_RES *result = 0;
  MYSQL_ROW curr_row;
  unsigned int row_ctr = 0;
  unsigned int field_ctr = 0;
  long affected_rows = 0;
  string prefix;
  string sub_handle;
  string handle;
  int index = -1;
  string type;
  Response_Type response;
  response.type = Response_Type::COMMAND_ONLY_TYPE;
```

833.

```
<Handle_Value_Type::unmark_handle_value_for_deletion definition 831> +≡
  status = extract(hv_str, prefix, sub_handle, handle, index, type, thread_str);
  if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Handle_Value_Type::unmark_handle_value_for_deletion':"
    endl << "'extract' failed, returning " << status << "."
    endl << "Failed to extract fields from handle specifier."
    endl << "Will send failure notice to client."
    endl << "Exiting function unsuccessfully with return value 1."
    endl;
    unlock_cerr_mutex();
    temp_strm.str("");
    temp_strm << "UNDELETE_HANDLE_VALUE_RESPONSE"
    << GW_HANDLE_VALUE_ERROR << "\"
    hv_str << "\"-1\"(Type)\"
    << "\"(Data)\"
    << "(Failure)\";
    response.command = temp_strm.str();
    response.deque.push_back(response);
    return 1;
  } /* if (status ≠ 0) */
```

834.

```
<Handle_Value_Type::unmark_handle_value_for_deletion definition 831> +≡
else
  if (prefix.empty() ∨ sub_handle.empty() ∨ (index ≤ 0 ∧ type.empty())) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Handle_Value_Type::unmark_h\
andle_value_for_deletion':" << endl << "'extract' failed to \
extract_prefix, non-prefix part of handle, index and/or type." <<
    endl << "Not enough information to query database for handle." <<
    endl << "Will send failure notice to client." << endl <<
    "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    temp_strm.str("");
    temp_strm << "UNDELETE_HANDLE_VALUE_RESPONSE" << GW_INVALID_HANDLE_VALUE_SPECIFIER <<
      "\" << hv_str << "\"-1\"(Type)\" " << "\"(Data)\" \"(Failure)\"";
    response.command = temp_strm.str();
    response_deque.push_back(response);
    return 1;
} /* else if */
```

835.

```
<Handle_Value_Type::unmark_handle_value_for_deletion definition 831> +≡
#ifndef DEBUG_COMPILE
else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Handle_Value_Type::unmark_handle_value_for_deletion':" <<
      endl << "'extract' succeeded, returning 0." << endl <<
      "Extracted fields from handle specifier successfully:" << endl <<
      "'prefix'" << prefix << endl << "'sub_handle'" << sub_handle << endl <<
      "'index'" << index << endl << "'handle'" << handle << endl <<
      "'type'" << type << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

836.

```
<Handle_Value_Type::unmark_handle_value_for_deletion definition 831> +≡
```

837.

```

⟨ Handle_Value_Type :: unmark_handle_value_for_deletion definition 831 ⟩ +≡
  string handle_database = (standalone_handle) ? "handlesystem_standalone" : "handlesystem";
  sql_strm << "select handle_id, handle_value_id, idx, type, length(data), data, "
  << "ttl_type, ttl, timestamp, length(refs), refs, admin_read, "
  << "admin_write, pub_read, pub_write, marked_for_deletion, created, "
  << "last_modified, handle, created_by_user_id, irods_object_id, "
  << "delete_from_database_timestamp" << "from" << handle_database <<
  ".handles where handle=" << ',' << prefix << "/" << sub_handle << ',';
  if (index > 0) sql_strm << "and idx=" << index << ",";
  if (!type.empty()) sql_strm << "and type=" << type << ',';
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Handle_Value_Type::unmark_handle_value_for_deletion':" << endl <<
      "'sql_strm.str()'==" << sql_strm.str() << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

838.

```

⟨ Handle_Value_Type :: unmark_handle_value_for_deletion definition 831 ⟩ +≡
  status = submit_mysql_query(sql_strm.str(), result, mysql_ptr, &row_ctr, &field_ctr, 0, thread_str);
  if (status != 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Handle_Value_Type::unmark_handle_value_for_deletion':" <<
      endl << "'submit_mysql_query' failed, returning" << status << "." << endl <<
      mysql_error(mysql_ptr) << endl << "Will send failure notice to client." << endl <<
      "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    temp_strm.str("");
    temp_strm << "UNDELETE_HANDLE_VALUE_RESPONSE" << GW_SERVER_SIDE_DATABASE_ERROR << " "
      << "\" " << hv_str << "\" -1 "\" (Type) "\" " << "\" (Data) "\" \" (Failure) "\" ";
    response.command = temp_strm.str();
    response.deque.push_back(response);
    return 1;
  } /* if (status != 0) */

```

839.

```

⟨ Handle_Value_Type::unmark_handle_value_for_deletion definition 831 ⟩ +≡
else
  if (row_ctr ≡ 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Handle_Value_Type::unmark_handle_value_for_dele"
        "tion'" << endl << "'submit_mysql_query' returned 0 rows." <<
        endl << "No matching handle values found in " << handle_database <<
        ".handles_database_table." << endl << "Will send failure notice to client." <<
        endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    temp_strm.str("");
    temp_strm << "UNDELETE_HANDLE_VALUE_RESPONSE" << GW_HANDLE_VALUE_NOT_FOUND << " "
        "\n" << hv_str << "\n";
    if (index > 0) temp_strm << index << " ";
    else temp_strm << "-1 ";
    if (!type.empty()) temp_strm << "\n" << type << "\n";
    else temp_strm << "\n"(Type)\n";
    temp_strm << "\n"(Data)\n"No matching handle value\n";
    response.command = temp_strm.str();
    response_deque.push_back(response);
    return 1;
} /* else if (row_ctr ≡ 0) */

```

840.

```

⟨ Handle_Value_Type::unmark_handle_value_for_deletion definition 831 ⟩ +≡
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << thread_str << "In 'Handle_Value_Type::unmark_handle_value_for_deletion':"
          endl << "'submit_mysql_query' succeeded, returning 0." << endl << "'row_ctr' == "
          row_ctr << endl << "'field_ctr' == " << field_ctr << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

841.

```

⟨ Handle_Value_Type::unmark_handle_value_for_deletion definition 831 ⟩ +≡
Handle_Value_Type hv;
vector<Handle_Value_Type> hv_vector;  for (int i = 0; i < row_ctr; ++i) {
    curr_row = mysql_fetch_row(result);
    if (curr_row == 0) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'Handle_Value_Type::unmark_handle_value_for_deletion':"
        endl << "'mysql_fetch_row' failed:" << endl << mysql_error(mysql_ptr) << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        mysql_free_result(result);
        temp_strm.str("");
        temp_strm << "UNDELETE_HANDLE_VALUE_RESPONSE" << GW_SERVER_SIDE_DATABASE_ERROR << " "
        << "\" << hv_str << "\" ";
        if (index > 0) temp_strm << index << " ";
        else temp_strm << "-1 ";
        if (!type.empty()) temp_strm << "\"" << type << "\" ";
        else temp_strm << "("Type)" ";
        temp_strm << "("Data)\" "Server-side_database_error\"";
        response.command = temp_strm.str();
        response.deque.push_back(response);
        return 1;
    } /* if (curr_row == 0) */
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Handle_Value_Type::unmark_handle_value_for_deletion':"
    if (curr_row[0]) cerr << "'curr_row[0]' == " << curr_row[0] << endl;
    else cerr << "'curr_row[0]' is NULL." << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
hv.clear();
status = hv.set(curr_row, field_ctr, handle);
if (status != 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Handle_Value_Type::unmark_handle_value_for_deletion':"
    endl << "'Handle_Value_Type::set' failed, returning "
    << status << ". "
    << endl << "Failed to set 'Handle_Value_Type' object."
    << endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    temp_strm.str("");
    temp_strm << "UNDELETE_HANDLE_VALUE_RESPONSE" << GW_HANDLE_VALUE_ERROR << " "
    << "\" << hv_str << "\" ";
    if (index > 0) temp_strm << index << " ";
    else temp_strm << "-1 ";
    if (!type.empty()) temp_strm << "\"" << type << "\" ";
    else temp_strm << "("Type)" ";
    temp_strm << "("Data)\" "(Failure)" ";
    response.command = temp_strm.str();
}

```

```

    response_deque.push_back(response);
    return 1;
} /* if (status != 0) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Handle_Value_Type::unmark_handle_value_for_deletion':" <<
            endl << "'Handle_Value_Type::set' succeeded, returning 0:" << endl;
#endif 0
    hv.show("hv:");
#endif
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

842.

```

⟨Handle_Value_Type::unmark_handle_value_for_deletion definition 831⟩ +≡
if (hv.created_by_user_id ≠ user_id ∧ ¬(privileges &
    Scan_Parse_Parameter_Type::SUPERUSER_PRIVILEGE)) {
    temp_strm << "UNDELETE_HANDLE_VALUE_RESPONSE" << GW_NO_PRIVILEGE_ERROR << " " << """
    hv_str << " " << hv.idx << " " << hv.type << " " << "(Data)" "User" " <<
    username << ",ID" << user_id << ", " << "didn't create this handle value" <<
    "and doesn't have 'superuser' privilege:" << endl << "'created_by_user_id' == " <<
    hv.created_by_user_id << ". " << endl << "Handle value not unmarked for deletion."";
    response.command = temp_strm.str();
    response_deque.push_back(response);
    response.command = "";
    continue;
} /* if */

```

843.

```

⟨Handle_Value_Type::unmark_handle_value_for_deletion definition 831⟩ +≡
    hv_vector.push_back(hv); } /* for */
    mysql_free_result(result);
    result = 0;
    sql_strm.str("");

```

844.

```
⟨ Handle_Value_Type :: unmark_handle_value_for_deletion definition 831 ⟩ +≡
  bool found = false;
  string comma_str;
  sql strm ≪ "update" ≪ handle_database ≪ ".handles" ≪
  "set last_modified=now(),marked_for_deletion=0" ≪
  "delete_from_database_timestamp=0" ≪ "where handle_value_id in ("; for
  (vector<Handle_Value_Type>::iterator iter = hv_vector.begin(); iter ≠ hv_vector.end();
  ++iter) {
#if DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    iter->show("*iter:");
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

845.

```

⟨ Handle_Value_Type :: unmark_handle_value_for_deletion definition 831 ⟩ +≡
  if (iter→marked_for_deletion ≡ false) {
    lock_cerr_mutex();
    cerr << thread_str << "WARNING! In 'Handle_Value_Type::unmark_handle_value_for_de\"
        lection'" << endl << "'Handle_Value_Type' object not marked for deletion." << endl <<
        "Not unmarking." << endl << "Will send response to client and continue." << endl;
    unlock_cerr_mutex();
    temp_strm.str("");
    temp_strm << "UNDELETE_HANDLE_VALUE_RESPONSE" <<
        GW_HANDLE_VALUE_NOT_MARKED_FOR_DELETION << " " << "\n" << hv_str << "\n";
    if (iter→idx > 0) temp_strm << iter→idx << " ";
    else temp_strm << "-1 ";
    if (¬iter→type.empty()) temp_strm << "\n" << iter→type << "\n";
    else temp_strm << "\n"(Type)\n";
    temp_strm << "\n"(Data)\n\Handle_value not marked for deletion.\n" <<
        "Not unmarking.\n";
    response.command = temp_strm.str();
    response_deque.push_back(response);
    response.command = "";
    continue;
} /* if (iter→marked_for_deletion ≡ false) */
else {
  found = true;
  sql_strm << comma_str << iter→handle_value_id;
  comma_str = ",";
}
} /* for */
sql_strm << ")";
#endif DEBUG_COMPILE
if (DEBUG) {
  lock_cerr_mutex();
  cerr << thread_str << "In 'Handle_Value_Type::unmark_handle_value_for_deletion':" << endl <<
      "'sql_strm.str()' == " << sql_strm.str() << endl;
  unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

846.

```

⟨ Handle_Value_Type::unmark_handle_value_for_deletion definition 831 ⟩ +≡
if (found) { status = submit_mysql_query(sql_strm.str(), result, mysql_ptr, 0, 0, &affected_rows, thread_str);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "ERROR! In 'Handle_Value_Type::unmark_handle_value_for_deletion':"
    endl ≪ "'submit_mysql_query' failed, returning" ≪ status ≪ "." ≪ endl ≪
    mysql_error(mysql_ptr) ≪ endl ≪ "Will send failure notice to client." ≪ endl ≪
    "Exiting function unsuccessfully with return value 1." ≪ endl;
unlock_cerr_mutex();
if (result) mysql_free_result(result);
temp_strm.str("");
temp_strm ≪ "UNDELETE_HANDLE_VALUE_RESPONSE" ≪ GW_SERVER_SIDE_DATABASE_ERROR ≪ " "
    " \" " ≪ hv_str ≪ " \" ";
if (index > 0) temp_strm ≪ index ≪ " ";
else temp_strm ≪ "-1 ";
if (!type.empty()) temp_strm ≪ " "(Type) " ";
else temp_strm ≪ " "(Data) " " "(Failure) " ";
response.command = temp_strm.str();
response_deque.push_back(response);
return 1;
} /* if (status ≠ 0) */

```

847.

```

⟨ Handle_Value_Type::unmark_handle_value_for_deletion definition 831 ⟩ +≡
#ifndef DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "In 'Handle_Value_Type::unmark_handle_value_for_deletion':"
    endl ≪ "'submit_mysql_query' succeeded, returning 0." ≪ endl ≪
    "'affected_rows' == " ≪ affected_rows ≪ endl;
unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
mysql_free_result(result);
result = 0; } /* if (found) */

```

848.

```

⟨ Handle_Value_Type::unmark_handle_value_for_deletion definition 831 ⟩ +≡
bool binary_data = false;
string temp_str; if (found) { for (vector<Handle_Value_Type>::iterator iter = hv_vector.begin();
    iter ≠ hv_vector.end(); ++iter) {

```

849. These are the handle values that were not marked for deletion before and therefore not unmarked. Responses were already pushed onto *response deque* for them above. [LDF 2013.09.11.]

```

⟨ Handle_Value_Type::unmark_handle_value_for_deletion definition 831 ⟩ +≡
if (iter→marked_for_deletion ≡ false) {
    continue;
}

```

850.

```
⟨Handle_Value_Type::unmark_handle_value_for_deletion definition 831⟩ +≡
temp_str = "";
binary_data = false;
temp_strm.str("");
temp_strm << "UNDELETE_HANDLE_VALUE_RESPONSE" << GW_SUCCESS << "\u" << "\\" << hv_str <<
"\\" << iter->idx << "\\" << iter->type << "\u";
if (iter->data->length == 0) temp_strm << "\"(No_data)\\";
```

851. Currently, the size of *string-value* in the (identical) *unions* declared for *yyparse* in *parser.web* and for *zzparse* in *prsrclnt.web* is 1024. This value is used literally in the **union** declaration, i.e., a preprocessor macro is *not* used. (It must certainly be a constant, so a variable cannot be used). Therefore, making the maximum data length here 512 is very conservative. However, I'm not sure at this time whether it's really necessary to send the contents of the data fields to the client at all, so I'm only sending them if they're reasonably short. !! TODO: Maybe define a preprocessor macro for use in the **union** declarations. [LDF 2013.09.11.]

```
⟨Handle_Value_Type::unmark_handle_value_for_deletion definition 831⟩ +≡
else
if (iter->data->length > 512) {
    temp_strm << "\"Data:\u" << iter->data->length << "\ubytes:\uu" <<
        "Exceeds\umaximum\ulength\ufor\uthis\upurpose\u(512\ubytes).uu" << "Not\usent.\\";
}
```

852.

```
⟨Handle_Value_Type::unmark_handle_value_for_deletion definition 831⟩ +≡
else {
    for (int i = 0; i < iter->data->length; ++i) {
        if (!isprint(iter->data[i])) {
            binary_data = true;
            break;
        }
    /* for */
    if (binary_data == true) temp_strm << "\"Binary\data.\uNot\usent.\\";
    else {
        temp_str.append(iter->data, iter->data->length);
        temp_strm << "\\" << temp_str << "\u";
    }
} /* else */
temp_strm << "\"Success\"";
response.command = temp_strm.str();
response_deque.push_back(response);
temp_strm.str("");
} /* if (found) */
```

853.

```
<Handle_Value_Type::unmark_handle_value_for_deletion definition 831> +≡
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_str << "WARNING! In 'Handle_Value_Type::unmark_handle_value_for_de\"
                lection'" << endl << "'found' == 'false': No handle values unmarked for deletion\
                ." << endl << "Continuing." << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

854.

```
<Handle_Value_Type::unmark_handle_value_for_deletion definition 831> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Exiting 'Handle_Value_Type::unmark_handle_value_for_deletion'" <<
            "successfully with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
return 0; } /* End of Handle_Value_Type::unmark_handle_value_for_deletion definition */
```

855. Extract (*extract*). [LDF 2013.08.30.]

Log

[LDF 2013.08.30.] Added this function.

[LDF 2013.09.11.] Added **string** &**handle** argument.

```
<Handle_Value_Type function declarations 754> +≡
static int extract(string hv_str, string &prefix, string &sub_handle, string &handle, int
&index, string &type, string thread_str = "");
```

856.

```

⟨Handle_Value_Type::extract definition 856⟩ ≡
int Handle_Value_Type::extract(string hv_str, string &prefix, string &sub_handle, string
                                &handle, int &index, string &type, string thread_str){ bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    string temp_str;
    unsigned long int temp_val = 0L;
    size_t pos = 0;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering 'Handle_Value_Type::extract'." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

See also sections 857, 858, 859, 860, 861, 862, 863, 864, 865, 866, 867, and 868.

This code is used in section 883.

857.

```

⟨Handle_Value_Type::extract definition 856⟩ +≡
pos = hv_str.find("/");
if (pos == string::npos) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Handle_Value_Type::extract':"
    << endl << "Failed to find slash in handle specifier:" << endl << "'hv_str' == "
    << hv_str << endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    return 1;
} /* if (pos == string::npos) */

```

858.

⟨Handle_Value_Type::extract definition 856⟩ +≡

859. Currently, it's not possible for this case to occur, because a string like "12345/00001:" would cause a parse error, since the parser would interpret it as an invalid ⟨time specification⟩. [LDF 2013.09.10.]

Log

[LDF 2013.09.10.] Added this section.

```

⟨Handle_Value_Type::extract definition 856⟩ +≡
if (pos ≥ hv_str.size() - 1) {
    cerr << thread_str << "ERROR! In 'Handle_Value_Type::extract':"
    << endl << "Slash is last character in handle specifier:" << endl <<
    "'hv_str' == "
    << hv_str << endl << "This isn't permitted." << endl <<
    "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    return 1;
} /* if (pos ≥ hv_str.size() - 1) */

```

860.

```
<Handle_Value_Type::extract definition 856> +≡
  prefix = hv_str;
  prefix.erase(pos);
  hv_str.erase(0, pos + 1);
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Handle_Value_Type::extract':" << endl << "prefix=="
      << prefix << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

861.

```
<Handle_Value_Type::extract definition 856> +≡
  pos = hv_str.find(":");
  if (pos == string::npos) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Handle_Value_Type::extract':" << endl <<
      "Failed to find colon in handle_value_specifier:" << endl << "'hv_str'=="
      hv_str << endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    return 1;
  } /* if (pos == string::npos) */
```

862.

```
<Handle_Value_Type::extract definition 856> +≡
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << thread_str << "In 'Handle_Value_Type::extract':" << endl <<
        "Found colon in handle_value_specifier." << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

863. Currently, it's not possible for this case to occur, because a string like "12345/00001:" would cause a parse error, since the parser would interpret it as an invalid <time specification>. [LDF 2013.09.10.]

Log

[LDF 2013.09.10.] Added this section.

```
<Handle_Value_Type::extract definition 856> +≡
if (pos ≥ hv_str.size() - 1) {
    cerr << thread_str << "ERROR! In 'Handle_Value_Type::extract':"
    << endl << "Failed to find handle_value_specifier:" << endl << "'hv_str' == "
    << hv_str << endl << "Exiting function unsuccessfully with return value 1."
    << endl;
    unlock_cerr_mutex();
    return 1;
} /* if (pos ≥ hv_str.size() - 1) */
```

864.

```
<Handle_Value_Type::extract definition 856> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Handle_Value_Type::extract':"
        << endl << "hv_str == "
        << hv_str << endl << "pos == "
        << pos << endl << "hv_str.size() == "
        << hv_str.size() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    temp_str = hv_str;
    temp_str.erase(0, pos + 1);
    hv_str.erase(pos);
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Handle_Value_Type::extract':"
        << endl << "hv_str == "
        << hv_str << endl << "temp_str == "
        << temp_str << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

865.

```

⟨Handle_Value_Type::extract definition 856⟩ +≡
  if (isdigit(temp_str[0])) {
    errno = 0;
    temp_val = strtoul(temp_str.c_str(), 0, 10);
    if (temp_val == ULONG_MAX) {
      cerr << thread_str << "ERROR! In 'Handle_Value_Type::extract':"
      << endl << "'strtoul' failed, returning 'ULONG_MAX':"
      << endl << strerror(errno) << endl <<
      "Exiting function unsuccessfully with return value 1." << endl;
      unlock_cerr_mutex();
      return 1;
    }
    else if (temp_val > INT_MAX) {
      cerr << thread_str << "ERROR! In 'Handle_Value_Type::extract':"
      << endl << "'strtoul' returned"
      << temp_val << " (> 'INT_MAX')."
      << endl << "This is not permitted: Handle index ('idx') exceeds maximum value."
      << endl << "Exiting function unsuccessfully with return value 1." << endl;
      unlock_cerr_mutex();
      return 1;
    } /* else if (temp_val > INT_MAX) */
    else {
      index = temp_val;
      type = "";
    }
  #if DEBUG_COMPILE
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << thread_str << "In 'Handle_Value_Type::extract':"
      << endl << "'index' == "
      << index << endl;
      unlock_cerr_mutex();
    } /* if (DEBUG) */
  #endif /* DEBUG_COMPILE */
  } /* else */
} /* if (isdigit(temp_str[0])) */

```

866.

```

⟨Handle_Value_Type::extract definition 856⟩ +≡
  else {
    type = temp_str;
    index = 0;
  #if DEBUG_COMPILE
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << thread_str << "In 'Handle_Value_Type::extract':"
      << endl << "'type' == "
      << type << endl;
      unlock_cerr_mutex();
    } /* if (DEBUG) */
  #endif /* DEBUG_COMPILE */
  } /* else */

```

867.

```
<Handle_Value_Type::extract definition 856> +≡
    sub_handle = hv_str;
    handle = prefix;
    handle += "/";
    handle += sub_handle;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Handle_Value_Type::extract':" << endl << "'sub_handle' == " <<
            sub_handle << endl << "'handle'" << endl << handle << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

868.

```
<Handle_Value_Type::extract definition 856> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Exiting 'Handle_Value_Type::extract' successfully" <<
            "with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; } /* End of Handle_Value_Type::extract definition */
```

869. Clear. [LDF 2012.10.12.]

Log

[LDF 2012.10.12.] Added this function.

[LDF 2013.02.07.] BUG FIX: Now setting *data_length* and *refs_length* to 0.

```
<Handle_Value_Type function declarations 754> +≡
    void clear(void);
```

870.

```

⟨Handle_Value_Type::clear definition 870⟩ ≡
void Handle_Value_Type::clear(void)
{
    filename = "";
    handle = "";
    idx = 0;
    type = "";
    if (data) {
        delete[] data;
        data = 0;
    }
    data_length = 0;
    ttl_type = 0;
    ttl = 0;
    timestamp = 0;
    if (refs) {
        delete[] refs;
        refs = 0;
    }
    refs_length = 0;
    admin_read = true;
    admin_write = true;
    pub_read = true;
    pub_write = false;
    handle_id = 0;
    handle_value_id = 0;
    created_by_user_id = 0;
    irods_object_id = 0UL;
    marked_for_deletion = false;
    created = static_cast<time_t>(0);
    last_modified = static_cast<time_t>(0);
    created_str = "";
    last_modified_str = "";
    delete_from_database_timestamp = static_cast<time_t>(0);
    delete_from_database_timestamp_str = "";
    return;
}      /* End of Handle_Value_Type::clear definition */

```

This code is used in section 883.

871. Show. [LDF 2012.10.12.]

Log

[LDF 2012.10.12.] Added this function.
 [LDF 2013.01.07.] Added optional argument **stringstream** *strm = 0. Now calling **stringstream** ::*write* instead of *fwrite*.

⟨Handle_Value_Type function declarations 754⟩ +≡

void show(string s = "Handle_Value_Type:", stringstream *strm = 0) const;

872.

```

⟨ Handle_Value_Type :: show definition 872 ⟩ ≡
void Handle_Value_Type :: show(string s, stringstream *strm) const
{
    stringstream temp_strm;
    temp_strm << s << endl;
    temp_strm << "handle" << handle <<
        endl << "filename" << filename <<
        endl << "idx" << idx << endl <<
        "type" << type << endl;
    if (data & data_length > 0 & strlen(data) > 0) {
        temp_strm << "data";
        temp_strm.write(data, data_length);
    #if 0
        fwrite(data, 1, data_length, stderr);
    #endif
    }
    else temp_strm << "data" << NULL;
    temp_strm << endl << "data_length" << data_length <<
        endl << "ttl_type" << ttl_type <<
        endl << "ttl" << ttl << endl <<
        "timestamp" << timestamp;
    if (timestamp > 0) temp_strm << "(" << convert_seconds(timestamp) << ")";
    temp_strm << endl;
    if (refs) {
        temp_strm << "refs";
        temp_strm.write(refs, refs_length);
    }
    else temp_strm << "refs" << NULL;
    temp_strm << endl << "refs_length" << refs_length <<
        endl << "admin_read" << admin_read <<
        endl << "admin_write" << admin_write <<
        endl << "pub_read" << pub_read << endl <<
        "pub_write" << pub_write << endl <<
        "handle_id" << handle_id << endl <<
        "handle_value_id" << handle_value_id << endl <<
        "created_by_user_id" << created_by_user_id << endl <<
        "irods_object_id" << irods_object_id << endl <<
        "marked_for_deletion" << marked_for_deletion <<
        endl << "created" << created << endl <<
        "created_str" << created_str << endl <<
        "last_modified" << last_modified << endl <<
        "last_modified_str" << last_modified_str << endl <<
        "delete_from_database_timestamp" << delete_from_database_timestamp << endl <<
        "delete_from_database_timestamp_str" << delete_from_database_timestamp_str << endl <<
        endl;
    if (strm == 0) cerr << temp_strm.str();
    else *strm << temp_strm.str();
    return;
} /* End of Handle_Value_Type :: show */

```

This code is used in section 883.

873. Write to database (normally client-side) [LDF 2013.04.26.]

Log

[LDF 2013.04.26.] Added this function.

[LDF 2013.05.10.] Added optional argument **bool** *replace* = *false*. If *replace* ≡ *true*, the SQL command **REPLACE** is used rather than **INSERT**.

⟨ **Handle_Value_Type** function declarations 754 ⟩ +≡

```
int write_to_database(MYSQL *mysql_ptr, string database, bool replace);
```

874.

⟨ **Handle_Value_Type**::*write_to_database* definition 874 ⟩ ≡

```
int Handle_Value_Type::write_to_database(MYSQL *mysql_ptr, string database, bool replace){ int
    status = 0;
    int ret_val = 0;
    bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    stringstream sql_strm;
    MYSQL_RES *result = 0;
    long affected_rows = 0;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Entering 'Handle_Value_Type::write_to_database'." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
}
```

See also sections 875, 876, 877, 878, 879, 880, and 881.

This code is used in section 883.

875.

```
<Handle_Value_Type::write_to_database definition 874> +≡
status = submit_mysql_query("lock_tables\gwirdcli.handles\write", result, mysql_ptr);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ "ERROR! In 'Handle_Value_Type::write_to_database':"
    ≪ endl ≪ "'submit_mysql_query'\u2014failed,\u2014returning" ≪ status ≪ ":" ≪ endl ≪
    mysql_error(mysql_ptr) ≪ endl ≪ "Failed\uto\lock\utable\gwirdcli.handles'." ≪ endl ≪
    "Exiting\ufuncunsuccessfully\with\return\value\1." ≪ endl;
    unlock_cerr_mutex();
if (result) {
    mysql_free_result(result);
    result = 0;
}
return 1;
} /* if (status ≠ 0) */
#if DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "In 'Handle_Value_Type::write_to_database':"
    ≪ endl ≪ "'submit_mysql_query'\u2014succeeded,\u2014returning\0." ≪ endl ≪ mysql_error(mysql_ptr) ≪
    endl ≪ "Locked\utable\gwirdcli.handles'\u2014successfully." ≪ endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
if (result) {
    mysql_free_result(result);
    result = 0;
}
```

876.

```

⟨ Handle_Value_Type :: write_to_database definition 874 ⟩ +≡
  string temp_str;
  sql_strm.str("");
  if (replace) sql_strm ≪ "replace";
  else sql_strm ≪ "insert";
  sql_strm ≪ "into gwirdcli.handles (handle, idx, "
  ≪ "type, data, ttl_type, ttl, timestamp, refs, admin_read, "
  ≪ "admin_write, pub_read, pub_write, handle_id, handle_value_id, "
  ≪ "created_by_user_id, irods_object_id, marked_for_deletion, created, "
  ≪ "last_modified, delete_from_database_timestamp) "
  ≪ "values (" ≪ "','" ≪ handle ≪
  "','" ≪ idx ≪ ", " ≪ "','" ≪ type ≪ ", ";
  if (data ≡ 0 ∨ data.length ≡ 0 ∨ strlen(data) ≡ 0) sql_strm ≪ "NULL";
  else {
    temp_str = "";
    status = hexl_encode(data, data.length, temp_str);
    if (status ≠ 0) {
      lock_cerr_mutex();
      cerr ≪ "ERROR! In 'Handle_Value_Type::write_to_database':"
      ≪ endl ≪
        "'hexl_encode' failed, returning " ≪ status ≪ "."
      ≪ endl ≪
        "Failed to encode contents of 'data' field to a hexadecimal string."
      ≪ endl;
      unlock_cerr_mutex();
      ret_val = 1;
      goto UNLOCK_TABLES;
    } /* if (status ≠ 0) */
    else if (temp_str.length() ≡ 0) {
      lock_cerr_mutex();
      cerr ≪ "ERROR! In 'Handle_Value_Type::write_to_database':"
      ≪ endl ≪
        "'hexl_encode' succeeded, returning 0, but 'temp_str.length()' == 0."
      ≪ endl ≪
        "Failed to encode contents of 'data' field to a hexadecimal string."
      ≪ endl;
      unlock_cerr_mutex();
      ret_val = 1;
      goto UNLOCK_TABLES;
    } /* else if (temp_str.length() ≡ 0) */
  #if DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr ≪ "In 'Handle_Value_Type::write_to_database':"
      ≪ endl ≪
        "'hexl_encode' succeeded, returning 0."
      ≪ endl ≪
        "'temp_str' == " ≪ temp_str ≪ endl ≪
        "Encoded contents of 'data' field to a hexadecimal string successfully."
      ≪ endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
  #endif /* DEBUG_COMPILE */
  sql_strm ≪ "X'" ≪ temp_str ≪ ", ";
  } /* else */
  sql_strm ≪ ttl_type ≪ ", " ≪ ttl ≪ ", " ≪ timestamp ≪ ", ";
  if (refs ≡ 0) sql_strm ≪ "NULL";
  else /* refs ≠ 0 */
  {
    temp_str = "";

```

```

status = hexl_encode(refs, refs_length, temp_str);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Handle_Value_Type::write_to_database':"
        << endl <<
        "'hexl_encode' failed, returning " << status << "."
        << endl <<
        "Failed to encode contents of 'refs' field to a hexadecimal string." << endl;
    unlock_cerr_mutex();
    ret_val = 1;
    goto UNLOCK_TABLES;
} /* if (status ≠ 0) */
else if (temp_str.length() ≡ 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Handle_Value_Type::write_to_database':"
        << endl <<
        "'hexl_encode' succeeded, returning 0, but 'temp_str.length()' == 0."
        << endl <<
        "Failed to encode contents of 'refs' field to a hexadecimal string." << endl;
    unlock_cerr_mutex();
    ret_val = 1;
    goto UNLOCK_TABLES;
} /* else if (temp_str.length() ≡ 0) */
#if DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'Handle_Value_Type::write_to_database':"
            << endl <<
            "'hexl_encode' succeeded, returning 0."
            << endl <<
            "'temp_str' == "
            << temp_str << endl <<
            "Encoded contents of 'refs' field to a hexadecimal string successfully." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    sql_strm << "X'" << temp_str << "',";
} /* else (refs ≠ 0) */
sql_strm << admin_read << ","
    << admin_write << ","
    << pub_read << ","
    << pub_write << ","
    << handle_id << ","
    << handle_value_id << ","
    << created_by_user_id << ","
    << irods_object_id << ","
    << marked_for_deletion << ",";
if (created ≡ 0) sql_strm << "0,";
else sql_strm << "from_unixtime(" << created << "),";
if (last_modified ≡ 0) sql_strm << "0,";
else sql_strm << "from_unixtime(" << last_modified << "),";
if (delete_from_database_timestamp ≡ 0) sql_strm << "0";
else sql_strm << "from_unixtime(" << delete_from_database_timestamp << ")";
    sql_strm << ")";
#if DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'Handle_Value_Type::write_to_database':"
            << sql_strm.str() << endl <<
            sql_strm.str() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

877.

```

⟨Handle_Value_Type::write_to_database definition 874⟩ +≡
status = submit_mysql_query(sql_strm.str(), result, mysql_ptr, 0, 0, &affected_rows);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ "ERROR! In 'Handle_Value_Type::write_to_database':"
    endl ≪ "'submit_mysql_query' failed, returning"
    status ≪ ":" ≪ endl ≪ mysql_error(mysql_ptr) ≪ endl ≪
    "Failed to insert row into 'gwirdcli.handles' database table." ≪ endl;
    unlock_cerr_mutex();
    if (result) {
        mysql_free_result(result);
        result = 0;
    }
    ret_val = 1;
    goto UNLOCK_TABLES;
} /* if (status ≠ 0) */

```

878.

```

⟨Handle_Value_Type::write_to_database definition 874⟩ +≡
#if DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr ≪ "In 'Handle_Value_Type::write_to_database':"
            endl ≪ "'submit_mysql_query' succeeded, returning 0."
            endl ≪
            "Inserted row into 'gwirdcli.handles' database table successfully."
            endl ≪ "'affected_rows' =="
            affected_rows ≪ endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    mysql_free_result(result);
    result = 0;

```

879.

```

⟨Handle_Value_Type::write_to_database definition 874⟩ +≡
UNLOCK_TABLES: status = submit_mysql_query("unlock_tables", result, mysql_ptr);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Handle_Value_Type::write_to_database':"
    << endl <<
    "'submit_mysql_query' failed, returning" << status << ":" << endl <<
    mysql_error(mysql_ptr) << endl << "Failed to lock table 'gwirdcli.handles'." << endl <<
    "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
if (result) {
    mysql_free_result(result);
    result = 0;
}
return 1;
} /* if (status ≠ 0) */
#if DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'Handle_Value_Type::write_to_database':"
        << endl <<
        "'submit_mysql_query' succeeded, returning 0." << endl << mysql_error(mysql_ptr) <<
        endl << "Locked table 'gwirdcli.handles' successfully." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

880.

```

⟨Handle_Value_Type::write_to_database definition 874⟩ +≡
if (ret_val ≠ 0) {
    lock_cerr_mutex();
    cerr << "Exiting 'Handle_Value_Type::write_to_database' unsuccessful\
        ly with return value" << ret_val << "." << endl;
    unlock_cerr_mutex();
    return ret_val;
} /* if (ret_val ≠ 0) */

```

881.

```

⟨Handle_Value_Type::write_to_database definition 874⟩ +≡
#if DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Exiting 'Handle_Value_Type::write_to_database' successfully\
            with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; } /* End of Handle_Value_Type::write_to_database definition */

```

882. struct Handle_Value_Triple declaration. [LDF 2013.01.14.]

Log

[LDF 2013.01.14.] Added this **struct** declaration.

[LDF 2013.06.16.] Added definitions of default constructor, assignment operator, *clear* and *show* within class declaration.

```
⟨ Declare struct Handle_Value_Triple 882 ⟩ ≡
struct Handle_Value_Triple {
    int idx;
    string type;
    string data_str;
    Handle_Value_Triple(void)
    {
        idx = 0;
    }
    Handle_Value_Triple(int iidx, string ttype, string ddata_str = "")
    : idx(iidx), type(ttype), data_str(ddata_str) {}
    const Handle_Value_Triple &operator=(const Handle_Value_Triple &hvt)
    {
        idx = hvt.idx;
        type = hvt.type;
        data_str = hvt.data_str;
        return hvt;
    }
    void clear(void)
    {
        idx = 0;
        type = data_str = "";
        return;
    }
    int show(string s = "")
    {
        if (s.empty()) s = "Handle_Value_Triple:";
        cerr << s << endl << "idx== " << idx << endl << "type== " << type << endl << "data== " <<
            data_str << endl;
        return 0;
    }
};
```

This code is used in section 884.

883. Putting Handle_Value_Type together. [LDF 2012.10.12.]

```

⟨ Include files 3 ⟩
using namespace std;
using namespace gwrdifpk;
class Irods_Object_Type;
class Response_Type;
typedef void *yysscan_t;
class Scan_Parse_Parameter_Type;
class Handle_Type;

⟨ Initialize static Handle_Value_Type data members 753 ⟩
⟨ Handle_Value_Type constructor definitions 755 ⟩
⟨ Handle_Value_Type destructor definition 761 ⟩
⟨ Handle_Value_Type::operator= definition 759 ⟩
⟨ Handle_Value_Type::set definitions 765 ⟩
⟨ Handle_Value_Type::delete_handle_value definition 797 ⟩
⟨ Handle_Value_Type::unmark_handle_value_for_deletion definition 831 ⟩
⟨ Handle_Value_Type::extract definition 856 ⟩
⟨ Handle_Value_Type::clear definition 870 ⟩
⟨ Handle_Value_Type::show definition 872 ⟩
⟨ Handle_Value_Type::initialize_maps definition 763 ⟩
⟨ Handle_Value_Type::write_to_database definition 874 ⟩

```

884. This is what's written to the header file hndlvltp.h. [LDF 2008.12.05.]

```

⟨ hndlvltp.h 884 ⟩ ≡
#ifndef HNDLVLTP_H
#define HNDLVLTP_H 1
class Irods_Object_Type;
class Scan_Parse_Parameter_Type;
class Response_Type;
typedef void *yysscan_t;
class Handle_Type;

⟨ Declare struct Handle_Value_Triple 882 ⟩
⟨ Declare class Handle_Value_Type 752 ⟩
#endif

```

885. Handle_Type (hndltype.web).

Log

[LDF 2012.10.12.] Added this file.

886. Include files.

```
<Include files 3> +≡
#ifndef _XOPEN_SOURCE
#define _XOPEN_SOURCE
#endif
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <unistd.h>
#include <limits.h>
#if 0
#include <sys/stat.h>
#include <sys/time.h>
#endif
#include <time.h>
#include <sys/types.h>
#if 0
#include <dirent.h>
#endif
#include <string.h>
#if 0
#include <pwd.h>
#endif
#include <errno.h>
#include <pthread.h>
#include <fstream>
#include <iomanip>
#include <ios>
#include <iostream>
#include <string>
#include <sstream>
#include <deque>
#include <map>
#include <set>
#include <vector>
#include <gcrypt.h> /* for gcry-control */
#include <gnutls/gnutls.h>
#include <gnutls/x509.h>
#include <mysql.h>
#if HAVE_CONFIG_H
#include <config.h>
#endif
#undef NAME_LEN
#undef LOCAL_HOST
#include "glblcnst.h++"
#include "glblvrb.h++"
#include "excptntp.h++"
#include "rspercds.h++"
#include "utilfncts.h++"
#include "parser.h++"
#include "scanner.h++"
#include "hndlvltp.h++"
```

```
#include "rspnstp.h++"
#include "irdsavtp.h++"
#include "irdsobtp.h++"
```

887. class Handle_Type. [LDF 2013.02.27.]

Log

[LDF 2013.02.27.] Added this class declaration. I have renamed the previous **class Handle_Type** to **Handle_Value_Type** (see above).

[LDF 2013.08.12.] Added **friend** declaration for **Irods_Object_Type::mark_for_deletion**.
 [LDF 2013.08.14.] Added **friend** declaration for **Irods_Object_Type::delete_from_gwirdsif_db**.
 [LDF 2013.08.15.] Added **friend** declaration for **Irods_Object_Type::add_handle_value**.
 [LDF 2013.08.22.] Added **bool marked_for_deletion**.

```
⟨ Declare class Handle_Type 887 ⟩ ≡
class Handle_Type {
    friend class Scan_Parse_Parameter_Type;
    friend int main(int argc, char *argv[]);
    friend int generate_pids(MYSQL *mysql_ptr, string prefix_str, string &pid_str, vector<string>
        *pid_vector_ptr, unsigned int number_of_pids, vector<unsigned long int> *handle_id_vector_ptr,
        vector<unsigned long int> *handle_value_id_vector_ptr, bool standalone_hs, string
        institute_str, string suffix_str, vector<Handle_Type> *handle_vector, string fifo_pathname, long
        int user_id, string username);
    friend int exchange_data_with_client(Scan_Parse_Parameter_Type &param);
    friend int Irods_Object_Type::mark_for_deletion(vector<Irods_Object_Type>
        &irods_object_vector, MYSQL *&mysql_ptr, Response_Type &response, int user_id, string
        irods_env_filename, time_t &save_delay, string thread_str, bool wake_purge_thread);
    friend int Irods_Object_Type::delete_from_archive(MYSQL *&mysql_ptr, string thread_str);
    friend int Irods_Object_Type::delete_from_gwirdsif_db(MYSQL *&mysql_ptr, string thread_str);
    friend int Irods_Object_Type::add_handle_value(Handle_Type &handle, MYSQL*&mysql_ptr, string
        old_type_str, string path, unsigned int index, string new_type_str, string data_str, string
        thread_str);
    string handle;
    unsigned long int handle_id;
    map<unsigned long int, Handle_Value_Type> handle_value_map;
public: ⟨ class Handle_Type function declarations 889 ⟩
};
```

This code is used in sections 992 and 993.

888. class Handle_Type functions.

889. Default constructor. [LDF 2013.02.27.]

Log

[LDF 2013.02.27.] Added this function.

```
⟨ class Handle_Type function declarations 889 ⟩ ≡
Handle_Type(void);
```

See also sections 891, 893, 895, 897, 899, 901, 926, 943, 948, 951, 959, 965, 977, and 981.

This code is used in section 887.

890.

```
<Handle_Type constructor definitions 890> ≡
Handle_Type::Handle_Type(void)
{
    handle_id = 0;
    return;
}
```

See also section 892.

This code is used in section 992.

891. Copy constructor. [LDF 2013.02.27.]

Log

[LDF 2013.02.27.] Added this function.

```
<class Handle_Type function declarations 889> +≡
Handle_Type(const Handle_Type &h);
```

892.

```
<Handle_Type constructor definitions 890> +≡
Handle_Type::Handle_Type(const Handle_Type &h)
{
    bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
#if DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Entering 'Handle_Type' copy constructor." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    operator=(h);
#if DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Exiting 'Handle_Type' copy constructor." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return;
}
```

893. Assignment operator. [LDF 2013.02.27.]

Log

[LDF 2013.02.27.] Added this function.

```
<class Handle_Type function declarations 889> +≡
void operator=(const Handle_Type &h);
```

894.

```

⟨ Handle_Type::operator= definition 894 ⟩ ≡
void Handle_Type::operator=(const Handle_Type &h)
{
    bool DEBUG = false;      /* true */
    set_debug_level(DEBUG, 0, 0);
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Entering 'Handle_Type' assignment operator." << endl;
        unlock_cerr_mutex();
    }      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */
    handle = h.handle;
    handle_id = h.handle_id;
    unsigned long int idx;
    Handle_Value_Type handle_value;
    for (map<unsigned long int,
              Handle_Value_Type>::const_iterator iter = h.handle_value_map.begin();
         iter != h.handle_value_map.end(); ++iter) {
        handle_value.clear();
        idx = iter->first;
        handle_value = iter->second;
        handle_value_map.insert(make_pair(idx, handle_value));
    }
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Exiting 'Handle_Type' assignment operator." << endl;
        unlock_cerr_mutex();
    }      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */
    return;
}      /* End of Handle_Type::operator= definition */

```

This code is used in section 992.

895. Clear. [LDF 2013.02.27.]

Log

[LDF 2013.02.27.] Added this function.

```

⟨ class Handle_Type function declarations 889 ⟩ +≡
void clear(void);

```

896.

```
<Handle_Type::clear definition 896> ≡
void Handle_Type::clear(void)
{
    handle = "";
    handle_id = 0;
    handle_value_map.clear();
}
```

This code is used in section 992.

897. Show. [LDF 2013.02.27.]

Log

[LDF 2013.02.27.] Added this function.
 [LDF 2013.03.07.] Added optional argument **stringstream** *strm = 0.

```
<class Handle_Type function declarations 889> +≡
void show(string s = "Handle_Type:", stringstream *strm = 0) const;
```

898.

```
<Handle_Type::show definition 898> ≡
void Handle_Type::show(string s, stringstream *strm) const
{
    stringstream temp_strm;
    stringstream temp_strm_1;

    temp_strm << s << endl << "handle==oooooooooooooo" << handle << endl <<
    "handle_id==oooooooooooooo" << handle_id << endl << "handle_value_map.size() == " <<
    handle_value_map.size() << endl;

    int i = 0;
    for (map<unsigned long int, Handle_Value_Type>::const_iterator iter = handle_value_map.begin();
         iter != handle_value_map.end(); ++iter) {
        temp_strm_1.str("");
        temp_strm_1 << "Handle_Value_Type" << i++ << ":";
        iter->second.show(temp_strm_1.str(), &temp_strm);
    }
    if (*strm) *strm << temp_strm.str();
    else cerr << temp_strm.str();
    return;
} /* End of Handle_Type::show definition */
```

This code is used in section 992.

899. Add handle value (*add_value*) [LDF 2013.02.27.]

Log

[LDF 2013.02.27.] Added this function.
 [LDF 2013.07.03.] Added optional argument **Handle_Value_Triple** *return_hvt = 0.

```
<class Handle_Type function declarations 889> +≡
int add_value(MYSQL *mysql_ptr, int iidx, string ttype, string ddata_str, int user_id,
              Handle_Value_Triple *return_hvt = 0);
```

900.

```

⟨Handle_Type::add_value definition 900⟩ ≡
int Handle_Type::add_value(MYSQL *mysql_ptr, int iidx, string ttype, string ddata_str, int
                           user_id, Handle_Value_Triple *return_hvt)
{
    bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Entering 'Handle_Type::add_value'." << endl;
        show(*this);
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    Handle_Value_Triple hvt;
    vector<Handle_Value_Triple> hvt_vector;
    vector<Handle_Value_Triple> return_hvt_vector;
    hvt.idx = iidx;
    hvt.type = ttype;
    hvt.data_str = ddata_str;
    hvt_vector.push_back(hvt);
    status = add_values(mysql_ptr, hvt_vector, user_id, &return_hvt_vector);
    if (status ≠ 0) {
        lock_cerr_mutex();
        cerr << "ERROR! In 'Handle_Type::add_value': "
             << "Handle_Type::add_values' failed, " << "returning" << status << "."
             << endl << "Exiting function unsuccessfully with return value" << status << "."
             << endl;
        unlock_cerr_mutex();
    } /* if (status ≠ 0) */
    else /* status ≡ 0 */
    {
        if (return_hvt) {
            *return_hvt = return_hvt_vector.back();
        } /* if (return_hvt) */
#endif DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "In 'Handle_Type::add_value': 'Handle_Type::add_values' succeeded, "
                 << "returning 0." << endl;
            if (return_hvt) return_hvt->show(*return_hvt);
            else cerr << "'return_hvt' is NULL." << endl;
            cerr << "Exiting function successfully with return value 0." << endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    } /* else (status ≡ 0) */
    return status;
} /* End of Handle_Type::add_value definition */

```

This code is used in section 992.

901. Add handle values (*add_values*) [LDF 2013.02.27.]

Log

[LDF 2013.02.27.] Added this function.

[LDF 2013.02.28.] Revised this function. It now replaces **Handle_Value_Type**::*add_values*. It contains slightly modified code from the latter function.

[LDF 2013.07.03.] Added optional argument **vector**<**Handle_Value_Triple**> **return_hvt_vector* = 0.

```
< class Handle_Type function declarations 889 > +≡
int add_values(MYSQL *mysql_ptr, vector<Handle_Value_Triple> hvt_vector, int user_id,
               vector<Handle_Value_Triple> *return_hvt_vector = 0);
```

902.

```
< Handle_Type::add_values definition 902 > ≡
int Handle_Type::add_values(MYSQL *mysql_ptr, vector<Handle_Value_Triple> hvt_vector, int
                           user_id, vector<Handle_Value_Triple> *return_hvt_vector){ bool DEBUG = false;
/* true */
set_debug_level(DEBUG, 0, 0);
int status = 0;
int ret_val = 0;
MYSQL_RES *result = 0;
MYSQL_ROW curr_row;
unsigned int row_ctr;
unsigned int field_ctr;
long int affected_rows;
Handle_Value_Type curr_handle_value;
unsigned long curr_handle_value_id;
stringstream sql_strm;
map<int, string> curr_idx_type_map;
map<int, string>::iterator map_iter;
map<int, string>::iterator prev_map_iter;
long int curr_idx;
string handle_database = (standalone_handle) ? "handlesystem_standalone" : "handlesystem";
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "Entering 'Handle_Type::add_values'." << endl << "'hvt_vector.size()' == " <<
        hvt_vector.size() << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

See also sections 903, 904, 905, 906, 907, 908, 909, 910, 911, 912, 913, 914, 915, 916, 917, 918, 919, 920, 921, 922, 923, 924, and 925.

This code is used in section 992.

903.**Log**

[LDF 2013.01.30.] Added this section.

```
<Handle_Type::add_values definition 902> +≡
    status = mysql_select_db(mysql_ptr, handle_database.c_str());
    if (status ≡ 0) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'Handle_Type::add_values': mysql_select_db succeeded." << endl <<
            "Selected database " << handle_database << " successfully." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (status ≡ 0) */
else /* status ≠ 0 */
{
    lock_cerr_mutex();
    cerr << "In 'Handle_Type::add_values': mysql_select_db failed, returning" <<
        status << ":" << endl << mysql_error(mysql_ptr) << endl <<
        "Failed to select database " << handle_database << "." << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    return 1;
} /* else (status ≠ 0) */
```

904.

```

⟨ Handle_Type::add_values definition 902 ⟩ +≡
status = submit_mysql_query("lock_tables_handles_write", result, mysql_ptr, 0, 0, 0);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Handle_Type::add_values':"
        << endl <<
        "'submit_mysql_query' failed, returning"
        << status <<
        "."
        << endl << "Failed to lock 'handles' table."
        << endl <<
        "Exiting function unsuccessfully with return value 1."
        << endl;
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    return 1;
} /* if (status ≠ 0) */
#if DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'Handle_Type::add_values':"
            << endl <<
            "'submit_mysql_query' succeeded, returning 0."
            << endl <<
            "Locked 'handles' table successfully."
            << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
mysql_free_result(result);
result = 0;

```

905.

```

⟨ Handle_Type::add_values definition 902 ⟩ +≡
if (hvt_vector.size() ≡ 0) {
    lock_cerr_mutex();
    cerr << "WARNING! In 'Handle_Type::add_values': 'hvt_vector.size()' == 0."
        << endl <<
        "No values to add. Will try to unlock"
        << handle_database << ".handles"
        << "database table and exit function unsuccessfully with return value 1."
        << endl;
    unlock_cerr_mutex();
    ret_val = 1;
    goto ADD_VALUES_UNLOCK_TABLES;
} /* if (hvt_vector.size() ≡ 0) */

```

906. Check whether handle exists. [LDF 2013.01.31.]

Log

[LDF 2013.01.31.] Added this section.

```

⟨Handle_Type::add_values definition 902⟩ +≡
  sql_strm.str("");
  sql_strm << "select handle_id from handles where handle_id = " << handle_id << " limit 1";
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'add_values': " << sql_strm.str() << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  status = submit_mysql_query(sql_strm.str().c_str(), result, mysql_ptr, &row_ctr, &field_ctr);
  if (status != 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Handle_Type::add_values':"
    << "'submit_mysql_query' failed, returning " << status << "."
    << "Failed to retrieve 'handle_id' from 'handles' database table."
    << "Will try to unlock tables and exit function unsuccessfully with return value " << endl <<
    endl;
    unlock_cerr_mutex();
    if (result) {
      mysql_free_result(result);
      result = 0;
    }
    ret_val = 1;
    goto ADD_VALUES_UNLOCK_TABLES;
  } /* if (status != 0) */
#endif DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "In 'Handle_Type::add_values':"
      << "'submit_mysql_query' succeeded, returning 0."
      << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

907.

```
<Handle_Type::add_values definition 902> +≡
  if (row_ctr == 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Handle_Type::add_values':"
    << endl <<
    "'submit_mysql_query' returned 0 rows." << endl <<
    "Failed to retrieve 'handle_id' from 'handles' database table."
    << endl <<
    "Will try to unlock tables and exit function unsuccessfully with return value 1."
    << endl;
    unlock_cerr_mutex();
  if (result) {
    mysql_free_result(result);
    result = 0;
  }
  ret_val = 1;
  goto ADD_VALUES_UNLOCK_TABLES;
} /* if (row_ctr == 0) */
```

908. We don't need to fetch the value, since we already know what it is; we just need to know that the handle really exists. [LDF 2013.01.31.]

```
<Handle_Type::add_values definition 902> +≡
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "In 'Handle_Type::add_values':"
      << endl <<
      "'submit_mysql_query' returned"
      << row_ctr << " rows." << endl <<
      "Retrieved 'handle_id' from 'handles' database successfully."
      << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  mysql_free_result(result);
  result = 0;
```

909. Get value for *handle_value_id*. [LDF 2013.01.31.]

```

⟨Handle_Type::add_values definition 902⟩ +≡
  sql_strm.str("");
  sql_strm << "select_handle_value_id_from_handles_order_by_handle_value_id_desc_limit_1";
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'add_values': " << sql_strm.str() << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  status = submit_mysql_query(sql_strm.str().c_str(), result, mysql_ptr, &row_ctr, &field_ctr);
  if (status != 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Handle_Type::add_values':" << endl <<
      "'submit_mysql_query' failed, returning " << status << "." << endl <<
      "Failed to retrieve highest value of 'handle_value_id' from "
      "'handles' database table." << endl << "Will try to unlock tables and exit function"
      "unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
  if (result) {
    mysql_free_result(result);
    result = 0;
  }
  ret_val = 1;
  goto ADD_VALUES_UNLOCK_TABLES;
} /* if (status != 0) */
#ifndef DEBUG_COMPILE
else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'Handle_Type::add_values':" << endl <<
      "'submit_mysql_query' succeeded, returning 0." << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

910.

```
<Handle_Type::add_values definition 902> +≡
  if (row_ctr ≡ 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Handle_Type::add_values':"
    << endl <<
      "'submit_mysql_query' returned 0 rows." << endl <<
      "Failed to retrieve \
      highest value of 'handle_value_id' from 'handles'" << "database_table."
      "Will try to unlock tables and exit function unsuccessfully with return value 1." <<
      endl;
    unlock_cerr_mutex();
  if (result) {
    mysql_free_result(result);
    result = 0;
  }
  ret_val = 1;
  goto ADD_VALUES_UNLOCK_TABLES;
} /* if (row_ctr ≡ 0) */
```

911.

```
<Handle_Type::add_values definition 902> +≡
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "In 'Handle_Type::add_values':"
      << endl <<
        "'submit_mysql_query' returned"
        << row_ctr << " rows." << endl <<
        "Retrieved highest value of 'handle_value_id' from 'handles'"
        << "database_table successfully." << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

912.

```

⟨Handle_Type::add_values definition 902⟩ +≡
  if ((curr_row = mysql_fetch_row(result)) ≡ 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Handle_Type::add_values:' << endl <<
      'mysql_fetch_row' failed:" << endl << mysql_error(mysql_ptr) << endl <<
      "Will try to unlock tables and exit function unsuccessfully with return value 1." <<
      endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    result = 0;
    ret_val = 1;
    goto ADD_VALUES_UNLOCK_TABLES;
} /* if (curr_row = mysql_fetch_row(result) ≡ 0) */
else if (DEBUG) {
  lock_cerr_mutex();
  cerr << "In 'Handle_Type::add_values:' mysql_fetch_row succeeded." << endl <<
    "'curr_row[0]' == " << curr_row[0] << endl;
  unlock_cerr_mutex();
} /* else if (DEBUG) */
errno = 0;
curr_handle_value_id = strtoul(curr_row[0], 0, 10);
if (curr_handle_value_id ≡ ULONG_MAX) {
  lock_cerr_mutex();
  cerr << "ERROR! In 'Handle_Type::add_values:' << endl <<
    "'strtoul' failed, returning ULONG_MAX:' << endl << strerror(errno) <<
    endl << "Failed to set 'curr_handle_value_id'." << endl <<
    "Will try to unlock tables and exit function unsuccessfully with return value 1." <<
    endl;
  unlock_cerr_mutex();
  mysql_free_result(result);
  result = 0;
  ret_val = 1;
  goto ADD_VALUES_UNLOCK_TABLES;
} /* if (curr_handle_value_id ≡ ULONG_MAX) */
#endif /* DEBUG_COMPILE
else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'Handle_Type::add_values:' << endl <<
      "'strtoul' succeeded: 'curr_handle_value_id' == " << curr_handle_value_id << "."
      << endl << "Will increment." << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
++curr_handle_value_id;
mysql_free_result(result);
result = 0;
sql_strm.str("");

```

913. Fill *curr_idx_type_map*. [LDF 2013.02.05.]

Log

[LDF 2013.02.05.] Added this section.

```

⟨Handle_Type::add_values definition 902⟩ +≡
  sql_strm << "select_idx, type from handles where handle_id = " << handle_id;
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "sql_strm.str() == " << sql_strm.str() << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  status = submit_mysql_query(sql_strm.str().c_str(), result, mysql_ptr, &row_ctr, &field_ctr);
  if (status != 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Handle_Type::add_values':"
    << endl <<
      "'submit_mysql_query' failed, returning " << status << "."
    << endl << "Failed to retrieve 'idx' and 'type' values from "
    << "'handles' database table for 'handle_id' == "
    << handle_id << "."
    << endl <<
      "Will try to unlock tables and exit function unsuccessfully with return value 1."
    << endl;
    unlock_cerr_mutex();
  if (result) {
    mysql_free_result(result);
    result = 0;
  }
  ret_val = 1;
  goto ADD_VALUES_UNLOCK_TABLES;
} /* if (status != 0) */
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "In 'Handle_Type::add_values':"
      << endl <<
        "'submit_mysql_query' succeeded, returning 0."
      << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

914. There should always be at least one handle value, i.e., one with type = HS_ADMIN. [LDF 2013.02.05.]

```
<Handle_Type::add_values definition 902> +≡
  if (row_ctr ≡ 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Handle_Type::add_values':" << endl <<
      "'submit_mysql_query' returned 0 rows." << endl <<
      "Failed to retrieve 'idx' and 'type' values from 'handles'" <<
      "'handles' database table for 'handle_id' == " << handle_id << "." << endl <<
      "Will try to unlock tables and exit function unsuccessfully with return value 1." <<
      endl;
    unlock_cerr_mutex();
    if (result) {
      mysql_free_result(result);
      result = 0;
    }
    ret_val = 1;
    goto ADD_VALUES_UNLOCK_TABLES;
  } /* if (row_ctr ≡ 0) */
```

915.

```
<Handle_Type::add_values definition 902> +≡
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "In 'Handle_Type::add_values':" << endl << "'submit_mysql_query' returned " <<
        row_ctr << " rows." << endl << "Retrieved 'idx' and 'type' values from 'handles'" <<
        "'handles' database table for 'handle_id' == " << handle_id << " successfully." << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

916.

```

⟨ Handle_Type :: add_values definition 902 ⟩ +≡
for (int i = 0; i < row_ctr; ++i) {
  if ((curr_row = mysql_fetch_row(result)) == 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Handle_Type::add_values:'" << endl <<
      "'mysql_fetch_row' failed:" << endl << mysql_error(mysql_ptr) << endl <<
      "Will try to unlock tables and exit function unsuccessfully" <<
      "with return value 1." << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    result = 0;
    ret_val = 1;
    goto ADD_VALUES_UNLOCK_TABLES;
  } /* if (curr_row = mysql_fetch_row(result) == 0) */
else if (DEBUG) {
  lock_cerr_mutex();
  cerr << "In 'Handle_Type::add_values:' mysql_fetch_row succeeded." << endl <<
    "'curr_row[0]' == " << curr_row[0] << endl << "'curr_row[1]' == " << curr_row[1] << endl;
  unlock_cerr_mutex();
} /* else if (DEBUG) */
errno = 0;
curr_idx = strtol(curr_row[0], 0, 10);
if (curr_idx == LONG_MAX || curr_idx == LONG_MIN) {
  lock_cerr_mutex();
  cerr << "ERROR! In 'Handle_Type::add_values:'" << endl <<
    "'strtol' failed, returning" << curr_idx << " ";
  if (curr_idx == LONG_MAX) cerr << "(LONG_MAX)." << endl;
  else cerr << "(LONG_MIN)." << endl;
  cerr << strerror(errno) << endl << "Will try to unlock tables and exit func\\
tion unsuccessfully" << "with return value 1." << endl;
  unlock_cerr_mutex();
  mysql_free_result(result);
  result = 0;
  ret_val = 1;
  goto ADD_VALUES_UNLOCK_TABLES;
} /* if */
#endif DEBUG_COMPILE
if (DEBUG) {
  lock_cerr_mutex();
  cerr << "'strtol' succeeded, returning" << curr_idx << " ('curr_idx')." << endl;
  unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
curr_idx_type_map[curr_idx] = curr_row[1];
} /* for */
mysql_free_result(result);
result = 0;
sql_strm.str("");
#endif DEBUG_COMPILE
if (DEBUG) {
  lock_cerr_mutex();

```

```

cerr << "{'curr_idx_type_map':}" << endl;
for (map<int,
      string>::const_iterator iter = curr_idx_type_map.begin( ); iter != curr_idx_type_map.end( );
      ++iter) {
    cerr << iter->first << ", " << iter->second << endl;
}
/* for */
cerr << endl;
unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

917.

```

⟨Handle_Type::add_values definition 902⟩ +≡
#if DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "Showing 'hvt_vector': " << endl;
    for (vector<Handle_Value_Triple>::const_iterator iter = hvt_vector.begin( ); iter != hvt_vector.end( );
          ++iter) {
      cerr << "'iter->idx' " << iter->idx << endl << "'iter->type' " << iter->type <<
          endl << "'iter->data_str' " << iter->data_str << endl;
    }
    /* for */
    cerr << endl;
    unlock_cerr_mutex();
  }
  /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

918.

Log

[LDF 2013.01.30.] Added this section.

```

⟨Handle_Type::add_values definition 902⟩ +≡
  for (vector<Handle_Value_Triple>::iterator iter = hvt_vector.begin( ); iter != hvt_vector.end( );
        ++iter) { sql strm.str(" ");

```

919. Check whether the index value has already been used. If it hasn't, we just use it. If it has, we check what range it belongs to, assuming that *iter->idx* and *iter->type* have been chosen to correspond with the pairs of values in **Handle_Value_Type**::*idx_type_map*. If a value within the range hasn't already been used, we use it (and push a pair onto *curr_idx_type_map*). Otherwise, we find the next unused value \geq **Handle_Value_Type**::OTHER_HANDLE_VALUE_TYPE. [LDF 2013.02.05.]

Log

[LDF 2013.03.07.] Added code for finding an unused value for *iter->idx*, as described above.

```

⟨Handle_Type::add_values definition 902⟩ +≡
    map_iter = curr_idx_type_map.find(iter->idx); if (map_iter != curr_idx_type_map.end())
        /* index value already used. */
    {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << 'map_iter' << !curr_idx_type_map.end() . " " << "Index " << iter->idx << " "
            << iter->idx << " has already " << "been used." << endl <<
            "Will search for range for another value to use." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif      /* DEBUG_COMPILE */
    map<int, string>::iterator range_iter_0;
    map<int, string>::iterator range_iter_1;
    bool found = false;
    for (range_iter_0 = Handle_Value_Type::idx_type_map.begin();
        range_iter_0 != Handle_Value_Type::idx_type_map.end(); ++range_iter_0)
    {
        range_iter_1 = range_iter_0;
        ++range_iter_1;
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "range_iter_0->first == " << range_iter_0->first << endl <<
                "range_iter_0->second == " << range_iter_0->second << endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */
        if (range_iter_1 == Handle_Value_Type::idx_type_map.end()) {
#ifndef DEBUG_COMPILE
            if (DEBUG) {
                lock_cerr_mutex();
                cerr << "Reached end of Handle_Value_Type::idx_type_map." << endl << "Breaking." <<
                    endl;
                unlock_cerr_mutex();
            } /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */
            break;
        } /* if */
#ifndef DEBUG_COMPILE
    else

```

```

if (DEBUG) {
    lock_cerr_mutex();
    cerr << "range_iter_1->first=="
        << range_iter_1-first << endl <<
    "range_iter_1->second=="
        << range_iter_1-second << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
if (iter->idx ≥ range_iter_0-first ∧ iter->idx < range_iter_1-first) {
#if DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Found range: "
            << range_iter_0-first << "≤ "
            << iter->idx << "≤ "
            << range_iter_1-first << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    ++iter->idx;
    for ( ; iter->idx < range_iter_1-first; ++iter->idx) {
        map_iter = curr_idx_type_map.find(iter->idx);
        if (map_iter ≡ curr_idx_type_map.end()) {
#if DEBUG_COMPILE
            if (DEBUG) {
                lock_cerr_mutex();
                cerr << "Found unused value within range: "
                    << iter->idx << endl << "Breaking."
                    << endl;
                unlock_cerr_mutex();
            } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
            found = true;
            break;
        } /* if */
    } /* for */
    if (!found) {
#if DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "Index value not found. Will use value >= "
                << Handle_Value_Type::idx_type_map[Handle_Value_Type::OTHER_HANDLE_VALUE_TYPE_INDEX] << endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        iter->idx = curr_idx_type_map.rbegin()-first;
        ++iter->idx;
        if (iter->idx < Handle_Value_Type::OTHER_HANDLE_VALUE_TYPE_INDEX)
            iter->idx = Handle_Value_Type::OTHER_HANDLE_VALUE_TYPE_INDEX;
    } /* if (!found) */
}

```

920.

```
<Handle_Type::add_values definition 902> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "iter->idx=="
        << iter->idx << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    } /* if (map_iter != curr_idx_type_map.end()) */
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) /* index value not already used */
    {
        lock_cerr_mutex();
        cerr << 'map_iter' == curr_idx_type_map.end() . " "
        << "Index 'iter->idx' == "
        << iter->idx << " hasn't already "
        << "been used." << endl <<
        "No need to search for range for another value to use." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

921. If *return_hvt_vector* is non-null, put *iter->idx* and *iter->type* onto **return_hvt_vector*. [LDF 2013.07.03.]

Log

[LDF 2013.07.03.] Added this section.

```

⟨Handle_Type::add_values definition 902⟩ +≡
  if (return_hvt_vector ≠ 0) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "In Handle_Type::add_values:" ≪ endl ≪
      "'return_hvt_vector' is non-null. Pushing 'Handle_Value_Triple'" ≪
      "onto the vector to which it points." ≪ endl ≪ "'iter->idx' == " ≪ iter->idx ≪
      endl ≪ "'iter->type' == " ≪ iter->type ≪ endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  return_hvt_vector->push_back(Handle_Value_Triple(iter->idx, iter->type, iter->data_str));
#endif DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    return_hvt_vector->back().show("*return_hvt_vector->back():");
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (return_hvt_vector ≠ 0) */
#ifndef DEBUG_COMPILE
else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "In Handle_Type::add_values:" ≪ endl ≪
      "'return_hvt_vector' is null. Not pushing a 'Handle_Value_Triple'" ≪
      "onto a vector." ≪ endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

922.

```

⟨Handle_Type::add_values definition 902⟩ +≡
sql_strm << "insert_into_handles_(handle, idx, type, "
<< "data, ttl_type, ttl, timestamp, refs, admin_read, admin_write, "
<< "pub_read, pub_write, handle_id, handle_value_id, created, "
<< "created_by_user_id)" << "values" << "(" << handle << ", "
<< iter->idx << ", " << iter->type << ", " << "" << iter->data_str << ", "
<< "0, 86400, " << time(0) << ", "
<< ", 1, 1, 1, 0, " << handle_id << ", "
<< curr_handle_value_id << ", now(), "
<< user_id << ")";
#endif /* DEBUG_COMPILE */
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "sql_strm.str() == " << sql_strm.str() << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
status = submit_mysql_query(sql_strm.str().c_str(), result, mysql_ptr, 0, 0, &affected_rows);
if (status != 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Handle_Type::add_values':"
        << endl <<
        "'submit_mysql_query' failed, returning " << status << "."
        << endl <<
        "Failed to insert row into " << handle_database << ".handles"
        << endl <<
        "database_table." << endl <<
        "Will try to unlock " << handle_database << ".handles"
        << endl <<
        "database_table_and_exit_function unsuccessfully with return value 1."
        << endl;
    unlock_cerr_mutex();
    ret_val = 1;
    if (result) {
        mysql_free_result(result);
        result = 0;
    }
    goto ADD_VALUES_UNLOCK_TABLES;
} /* if (status != 0) */
#endif /* DEBUG_COMPILE */
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'Handle_Type::add_values':"
        << endl <<
        "'submit_mysql_query' succeeded, returning 0."
        << endl << "'affected_rows' == "
        << affected_rows << "."
        << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
mysql_free_result(result);
result = 0;

```

923.

Log

[LDF 2013.02.08.] Added this section.
 [LDF 2013.08.01.] BUG FIX: Now incrementing *curr_handle_value_id*.

```
<Handle_Type::add_values definition 902> +≡
curr_handle_value.clear();
curr_handle_value.set(handle, iter->idx, iter->type, const_cast<char*>(iter->data_str.c_str())),
    iter->data_str.size(), 0, 86400, time(0), 0, 0, true, true, false, user_id, handle_id,
    curr_handle_value_id, false, time(0), 0);
if (iter->type == "IRODS_OBJECT") curr_handle_value.filename = iter->data_str;
handle_value_map[iter->idx] = curr_handle_value;
++curr_handle_value_id; } /* for */
```

924.

Log

[LDF 2013.01.30.] Added this section.

```
<Handle_Type::add_values definition 902> +≡
ADD_VALUES_UNLOCK_TABLES:
if (result) {
    mysql_free_result(result);
    result = 0;
}
status = submit_mysql_query("unlock_tables", result, mysql_ptr, 0, 0, 0);
if (status != 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Handle_Type::add_values':"
        << endl <<
        "'submit_mysql_query' failed, returning" << status << "."
        << endl << "Failed to unlock 'handles' table."
        << endl <<
        "Will exit function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    if (result) {
        mysql_free_result(result);
        result = 0;
    }
    ret_val = 1;
} /* if (status != 0) */
#endif DEBUG_COMPILE
else
{
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'Handle_Type::add_values':"
            << endl <<
            "'submit_mysql_query' succeeded, returning 0."
            << endl <<
            "Unlocked 'handles' table successfully."
            << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
mysql_free_result(result);
result = 0;
```

925.

```

⟨ Handle_Type :: add_values definition 902 ⟩ +≡
  if (ret_val ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ "Exiting 'Handle_Type::add_values' unsuccessfully with return value "
    ret_val ≪ "." ≪ endl;
    unlock_cerr_mutex();
    return ret_val;
  } /* if (ret_val ≠ 0) */
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "Exiting 'Handle_Type::add_values' successfully with return value 0." ≪ endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  return 0; } /* End of Handle_Type::add_values definition */

```

926. Delete from database (*delete_from_database*). [LDF 2013.07.04.]

Log

[LDF 2013.07.04.] Added this function.

[LDF 2013.07.25.] Added **int user_id** argument.

```

⟨ class Handle_Type function declarations 889 ⟩ +≡
  int delete_from_database(MYSQL *mysql_ptr, int user_id, unsigned int options, unsigned long int
  delay_value = 0UL, int *handle_rows = 0, int thread_ctr = 0);

```

927.

```

⟨ Handle_Type :: delete_from_database definition 927 ⟩ ≡
int Handle_Type :: delete_from_database(MYSQL* mysql_ptr, int user_id, unsigned int options, unsigned
                                         long int delay_value, int *handle_rows, int thread_ctr){ bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    string thread_str;
    stringstream temp_strm;
    MYSQL_RES * result = 0;
    long int affected_rows;
    stringstream sql_strm;
    int status;
    int ret_val = 0;
    if (thread_ctr > 0) {
        temp_strm << "[Thread" << thread_ctr << "]";
        thread_str = temp_strm.str();
        temp_strm.str("");
    }
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering‘Handle_Type::delete_from_database’." << endl <<
            "'handle'" << handle << endl << "'options'" << hex << options << "(hex)" <<
            dec << endl;
        if (options & 1U) {
            cerr << "'IMMEDIATE'" << (0x1) << "'true'" << endl;
        }
        else {
            cerr << "'IMMEDIATE'" << (0x1) << "'false'" << endl;
        }
        cerr << "delay_value" << delay_value << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

See also sections 928, 929, 930, 931, 932, 933, 934, 935, 936, 937, 938, 939, 940, 941, and 942.

This code is used in section 992.

928. Error handling: "immediate" option set and *delay_value* > 0. [LDF 2013.08.26.]

Log

[LDF 2013.08.26.] Added this section.

```
<Handle_Type::delete_from_database definition 927> +≡
if (options & 1U & delay_value > 0) {
    lock_cerr_mutex();
    cerr << thread_str << "WARNING! In 'Handle_Type::delete_from_database':"
        << endl << "\'immediate\'_option_set_and_\'delay_value\'_==_"
        << endl << delay_value << "(>0)" << endl << "This shouldn't be possible." << endl <<
        "Unsetting_\'immediate\'_option_and_using_default_delay_value." << endl;
    unlock_cerr_mutex();
    options &= ~1UL;
    delay_value = 1;
} /* if (options & 1U & delay_value > 0) */
```

929.

Log

[LDF 2013.08.26.] Added this section.

```
<Handle_Type::delete_from_database definition 927> +≡
errno = 0;
time_t delay_time = time(0);
if (delay_time ≡ static_cast<time_t>(-1)) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Handle_Type::delete_from_database':"
        << endl << "'time'_failed,_returning_<'(time_t)_-1':"
        << endl << strerror(errno) << endl << "Failed_to_set_current_time."
        << endl << "Exiting_function_unsuccessfully_with_return_value_1."
        << endl;
    unlock_cerr_mutex();
    return 1;
} /* if (delay_time ≡ static_cast<time_t>(-1)) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Handle_Type::delete_from_database':"
            << endl << "'time'_succeeded._<'delay_time\'_==_"
            << delay_time << " _<convert_seconds(delay_time,"
            << true,false) << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

930. Error handling: *delay_value* too large. [LDF 2013.08.26.]

Log

[LDF 2013.08.26.] Added this section.

```

⟨Handle_Type::delete_from_database definition 927⟩ +≡
if (delay_value > 0) {
    if (delay_value ≡ ULONG_MAX ∨ ULONG_MAX - delay_time ≤ delay_value) {
        lock_cerr_mutex();
        cerr << thread_str << "WARNING! In 'Handle_Type::delete_from_database':"
            << endl << "'delay_value' == " << delay_value << " exceeds maximum value." << endl <<
            "Will use default delay value." << endl;
        unlock_cerr_mutex();
        delay_value = 1;
    } /* if (delay_value ≡ ULONG_MAX ∨ ULONG_MAX - delay_time ≤ delay_value) */
    else {
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_str << "In 'Handle_Type::delete_from_database':"
                << endl << "'delay_value' == " << delay_value << " doesn't exceed maximum value." << endl <<
                "Will set deletion time to " << convert_seconds(delay_value + delay_time, true) << endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        delay_time += delay_value;
    } /* else */
} /* if (delay_value > 0) */

```

931. If not marking for immediate deletion: Add handle value of *type* "HANDLE_MARKED_FOR_DELETION". [LDF 2013.07.25.]

Log

[LDF 2013.07.25.] Added this section.

```

⟨Handle_Type::delete_from_database definition 927⟩ +≡
if (¬(options & 1U)) /* Not immediate. */
{
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Handle_Type::delete_from_database':"
            << endl << "'!(options & 1U): Not marking 'Handle_Type' object for"
            << "immediate deletion." << endl << "Will call 'Handle_Type::add_value' to add"
            << "handle value of type 'HANDLE_MARKED_FOR_DELETION'." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

932.

```
<Handle_Type::delete_from_database definition 927> +≡
Handle_Value_Triple hvt;
status = add_value(mysql_ptr, Handle_Value_Type::HANDLE_MARKED_FOR_DELETION_INDEX,
                  "HANDLE_MARKED_FOR_DELETION", convert_seconds(0, true, true), user_id, &hvt);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "ERROR! In 'Handle_Type::delete_from_database':"
    ≪ endl ≪ "'Handle_Type::add_value' failed, returning "
    ≪ status ≪ ":" ≪ endl ≪
    "Failed to add handle value of type \"HANDLE_MARKED_FOR_DELETION\"."
    ≪ endl ≪
    "Exiting function unsuccessfully with return value 1."
    ≪ endl;
    unlock_cerr_mutex();
    return 1;
} /* if (status ≠ 0) */
```

933.

```
<Handle_Type::delete_from_database definition 927> +≡
#ifndef DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "In 'Handle_Type::delete_from_database':"
    ≪ endl ≪ "'Handle_Type::add_value' succeeded, returning 0."
    ≪ endl ≪
    "Added handle value of type \"HANDLE_MARKED_FOR_DELETION\" successfully."
    ≪ endl;
    hvt.show("hvt:");
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
/* if (¬(options & 1U)) Not immediate. */
```

934.

```

⟨ Handle_Type :: delete_from_database definition 927 ⟩ +≡
  string handle_database = (standalone_handle) ? "handlesystem_standalone" : "handlesystem";
  sql_strm << "lock_tables" << handle_database << ".handles_write," <<
    "gwirdsif.Irods_Objects_Handles_write";
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Handle_Type::delete_from_database':" << endl <<
      "'sql_strm.str()' == " << sql_strm.str() << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  status = submit_mysql_query(sql_strm.str(), result, mysql_ptr);
  if (status != 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Handle_Type::delete_from_database':" <<
      endl << "'submit_mysql_query' failed, returning" <<
      status << "." << endl << "Failed to lock tables." << endl <<
      "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    return 1;
  }
#endif DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << thread_str << "In 'Handle_Type::delete_from_database':" <<
        endl << "'submit_mysql_query' succeeded, returning 0." << endl <<
        "Locked tables successfully." << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  mysql_free_result(result);
  result = 0;

```

935. The value used below 31622400 = Number of seconds in a year + 1 day (366 days). This way, **Handle_Type** objects marked for immediate deletion can easily be identified because their *delete_from_database_timestamp* values will be more than a year older than the current date/time. [LDF 2013.07.21.] */

Log

[LDF 2013.07.25.] BUG FIX: Now using *purge_database_limit* instead of *purge_database_interval* to set the *last_modified* field.

Added code to ensure that the value used to set the *last_modified* field is reasonable.

```
<Handle_Type::delete_from_database definition 927> +≡
    sql_strm.str("");
    sql_strm << "update " << handle_database << ".handles_set_marked_for_deletion_=1,";
    if (options & 1U) /* IMMEDIATE */
    {
        time_t temp_time = time(0);
        time_t temp_time_1 = purge_database_limit + 31622400;
        if (temp_time_1 ≥ temp_time) {
            lock_cerr_mutex();
            cerr << thread_str << "WARNING! In 'Handle_Type::delete_from_database':"
                << endl << "'temp_time_1' >= 'temp_time':"
                << endl << "'temp_time' == 'time(0)':"
                << temp_time << endl << "'temp_time_1' == 'purge_database_limit' + 31622400 == "
                << temp_time_1 << endl << "'purge_database_limit' == "
                << purge_database_limit << endl << "31622400 == number_of_seconds_per_366_days(1 year + 1 day)"
                << endl << "Can't subtract 'temp_time_1' from 'temp_time'." << endl <<
                "Setting 'delete_from_database_timestamp' to 'purge_database_limit' - 1 == "
                (purge_database_limit - 1) << endl;
            unlock_cerr_mutex();
            temp_time -= purge_database_limit + 1;
        } /* if (temp_time_1 ≥ temp_time) */
        else temp_time -= temp_time_1;
        sql_strm << "delete_from_database_timestamp=" << convert_seconds(temp_time) << ",";
    } /* if (options & 1U) IMMEDIATE */
```

936.

Log

[LDF 2013.08.26.] Added this section.

```
<Handle_Type::delete_from_database definition 927> +≡
else
    if (delay_value > 1) /* Non-default delay */
    {
        sql_strm << "delete_from_database_timestamp=" << convert_seconds(delay_time) << ",";
    }
```

937.

```

⟨ Handle_Type :: delete_from_database definition 927 ⟩ +≡
else      /* Not immediate, default delay */
{
  sql_strm ≪ "delete_from_database_timestamp=" ≪ convert_seconds() ≪ ",";
}
sql_strm ≪ "last_modified=now()" ≪ "where handle_id=" ≪ handle_id ≪
"and marked_for_deletion=0";
#ifndef DEBUG_COMPILE
if (DEBUG) {
  lock_cerr_mutex();
  cerr ≪ thread_str ≪ "In 'Handle_Type::delete_from_database':" ≪ endl ≪
    "'sql_strm.str()'==" ≪ sql_strm.str() ≪ endl;
  unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
status = submit_mysql_query(sql_strm.str(), result, mysql_ptr, 0, 0, &affected_rows);
if (status ≠ 0) {
  lock_cerr_mutex();
  cerr ≪ thread_str ≪ "ERROR! In 'Handle_Type::delete_from_database':" ≪
    endl ≪ "'submit_mysql_query' failed, returning" ≪ status ≪ "." ≪ endl ≪
    "Failed to mark entries in '" ≪ handle_database ≪ ".handles' table" ≪
    "for deletion." ≪ endl ≪ "Exiting function unsuccessfully with return value 1." ≪
    endl;
  unlock_cerr_mutex();
  if (result) mysql_free_result(result);
  result = 0;
  ret_val = 1;
  goto UNLOCK_TABLES;
}

```

938.

```
{Handle_Type::delete_from_database definition 927} +≡
else
  if (affected_rows ≡ 0) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "WARNING! In 'Handle_Type::delete_from_database':"
    endl ≪ "'affected_rows'==0" ≪ endl ≪ "No rows in "
    handle_database ≪ ".handles_table" ≪ "marked_for_deletion."
    endl ≪ "Exiting function unsuccessfully with return value 2." ≪ endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    result = 0;
    ret_val = 2;
    goto UNLOCK_TABLES;
  }
#endif DEBUG_COMPILE
else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "In 'Handle_Type::delete_from_database':"
    endl ≪ "'submit_mysql_query'succeeded, returning 0." ≪ endl ≪ "'affected_rows'=="
    affected_rows ≪ endl ≪ "Marked entries for deletion from "
    handle_database ≪ ".handles_table" ≪ "successfully." ≪ endl ≪ "'affected_rows'=="
    affected_rows ≪ endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
if (handle_rows) *handle_rows = affected_rows;
mysql_free_result(result);
result = 0;
```

939.

```

⟨ Handle_Type :: delete_from_database definition 927 ⟩ +≡
UNLOCK_TABLES:
  if (result) {
    mysql_free_result(result);
    result = 0;
  }
  sqlStrm.str("");
  sqlStrm << "unlock_tables";
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Handle_Type::delete_from_database':" << endl <<
      "'sql_strm.str()' == " << sqlStrm.str() << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  status = submit_mysql_query(sqlStrm.str(), result, mysql_ptr);
  if (status != 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Handle_Type::delete_from_database':" <<
      endl << "'submit_mysql_query' failed, returning" << status <<
      "." << endl << "Failed to unlock_tables." << endl <<
      "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    return 1;
  }
#endif DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << thread_str << "In 'Handle_Type::delete_from_database':" <<
        endl << "'submit_mysql_query' succeeded, returning 0." << endl <<
        "Unlocked_tables successfully." << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  mysql_free_result(result);
  result = 0;

```

940.

Log

[LDF 2013.07.18.] Added this section.
 [LDF 2013.08.19.] Added code for checking whether the “purge database” thread is running.

```

⟨Handle_Type::delete_from_database definition 927⟩ +≡
  if (options & 1U) /* IMMEDIATE */
  {
    if (purge_database_thread_id == static_cast<pthread_t>(0)) {
      lock_cerr_mutex();
      cerr << thread_str << "NOTICE! In 'Handle_Type::delete_from_database':"
          << endl << "'purge_database_thread_id'==0." << endl <<
          "\\"Purge_database\\\"thread\\not\\running.\\\"Not\\calling\\\"pthread_cond_signal\\\""
          "<< endl << "to\\wake\\it\\up." << endl << "Continuing." << endl;
      unlock_cerr_mutex();
    } /* if (purge_database_thread_id == 0) */
    else {
      pthread_mutex_lock(&purge_server_database_mutex);
      status = pthread_cond_signal(&purge_server_database_cond);
      if (status != 0) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'Handle_Type::delete_from_database':"
            << endl << "'pthread_cond_signal' failed, returning" << status << ":" << endl << "Error:\\\""
            strerror(status) << endl << "Exiting\\function\\unsuccessfully\\with\\return\\value\\\"1."
            << endl;
        unlock_cerr_mutex();
        ret_val = 1;
      } /* if (status != 0) */
    }
  #if DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << thread_str << "In 'Handle_Type::delete_from_database':"
          << endl << "'pthread_cond_signal' succeeded, returning\\\"0." << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
  #endif /* DEBUG_COMPILE */
  pthread_mutex_unlock(&purge_server_database_mutex);
  } /* else */
} /* if (options & 1U) (IMMEDIATE) */

```

941. Exit function with return value *ret_val*. [LDF 2013.07.25.]

This may be a successful exit or not. [LDF 2013.07.25.]

```
<Handle_Type::delete_from_database definition 927> +≡
  if (ret_val ≠ 0) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Handle_Type::delete_from_database'" << endl << "'ret_val' == " <<
      ret_val << " != 0" << endl << "Exiting function with return value" << ret_val << "." <<
      endl;
    unlock_cerr_mutex();
    return ret_val;
  } /* if (ret_val ≠ 0) */
```

942. Exit function successfully with return value 0. [LDF 2013.07.25.]

```
<Handle_Type::delete_from_database definition 927> +≡
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "Exiting 'Handle_Type::delete_from_database' successfully" <<
      "with return value 0." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  return 0; } /* End of Handle_Type::delete_from_database definition */
```

943. Find (*find*). [LDF 2013.07.04.]

Log

[LDF 2013.07.04.] Added this function.

```
<class Handle_Type function declarations 889> +≡
  const map<unsigned long int, Handle_Value_Type>::iterator find(string type);
```

944.

```
<Handle_Type::find definition 944> ≡
  const map<unsigned long int, Handle_Value_Type>::iterator Handle_Type::find(string type){
    bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status = 0;
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "Entering 'Handle_Type::find' ." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

See also sections 945 and 946.

This code is used in section 992.

945.

```
<Handle_Type::find definition 944> +≡
for (map<unsigned long int,Handle_Value_Type>::iterator iter = handle_value_map.begin();
     iter ≠ handle_value_map.end(); ++iter) {
    if (iter->second.type ≡ type) return iter;
}
```

946.

```
<Handle_Type::find definition 944> +≡
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "In 'Handle_Type::find': Handle_value with 'type' == " ≪ type ≪
          "' not found." ≪ endl ≪ "Exiting function successfully with return value"
          "handle_value_map.end()." ≪ endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
return handle_value_map.end(); } /* End of Handle_Type::find definition */
```

947. Functions for retrieving values from fields. [LDF 2013.07.17.]

The fields are actually data members of **Handle_Value_Type**. [LDF 2013.07.17.]

948. *created_by_user_id*. [LDF 2013.07.17.]

Log

[LDF 2013.07.17.] Added this function.

```
<class Handle_Type function declarations 889> +≡
int created_by_user_id(void);
```

949.

```
<Handle_Type::created_by_user_id definition 949> ≡
int Handle_Type::created_by_user_id(void)
{
    bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    if (handle_value_map.size() ≡ 0) return 0;
    else {
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr ≪ "handle_value_map.begin()->second.created_by_user_id == "
                  "handle_value_map.begin()->second.created_by_user_id" ≪ endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        return handle_value_map.begin()->second.created_by_user_id;
    } /* else */
}
```

This code is used in section 992.

950. Fetch handles from database (version with vector arguments). [LDF 2013.08.12.]

Log

[LDF 2013.08.12.] Added this function.

951.

```
< class Handle_Type function declarations 889 > +≡
static int fetch_handles_from_database(MYSQL *&mysql_ptr, vector<unsigned long int>
    &handle_id_vector, vector<Handle_Type> &handle_vector, string type = "", string
    thread_str = "");
```

952.

```
< Handle_Type::fetch_handles_from_database definition 952 > ≡
int Handle_Type::fetch_handles_from_database(MYSQL *&mysql_ptr, vector<unsigned long int>
    &handle_id_vector, vector<Handle_Type> &handle_vector, string type, string thread_str){
    int status;
    bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    MYSQL_RES *result = 0;
    unsigned int row_ctr;
    unsigned int field_ctr;
    stringstream sql_strm;
    MYSQL_ROW curr_row;
    Handle_Type curr_handle;
    Handle_Value_Type curr_handle_value;
    unsigned long int prev_handle_id = 0;
    char temp_str[256];
    memset(temp_str, 0, 256);
    string handle_database = (standalone_handle) ? "handlesystem_standalone" : "handlesystem";
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering 'Handle_Type::fetch_handles_from_database' " <<
            "(with vector arguments)." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

See also sections 953, 954, 955, 956, and 957.

This code is used in section 992.

953.

```

⟨Handle_Type::fetch_handles_from_database definition 952⟩ +≡
  sql_strm << "select handle_id, handle_value_id, idx, type, length(data), \
    data, ttl_type, " << "ttl, timestamp, length(refs), refs, admin_read, admin_write \
    , pub_read, " << "pub_write, marked_for_deletion, created, last_modified, handle, " <<
    "created_by_user_id, " << "irods_object_id, delete_from_database_timestamp" <<
    "from" << handle_database << ".handles where" << "handle_id in(";
  string comma_str = "";
  for (vector<unsigned long int>::const_iterator iter = handle_id_vector.begin();
       iter != handle_id_vector.end(); ++iter) {
    sql_strm << comma_str << *iter;
    comma_str = ",";
  } /* for */
  sql_strm << ")";
  if (!type.empty()) {
    sql_strm << " and type=" << type << ',';
  } /* if (!type.empty()) */
  sql_strm << "order by handle, idx";
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "sql_strm.str() == " << sql_strm.str() << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  status = submit_mysql_query(sql_strm.str().c_str(), result, mysql_ptr, &row_ctr, &field_ctr);
  if (status != 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Handle_Type::fetch_handles_from_database':" << endl <<
      "'Scan_Parse_Parameter_Type::submit_mysql_query' failed, returning" <<
      status << "." << endl << mysql_error(mysql_ptr) << endl <<
      "Failed to fetch handles from database." << endl <<
      "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    if (result) {
      mysql_free_result(result);
    }
    return 1;
  } /* if (status != 0) */
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << thread_str << "In 'Handle_Type::fetch_handles_from_database':" << endl <<
        "'Scan_Parse_Parameter_Type::submit_mysql_query' succeeded." << endl <<
        "'row_ctr' == " << row_ctr << endl << "'field_ctr' == " << field_ctr << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

954.

```
<Handle_Type::fetch_handles_from_database definition 952> +≡
  if (row_ctr ≡ 0) {
#if DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Handle_Type::fetch_handles_from_database':"
    << endl << "Scan_Parse_Parameter_Type::submit_mysql_query' returned 0 rows."
    << endl << "No handles fetched." << endl <<
    "Exiting function successfully with return value 2." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  mysql_free_result(result);
  return 2;
} /* if (row_ctr ≡ 0) */
```

955.

```
<Handle_Type::fetch_handles_from_database definition 952> +≡
#ifndef DEBUG_COMPILE
else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Handle_Type::fetch_handles_from_database':"
    << endl << "Scan_Parse_Parameter_Type::submit_mysql_query' returned"
    << row_ctr << " rows." << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

956.

```

⟨ Handle_Type::fetch_handles_from_database definition 952 ⟩ +≡
for (int i = 0; i < row_ctr; ++i) {
    if ((curr_row = mysql_fetch_row(result)) == 0) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'Handle_Type::fetch_handles_from_database':"
        endl << "'mysql_fetch_row' failed:" << endl << mysql_error(mysql_ptr) << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        mysql_free_result(result);
        return 1;
    } /* if (curr_row = mysql_fetch_row(result) == 0) */
#endif /* DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Handle_Type::fetch_handles_from_database':"
    << endl << "'mysql_fetch_row' succeeded."
    << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
curr_handle_value.clear();
status = curr_handle_value.set(curr_row, field_ctr);
if (status == 2) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Handle_Type::fetch_handles_from_database':"
    << endl << "'Handle_Value_Type::set' returned 2: No entry found in the 'handles' "
    << "database." << endl << "Exiting function unsuccessfully with return value 1."
    << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    return 1;
} /* if (status == 2) */
else if (status != 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Handle_Type::fetch_handles_from_database':"
    << endl << "'Handle_Value_Type::set' failed, returning"
    << status << "."
    << endl << "Exiting function unsuccessfully with return value 1."
    << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    return 1;
} /* if (status != 0) */
#endif /* DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Handle_Type::fetch_handles_from_database':"
    << endl << "'Handle_Value_Type::set' succeeded, returning 0."
    << endl;
    curr_handle_value.show("curr_handle_value:");
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

```

if (type ≡ "IRODS_OBJECT" ∨ type ≡ "DC_METADATA_IRODS_OBJECT") {
    memset(temp_str, 0, 256);
    strncpy(temp_str, curr_handle_value.data, curr_handle_value.data_length);
    curr_handle_value.filename = temp_str;
}
if (prev_handle_id ≡ 0) {
    curr_handle.handle = curr_handle_value.handle;
    curr_handle.handle_id = curr_handle_value.handle_id;
    curr_handle.handle_value_map[curr_handle_value.idx] = curr_handle_value;
}
else if (prev_handle_id ≡ curr_handle_value.handle_id) {
    curr_handle.handle_value_map[curr_handle_value.idx] = curr_handle_value;
}
else /* Create new Handle_Type object */
{
    handle_vector.push_back(curr_handle);
    curr_handle.clear();
    curr_handle.handle = curr_handle_value.handle;
    curr_handle.handle_id = curr_handle_value.handle_id;
    curr_handle.handle_value_map[curr_handle_value.idx] = curr_handle_value;
}
prev_handle_id = curr_handle_value.handle_id;
curr_handle_value.clear();
} /* for */
if (curr_handle.handle_value_map.size() > 0) {
    handle_vector.push_back(curr_handle);
}
mysql_free_result(result);
result = 0;

```

957.

```

⟨Handle_Type::fetch_handles_from_database definition 952⟩ +≡
#if DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Exiting 'Handle_Type::fetch_handles_from_database' " <<
            "(with_vector arguments) successfully with return value 0." << endl <<
            "handle_vector.size() == " << handle_vector.size() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0;
/* End of Handle_Type::fetch_handles_from_database (with vector arguments) definition */

```

958. Fetch single handle from database (*fetch_handle_from_database*). [LDF 2013.08.14.]

Log

[LDF 2013.08.14.] Added this function.

959.

```
< class Handle_Type function declarations 889 > +≡
  static int fetch_handle_from_database(MYSQL *&mysql_ptr, unsigned long int handle_id, Handle_Type
    &handle, string type = "", string thread_str = "");
```

960.

```
< Handle_Type::fetch_handle_from_database definition 960 > ≡
  int Handle_Type::fetch_handle_from_database(MYSQL * &mysql_ptr, unsigned long int
    handle_id, Handle_Type &handle, string type, string thread_str){ int status;
  bool DEBUG = false; /* true */
  set_debug_level(DEBUG, 0, 0);
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "Entering 'Handle_Type::fetch_handle_from_database'." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
See also sections 961, 962, 963, and 964.
This code is used in section 992.
```

961.

```
< Handle_Type::fetch_handle_from_database definition 960 > +≡
  string handle_database = (standalone_handle) ? "handlesystem_standalone" : "handlesystem";
  vector<unsigned long int> handle_id_vector;
  vector<Handle_Type> handle_vector;
  handle_id_vector.push_back(handle_id);
  status = fetch_handles_from_database(mysql_ptr, handle_id_vector, handle_vector, type, thread_str);
  if (status ≡ 2) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Handle_Type::fetch_handle_from_database':" << endl <<
      "'Handle_Type::fetch_handles_from_database'(with_vector_arguments)" <<
      "returned 2. No handles fetched." << endl <<
      "Exiting function successfully with return value 2." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  return 2;
} /* if (status ≡ 2) */
else if (status ≠ 0) {
  lock_cerr_mutex();
  cerr << thread_str << "ERROR! In 'Handle_Type::fetch_handle_from_database':" << endl <<
    "'Handle_Type::fetch_handles_from_database'(with_vector_arguments)" <<
    "failed, returning" << status << "." << endl <<
    "Exiting function unsuccessfully with return value" << status << "." << endl;
  unlock_cerr_mutex();
  return status;
} /* if (status ≠ 0) */
```

962.

```
<Handle_Type::fetch_handle_from_database definition 960> +≡
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_str << "In 'Handle_Type::fetch_handle_from_database':" << endl <<
                "'Handle_Type::fetch_handles_from_database'(with_vector<arguments>)" <<
                "succeeded, returning 0." << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

963.

```
<Handle_Type::fetch_handle_from_database definition 960> +≡
if (handle_vector.size() == 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Handle_Type::fetch_handle_from_database':" <<
        endl << "'handle_vector' is empty. Failed to fetch handle from" <<
        endl << handle_database << ".handles' database_table." << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    return 1;
} /* if (handle_vector.size() == 0) */
handle = handle_vector.front();
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Handle_Type::fetch_handle_from_database':" << endl;
        handle.show("handle:");
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

964.

```
<Handle_Type::fetch_handle_from_database definition 960> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Exiting 'Handle_Type::fetch_handle_from_database' successfully" <<
            endl << "with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
return 0; } /* End of Handle_Type::fetch_handle_from_database definition */
```

965. Delete handle values (*delete_handle_values—static* version). [LDF 2013.08.19.]

```
<class Handle_Type function declarations 889> +≡
static int delete_handle_values ( std::set<unsigned long int> &handle_id_set, vector<string>
    &handle_value_type_vector, MYSQL*&mysql_ptr, string thread_str = "", bool
    lock_table = true );
```

966.

```

⟨ Handle_Type :: delete_handle_values definitions 966 ⟩ ≡
int Handle_Type :: delete_handle_values ( std::set < unsigned long int > &handle_id_set,
                                         vector<string> &handle_value_type_vector, MYSQL*&mysql_ptr, string thread_str,
                                         bool lock_table ) { bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status = 0;
    stringstream sql_strm;
    MYSQL_RES * result = 0;
    MYSQL_ROW curr_row;
    unsigned int row_ctr;
    unsigned int field_ctr;
    long int affected_rows;
    string handles_database = (standalone_handle) ? "handlesystem_standalone" :
                                              "handlesystem";
    int ret_val = 0;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering 'Handle_Type::delete_handle_val"
            lues'" (static_version). " << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

See also sections 967, 968, 969, 970, 971, 972, 973, 974, 975, 976, 978, 979, and 980.

This code is used in section 992.

967.

```

⟨ Handle_Type :: delete_handle_values definitions 966 ⟩ +≡
if (handle_id_set.size() ≡ 0 ∨ handle_value_type_vector.size() ≡ 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Handle_Type::delete_handle_values':"
        << endl << "'handle_id_set' and/or 'handle_value_type_vector' is empty."
        << endl << "Not deleting handle values." << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    return 1;
} /* if */

```

968.

```

⟨ Handle_Type::delete_handle_values definitions 966 ⟩ +≡
  if (lock_table) {
    sql_strm << "lock_tables" << handles_database << ".handles_write";
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Handle_Type::delete_handle_values':" << endl <<
      "'lock_table'" == 'true' . Will_try_to_lock " << handles_database << ".handles" <<
      "database_table." << endl << "'sql_strm.str()'" == " << sql_strm.str() << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  status = submit_mysql_query(sql_strm.str(), result, mysql_ptr);
  if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Handle_Type::delete_handle_values':" <<
      endl << "'submit_mysql_query' failed, returning" << status << "." << endl <<
      "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    return 1;
  } /* if (status ≠ 0) */
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << thread_str << "In 'Handle_Type::delete_handle_values':" << endl <<
        "'submit_mysql_query' succeeded, returning 0." << endl << "Locked" <<
        handles_database << ".handles" << "database_table successfully." << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  mysql_free_result(result);
  result = 0;
} /* if (lock_table) */

```

969.

```

⟨ Handle_Type::delete_handle_values definitions 966 ⟩ +≡
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << thread_str << "In 'Handle_Type::delete_handle_values':" << endl <<
        "'lock_table'" == 'false' . Not_locking " << handles_database << ".handles" <<
        "database_table." << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

970.

```

⟨Handle_Type::delete_handle_values definitions 966⟩ +≡
sql_strm.str("");
string comma_str = "";
sql_strm << "delete_from" << handles_database << ".handles_where_handle_id_in("; for ( set <
unsigned long int > ::iterator iter = handle_id_set.begin();
iter != handle_id_set.end(); ++iter )
{
    sql_strm << comma_str << *iter;
    comma_str = ",";
}
sql_strm << ")" and type_in(";
comma_str = "";
for (vector<string>::iterator iter = handle_value_type_vector.begin();
     iter != handle_value_type_vector.end(); ++iter) {
    sql_strm << comma_str << "," << *iter << ",";
    comma_str = ",";
}
sql_strm << ")";
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Handle_Type::delete_handle_values':" << endl <<
        "'sql_strm.str()' == " << sql_strm.str() << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

971.

```

⟨Handle_Type::delete_handle_values definitions 966⟩ +≡
status = submit_mysql_query(sql_strm.str(), result, mysql_ptr, 0, 0, &affected_rows);
if (status != 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Handle_Type::delete_handle_values':" <<
        endl << "'submit_mysql_query' failed, returning" << status << "." << endl <<
        "Will try to unlock" << handles_database << ".handles_database_table" <<
        "before exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    ret_val = 1;
    goto UNLOCK_TABLES;
} /* if (status != 0) */

```

972.

```
<Handle_Type::delete_handle_values definitions 966> +≡
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_str << "In `Handle_Type::delete_handle_values':"
            << endl << "Deleted row(s) from `"
            handles_database << ".handles` database table`"
            << " successfully." << endl <<
            "'affected_rows' == `"
            << affected_rows << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
mysql_free_result(result);
result = 0;
```

973.

```

⟨ Handle_Type :: delete_handle_values definitions 966 ⟩ +≡
UNLOCK_TABLES:
  if (result) {
    mysql_free_result(result);
    result = 0;
  }
  if (lock_table) {
    sql_strm.str("");
    sql_strm << "unlock_tables";
  }
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Handle_Type::delete_handle_values':" << endl <<
      "'lock_table'" == "true". Will try to unlock " << handles_database << ".handles'" <<
      "database_table." << endl << "'sql_strm.str()' == " << sql_strm.str() << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  status = submit_mysql_query(sql_strm.str(), result, mysql_ptr);
  if (status != 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Handle_Type::delete_handle_values':" <<
      "'submit_mysql_query' failed, returning " << status << "." << endl <<
      "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    return 1;
  } /* if (status != 0) */
#endif DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << thread_str << "In 'Handle_Type::delete_handle_values':" << endl <<
        "'submit_mysql_query' succeeded, returning 0." << endl << "Unlocked '" <<
        handles_database << ".handles'" << "database_table successfully." << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  mysql_free_result(result);
  result = 0;
} /* if (lock_table) */

```

974.

```
<Handle_Type::delete_handle_values definitions 966> +≡
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_str << "In 'Handle_Type::delete_handle_values':" << endl <<
                "'lock_table'==false'. Not unlocking" << handles_database << ".handles" <<
                "database_table." << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

975.

```
<Handle_Type::delete_handle_values definitions 966> +≡
if (ret_val ≠ 0) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Handle_Type::delete_handle_values':" <<
        endl << "'ret_val'==0" << ret_val << "≠0" << endl <<
        "Exiting function unsuccessfully with return value 'ret_val'==0" <<
        ret_val << "." << endl;
    unlock_cerr_mutex();
    return ret_val;
} /* if (ret_val ≠ 0) */
```

976.

```
<Handle_Type::delete_handle_values definitions 966> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Exiting 'Handle_Type::delete_handle_values' (static version)" <<
            "successfully with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; /* End of Handle_Type::delete_handle_values (static version) definition */
```

977. Delete handle values (*delete_handle_values*—member version). [LDF 2013.08.19.]

```
<class Handle_Type function declarations 889> +≡
int delete_handle_values(vector<string> &handle_value_type_vector, MYSQL *&mysql_ptr, string
    thread_str = "", bool lock_table = true);
```

978.

```

⟨ Handle_Type::delete_handle_values definitions 966 ⟩ +≡
int Handle_Type::delete_handle_values(vector<string> &handle_value_type_vector,
                                       MYSQL *&mysql_ptr, string thread_str, bool lock_table){ bool DEBUG = false; /* true */
set_debug_level(DEBUG, 0, 0);
int status = 0;
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "Entering 'Handle_Type::delete_handle_values' (member_version\
n)." << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
std::set < unsigned long int > handle_id_set;
handle_id_set.insert(handle_id);
status = delete_handle_values(handle_id_set, handle_value_type_vector, mysql_ptr, thread_str,
                             lock_table);
if (status != 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Handle_Type::delete_handle_values' (member_version\
on):" << endl << "'Handle_Type::delete_handle_values'(st\
atic_version)_failed, returning" << status << "." << endl <<
    "Exiting function unsuccessfully with return value 'status' == " << status <<
    "." << endl;
    unlock_cerr_mutex();
    return status;
} /* if (status != 0) */

```

979.

```

⟨ Handle_Type::delete_handle_values definitions 966 ⟩ +≡
#ifndef DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Handle_Type::delete_handle_values' (member_version):" << endl <<
        "'Handle_Type::delete_handle_values'(static_version)_succeeded, returning 0." <<
        endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

980.

```
<Handle_Type::delete_handle_values definitions 966> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Exiting 'Handle_Type::delete_handle_values' (member version) "
            << "successfully with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; } /* End of Handle_Type::delete_handle_values (member version) definition */
```

981. Unmark for deletion (*unmark_handle_for_deletion*). [LDF 2013.08.22.]

Log

[LDF 2013.08.22.] Added this function.

[LDF 2013.09.11.] Renamed this function from *unmark_for_deletion* to *unmark_handle_for_deletion*.

```
<class Handle_Type function declarations 889> +≡
int unmark_handle_for_deletion(MYSQL *&mysql_ptr, string thread_str = "");
```

982.

```
<Handle_Type::unmark_handle_for_deletion definition 982> ≡
int Handle_Type::unmark_handle_for_deletion(MYSQL *&mysql_ptr, string thread_str){ bool
    DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status = 0;
    MYSQL_RES * result = 0;
    MYSQL_ROW curr_row;
    unsigned int row_ctr;
    unsigned int field_ctr;
    long int affected_rows;
    stringstream sql_strm;
    map<unsigned long int, Handle_Value_Type>::iterator hv_iter;
    int ret_val = 0;
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering 'Handle_Type::unmark_handle_for_deletion'." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

See also sections 983, 984, 985, 986, 987, 988, 989, 990, and 991.

This code is used in section 992.

983.

```
<Handle_Type::unmark_handle_for_deletion definition 982> +≡
  if (handle_value_map.size() == 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Handle_Type::unmark_handle_for_deletion':"
    endl << "'handle_value_map.size()' == 0. No handle values." << endl <<
      "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    return 1;
} /* if (handle_value_map.size() == 0) */
```

984.

```
<Handle_Type::unmark_handle_for_deletion definition 982> +≡
  hv_iter = find("HANDLE_MARKED_FOR_DELETION");
  if (hv_iter == handle_value_map.end()) {
    lock_cerr_mutex();
    cerr << thread_str << "WARNING! In 'Handle_Type::unmark_handle_for_deletion':"
    endl << "Handle " << handle << "' isn't marked for deletion.' <<
    endl << "Can't unmark handle for deletion." << endl <<
      "Exiting function unsuccessfully with return value "
      "'GW_HANDLE_NOT_MARKED_FOR_DELETION'." << endl;
    unlock_cerr_mutex();
    return GW_HANDLE_NOT_MARKED_FOR_DELETION;
} /* if (hv_iter == handle_value_map.end()) */
```

985.

```
<Handle_Type::unmark_handle_for_deletion definition 982> +≡
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << thread_str << "In 'Handle_Type::unmark_handle_for_deletion':"
      << endl <<
        "Handle " << handle << "' is marked for deletion.' << endl <<
          "Will unmark." << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

986.

```

⟨ Handle_Type :: unmark_handle_for_deletion definition 982 ⟩ +≡
  string handles_database = (standalone_handle) ? "handlesystem_standalone" : "handlesystem";
  sql_strm ≪ "lock_tables" ≪ handles_database ≪ ".handles_write";
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "In‘Handle_Type::unmark_handle_for_deletion’:"
      ≪ endl ≪ "'sql_strm.str()'"
      ≪ sql_strm.str() ≪ endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  status = submit_mysql_query(sql_strm.str(), result, mysql_ptr);
  if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "ERROR! In‘Handle_Type::unmark_handle_for_deletion’:"
      ≪ endl ≪ "'submit_mysql_query' failed, returning"
      ≪ status ≪ "."
      ≪ endl ≪ "Failed to lock"
      ≪ handles_database ≪ ".handles’database_table."
      ≪ endl ≪ "Exiting function unsuccessfully with return value 1."
      ≪ endl;
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    return 1;
  } /* if (status ≠ 0) */
#endif DEBUG_COMPILE
  else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "In‘Handle_Type::unmark_handle_for_deletion’:"
      ≪ endl ≪ "'submit_mysql_query' succeeded, returning 0."
      ≪ endl ≪ "Locked"
      ≪ handles_database ≪ ".handles’database_table successfully."
      ≪ endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  mysql_free_result(result);
  result = 0;
  sql_strm.str("");

```

987.

```
<Handle_Type::unmark_handle_for_deletion definition 982> +≡
  sql_strm << "delete from " << handles_database << ".handles" << "where handle_value_id=" <<
    hv_iter->second.handle_value_id;
#endif DEBUG_COMPILE
if (DEBUG) {
  lock_cerr_mutex();
  cerr << thread_str << "In 'Handle_Type::unmark_handle_for_deletion':" << endl <<
    "'sql_strm.str()' == " << sql_strm.str() << endl;
  unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
status = submit_mysql_query(sql_strm.str(), result, mysql_ptr, 0, 0, &affected_rows);
if (status != 0) {
  lock_cerr_mutex();
  cerr << thread_str << "ERROR! In 'Handle_Type::unmark_handle_for_deletion':" <<
    endl << "'submit_mysql_query' failed, returning" <<
    status << "." << endl << "Failed to delete row from " <<
    handles_database << ".handles.database.table." << endl <<
    "Will try to unlock tables and exit function unsuccessfully with return value 1." <<
    endl;
  unlock_cerr_mutex();
  if (result) {
    mysql_free_result(result);
    result = 0;
  }
  ret_val = 1;
  goto UNLOCK_TABLES;
} /* if (status != 0) */
#endif DEBUG_COMPILE
else
if (DEBUG) {
  lock_cerr_mutex();
  cerr << thread_str << "In 'Handle_Type::unmark_handle_for_deletion':" << endl <<
    "'submit_mysql_query' succeeded, returning 0." << endl << "'affected_rows' == " <<
    affected_rows << endl;
  unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
mysql_free_result(result);
result = 0;
sql_strm.str("");
```

988.

```

⟨Handle_Type::unmark_handle_for_deletion definition 982⟩ +≡
  sql_strm << "update" << handles_database << ".handles" <<
    "set_marked_for_deletion=0,delete_from_database_timestamp=0," <<
    "last_modified=now()" << "where handle_id=" << handle_id;
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Handle_Type::unmark_handle_for_deletion':" << endl <<
      "'sql_strm.str()' == " << sql_strm.str() << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
status = submit_mysql_query(sql_strm.str(), result, mysql_ptr, 0, 0, &affected_rows);
if (status != 0) {
  lock_cerr_mutex();
  cerr << thread_str << "ERROR! In 'Handle_Type::unmark_handle_for_deletion':" <<
    endl << "'submit_mysql_query' failed, returning" <<
    status << "." << endl << "Failed to update rows in " <<
    handles_database << ".handles'database'table." << endl <<
    "Will try to unlock tables and exit function unsuccessfully with return value 1." <<
    endl;
  unlock_cerr_mutex();
  if (result) {
    mysql_free_result(result);
    result = 0;
  }
  ret_val = 1;
  goto UNLOCK_TABLES;
} /* if (status != 0) */
#ifndef DEBUG_COMPILE
else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Handle_Type::unmark_handle_for_deletion':" << endl <<
      "'submit_mysql_query' succeeded, returning 0." << endl << "'affected_rows' == " <<
      affected_rows << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
mysql_free_result(result);
result = 0;
sql_strm.str("");

```

989.

```

⟨ Handle_Type::unmark_handle_for_deletion definition 982 ⟩ +≡
UNLOCK_TABLES:
  if (result) {
    mysql_free_result(result);
    result = 0;
  }
  sqlStrm.str("");
  sqlStrm << "unlock_tables";
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Handle_Type::unmark_handle_for_deletion':"
      << endl <<
      "'sql_strm.str()' == " << sqlStrm.str() << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  status = submit_mysql_query(sqlStrm.str(), result, mysql_ptr);
  if (status != 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Handle_Type::unmark_handle_for_deletion':"
      << endl << "'submit_mysql_query' failed, returning"
      << status <<
      "." << endl << "Failed to unlock database tables."
      << endl <<
      "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    return 1;
  } /* if (status != 0) */
#endif DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << thread_str << "In 'Handle_Type::unmark_handle_for_deletion':"
        << endl << "'submit_mysql_query' succeeded, returning 0."
        << endl <<
        "Unlocked database tables successfully." << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  mysql_free_result(result);
  result = 0;

```

990.

```

⟨ Handle_Type::unmark_handle_for_deletion definition 982 ⟩ +≡
  if (ret_val != 0) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Handle_Type::unmark_handle_for_deletion':"
      << endl <<
      "Exiting function with return value"
      << ret_val << "." << endl;
    unlock_cerr_mutex();
    return ret_val;
  } /* if (ret_val != 0) */

```

991.

```

⟨ Handle_Type :: unmark_handle_for_deletion definition 982 ⟩ +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Exiting `Handle_Type::unmark_handle_for_deletion' successfully"
        "with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; } /* End of Handle_Type :: unmark_handle_for_deletion definition */

```

992. Putting Handle_Type together. [LDF 2012.10.12.]

```

⟨ Include files 3 ⟩
using namespace std;
using namespace gwrdifpk;
class Irods_Object_Type;
typedef void *yyscan_t;
class Scan_Parse_Parameter_Type;
#ifndef 0
    class Handle_Type;
    class Handle_Value_Type;
#endif
⟨ Declare class Handle_Type 887 ⟩
⟨ Handle_Type constructor definitions 890 ⟩
⟨ Handle_Type :: operator= definition 894 ⟩
⟨ Handle_Type :: show definition 898 ⟩
⟨ Handle_Type :: clear definition 896 ⟩
⟨ Handle_Type :: add_value definition 900 ⟩
⟨ Handle_Type :: add_values definition 902 ⟩
⟨ Handle_Type :: delete_from_database definition 927 ⟩
⟨ Handle_Type :: find definition 944 ⟩
⟨ Handle_Type :: created_by_user_id definition 949 ⟩
⟨ Handle_Type :: fetch_handles_from_database definition 952 ⟩
⟨ Handle_Type :: fetch_handle_from_database definition 960 ⟩
⟨ Handle_Type :: delete_handle_values definitions 966 ⟩
⟨ Handle_Type :: unmark_handle_for_deletion definition 982 ⟩

```

993. This is what's written to the header file `hdltype.h`. [LDF 2008.12.05.]

```
<hdltype.h 993>≡
#ifndef HNDLTYPE_H
#define HNDLTYPE_H 1
    class Irods_Object_Type;
    class Scan_Parse_Parameter_Type;
    typedef void *yyscan_t;
#endif 0
    class Handle_Type;
    class Handle_Value_Type;
#endif
<Declare class Handle_Type 887>
#endif
```

994. `Irods_AVU_Type` (`irdsavtp.web`).

Log

[LDF 2013.01.10.] Added this file. It contains the `class` declaration of *Irods_AVU_Type*, which was formerly in `irdsavtp.web`.

995. Include files.

```
<Include files 3> +≡
#ifndef _XOPEN_SOURCE
#define _XOPEN_SOURCE
#endif
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <unistd.h>
#include <time.h>
#if 0
#include <sys/time.h>
#include <sys/stat.h>
#endif
#include <sys/types.h>
#if 0
#include <dirent.h>
#endif
#include <string.h>
#if 0
#include <pwd.h>
#endif
#include <errno.h>
#include <pthread.h>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <iostream>
#include <string>
#include <sstream>
#include <deque>
#include <map>
#include <vector>
#include <stack>
#include <set>
#include <gcrypt.h> /* for gcry-control */
#include <gnutls/gnutls.h>
#include <gnutls/x509.h>
#include <mysql.h>
#include <expat.h>
#if HAVE_CONFIG_H
#include <config.h>
#endif
#undef NAME_LEN
#undef LOCAL_HOST
#include "glblcnst.h++"
#include "glblvrb1.h++"
#include "excptntp.h++"
#include "utilfncts.h++"
#include "parser.h++"
#include "scanner.h++"
#include "grouptp.h++"
```

```
#include "hndlvltp.h++"
#include "dcmtddtp.h++"
#include "rspnntp.h++"
#include "irdsavtp.h++"
#include "irdsobtp.h++"
#include "hndltype.h++"
#include "dstngnmt.h++"
#include "x509cert.h++"
#include "scprpmtp.h++"
#include "usrinftp.h++"
```

996. class String_Triple declaration. [LDF 2012.10.11.]

Log

[LDF 2012.10.11.] Added this **class** declaration.
 [LDF 2013.07.17.] Added **string irods_object_path**.
 [LDF 2013.07.18.] Added **int user_id**.
 [LDF 2013.07.21.] Added **bool deleted_from_archive** and **bool deleted_from_gwirdsif_db**.
 [LDF 2013.07.21.] Added the functions **get_deleted_from_archive** and **get_deleted_from_gwirdsif_db**.

```
<Declare class Irods_AVU_Type 996> ≡
class Irods_AVU_Type {
    friend class Irods_Object_Type;
    friend class Scan_Parse_Parameter_Type;
    friend int zzparse(yyscan_t);
    unsigned long id;
    int user_id;
    unsigned long irods_object_id;
    bool deleted_from_archive;
    bool deleted_from_gwirdsif_db;
    string attribute;
    string value;
    string units;
    unsigned int time_set;
    string irods_object_path;

public: <Irods_AVU_Type function declarations 997>
    unsigned long get_id(void) const
    {
        return id;
    }
    bool get_deleted_from_archive(void) const
    {
        return deleted_from_archive;
    }
    bool get_deleted_from_gwirdsif_db(void) const
    {
        return deleted_from_gwirdsif_db;
    }
};
```

This code is cited in section 1034.

This code is used in section 1035.

997. Default constructor. [LDF 2012.10.11.]

Log

[LDF 2012.10.11.] Added this function.

{ Irods_AVU_Type function declarations 997 }
≡
Irods_AVU_Type(void);

See also sections 999, 1001, 1003, 1005, 1007, 1009, 1023, and 1025.

This code is used in section 996.

998.

{ Irods_AVU_Type constructor definitions 998 }
≡
Irods_AVU_Type::Irods_AVU_Type(void)
{
 id = *irods_object_id* = 0UL;
 user_id = 0;
 time_set = 0U;
 deleted_from_archive = *deleted_from_gwirdsif_db* = false;
 return;
}

See also sections 1000 and 1002.

This code is used in section 1034.

999. Non-default constructor. [LDF 2012.10.11.]

Log

[LDF 2012.10.11.] Added this function.

[LDF 2013.06.07.] Added optional argument **unsigned int iirods_object_id** = 0.

[LDF 2013.07.21.] Added optional arguments **bool ddeleted_from_archive** = false and **bool ddeleted_from_gwirdsif_db** = false.

{ Irods_AVU_Type function declarations 997 } +≡

Irods_AVU_Type(string attrib, string val, string u = "", unsigned int t = 0U, unsigned int iirods_object_id = 0U, int uuser_id = 0, bool ddeleted_from_archive = false, bool ddeleted_from_gwirdsif_db = false);

1000.

```
< Irods_AVU_Type constructor definitions 998 > +≡
Irods_AVU_Type::Irods_AVU_Type(string attrib, string val, string u, unsigned int t, unsigned
                                int iirods_object_id, int uuser_id, bool ddeleted_from_archive, bool ddeleted_from_gwirdsif_db)
{
    irods_object_id = iirods_object_id;
    user_id = uuser_id;
    id = 0UL;
    attribute = attrib;
    value = val;
    units = u;
    time_set = (t ≡ 0) ? time(0) : t;
    deleted_from_archive = ddeleted_from_archive;
    deleted_from_gwirdsif_db = ddeleted_from_gwirdsif_db;
    return;
} /* End of Irods_AVU_Type non-default constructor definition */
```

1001. Copy constructor. [LDF 2013.06.06.]

Log

[LDF 2013.06.06.] Added this function.

```
< Irods_AVU_Type function declarations 997 > +≡
Irods_AVU_Type(const Irods_AVU_Type &i);
```

1002.

```
< Irods_AVU_Type constructor definitions 998 > +≡
Irods_AVU_Type::Irods_AVU_Type(const Irods_AVU_Type &i)
{
    operator=(i);
    return;
} /* End of Irods_AVU_Type copy constructor definition */
```

1003. Assignment operator. [LDF 2013.06.06.]

Log

[LDF 2013.06.06.] Added this function.

```
< Irods_AVU_Type function declarations 997 > +≡
const Irods_AVU_Type &operator=(const Irods_AVU_Type &i);
```

1004.

```
< Irods_AVU_Type::operator= definition 1004 > ≡
const Irods_AVU_Type &Irods_AVU_Type::operator=(const Irods_AVU_Type &i)
{
    id = i.id;
    irods_object_id = i.irods_object_id;
    user_id = i.user_id;
    attribute = i.attribute;
    value = i.value;
    units = i.units;
    time_set = i.time_set;
    irods_object_path = i.irods_object_path;
    deleted_from_archive = i.deleted_from_archive;
    deleted_from_gwirdsif_db = i.deleted_from_gwirdsif_db;
    return i;
} /* End of Irods_AVU_Type::operator= definition */
```

This code is used in section 1034.

1005. Set. [LDF 2013.03.07.]**Log**

[LDF 2013.03.07.]	Added this function.
[LDF 2013.06.07.]	Added optional argument unsigned int <i>iirods_object_id</i> = 0.
[LDF 2013.07.21.]	Added optional arguments bool <i>ddeleted_from_archive</i> = <i>false</i> and bool <i>ddeleted_from_gwirdsif_db</i> = <i>false</i> .

< Irods_AVU_Type function declarations 997 > +=

```
void set(string attrib, string val, string u = "", unsigned int t = 0, unsigned int
        iirods_object_id = 0, unsigned int iid = 0, string iirods_object_path = "", int uuser_id = 0, bool
        ddeleted_from_archive = false, bool ddeleted_from_gwirdsif_db = false);
```

1006.

```

⟨ Irods_AVU_Type::set definition 1006 ⟩ ≡
void Irods_AVU_Type::set(string attrib, string val, string u, unsigned int t, unsigned
    int irods_object_id, unsigned int iid, string irods_object_path, int uuser_id, bool
    ddeleted_from_archive, bool ddeleted_from_gwirdsif_db)
{
    bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    if (irods_object_id > 0) irods_object_id = irods_object_id;
    if (iid > 0_U) id = iid;
    if (uuser_id > 0) user_id = uuser_id;
    irods_object_path = irods_object_path;
    attribute = attrib;
    value = val;
    units = u;
    time_set = (t == 0) ? time(0) : t;
    deleted_from_archive = ddeleted_from_archive;
    deleted_from_gwirdsif_db = ddeleted_from_gwirdsif_db;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In Irods_AVU_Type::set:" << endl << "deleted_from_archive=" <<
            deleted_from_archive << endl << "deleted_from_gwirdsif_db=" <<
            deleted_from_gwirdsif_db << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return;
} /* End of Irods_AVU_Type::set definition */

```

This code is used in section 1034.

1007. Clear. [LDF 2012.10.12.]

Log

[LDF 2012.10.12.] Added this function.

```

⟨ Irods_AVU_Type function declarations 997 ⟩ +≡
void clear(void);

```

1008.

```

⟨ Irods_AVU_Type::clear definition 1008 ⟩ ≡
void Irods_AVU_Type::clear(void)
{
    id = irods_object_id = 0_U;
    user_id = 0;
    irods_object_path = attribute = value = units = "";
    time_set = 0;
    deleted_from_archive = deleted_from_gwirdsif_db = false;
    return;
}

```

This code is used in section 1034.

1009. Delete iRODS AVU (*delete_irods_avu*). [LDF 2013.07.17.]

Log

[LDF 2013.07.17.] Added this **static** function.

[LDF 2013.08.02.] Added the optional argument `MYSQL * mysql_ptr = 0` and code for deleting the row corresponding to the **Irods_AVU_Type** & *irods_avu* argument from the `gwirdsif.Irods_AVUs` database table.

`< Irods_AVU_Type function declarations 997 > +≡`

```
static int delete_irods_avu(Irods_AVU_Type &irods_avu, bool delete_from_database = true,
    MYSQL * mysql_ptr = 0);
```

1010.

`< Irods_AVU_Type::delete_irods_avu definition 1010 > ≡`

```
int Irods_AVU_Type::delete_irods_avu(Irods_AVU_Type &irods_avu, bool delete_from_database,
    MYSQL * mysql_ptr){ bool DEBUG = false; /* true */
    MYSQL_RES * result = 0;
    MYSQL_ROW curr_row;
    unsigned int row_ctr;
    unsigned int field_ctr;
    long int affected_rows;
    set_debug_level(DEBUG, 0, 0);
    stringstream cmnd_strm;
    stringstream sql_strm;
    int status;
    FILE *fp;
    char buffer[1024];
    memset(buffer, 0, 1024);
    int temp_val = 0;
    int ret_val = 0;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Entering 'Irods_AVU_Type::delete_irods_avu'." << endl;
        irods_avu.show("irods_avu:");
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

See also sections 1011, 1012, 1013, 1014, 1015, 1016, 1017, 1018, 1019, 1020, 1021, and 1022.

This code is used in section 1034.

1011.

```
< Irods_AVU_Type :: delete_irods_avu definition 1010 > +≡
  if (irods_avu.user_id ≤ 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Irods_AVU_Type::delete_irods_avu':"
    endl << "'irods_avu.user_id' == "
    << irods_avu.user_id <<
    " (<= 0)" << endl << "Can't delete iRODS AVU." << endl <<
    "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    return 1;
} /* if (irods_avu.user_id ≤ 0) */
```

1012. Contrary to what the iRODS documentation says, setting the environment variable `clientUserName` does not work! That is, a “rodsadmin” user can’t “alias” as another user. Therefore, we need a `User_Info_Type` object for each the owner of `*this`, so that we have access to the iRODS environment and authorization files for that user.

`global_user_info_map` may contain an entry for the user. If so, we just take the `User_Info_Type` from it. Otherwise, we must call `Scan_Parse_Parameter_Type::get_user`. This requires declaring a `Scan_Parse_Parameter_Type` object, because `Scan_Parse_Parameter_Type::get_user` isn’t `static`. It’s a fairly complex function, so it doesn’t pay to try to make a `static` version or to program a member function of `User_Info_Type` that behaves similarly (I tried this already). [LDF 2013.07.18.]

```

⟨Irods_AVU_Type::delete_irods_avu definition 1010⟩ +≡
User_Info_Type user_info;
pthread_mutex_lock(&global_user_info_map_mutex);
map<int, User_Info_Type>::iterator iter = global_user_info_map.find(irods_avu.user_id);
if (iter == global_user_info_map.end()) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'Irods_AVU_Type::delete_irods_avu':" << endl << "User_info_for_user" <<
            irods_avu.user_id << " not found on " << "'global_user_info_map'." << endl <<
            "Will call 'Scan_Parse_Parameter_Type::get_user'." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
Scan_Parse_Parameter_Type param;
status = param.get_user(irods_avu.user_id, 0, "", &user_info, true);
if (status != 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Irods_AVU_Type::delete_irods_avu':" << endl <<
        "'Scan_Parse_Parameter_Type::get_user' failed, returning" <<
        status << "." << endl << "Can't delete iRODS AVU." << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    pthread_mutex_unlock(&global_user_info_map_mutex);
    return 1;
} /* if (status != 0) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'Irods_AVU_Type::delete_irods_avu':" << endl <<
            "'Scan_Parse_Parameter_Type::get_user' succeeded, returning 0." << endl;
#endif /* DEBUG_COMPILE */
#if 0
    param.show("param:");
    user_info.show("user_info:");
#endif
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
global_user_info_map[irods_avu.user_id] = user_info;
} /* if (iter == global_user_info_map.end()) */
else {

```

```

    user_info = iter->second;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'Irods_AVU_Type::delete_irods_avu':" << endl << "User_info_for_user" <<
            irods_avu.user_id << " found on " << "global_user_info_map'." << endl;
#endif 0
    user_info.show("user_info:");
#endif
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (iter == global_user_info_map.end()) */
pthread_mutex_unlock(&global_user_info_map_mutex);

```

1013.

```

⟨ Irods_AVU_Type::delete_irods_avu definition 1010 ⟩ +≡
cmnd_strm << "export_irodsEnvFile=" << user_info.get_irods_env_filename() << ";" <<
    "a='imeta_rm-d'" << irods_avu.irods_object_path << "\" << "\"" << irods_avu.attribute <<
    "\"\" << irods_avu.value << "\"" << "2>&1;" << "echo$?;echo$a";
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "cmnd_strm.str() == " << cmnd_strm.str() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1014.

```

⟨ Irods_AVU_Type::delete_irods_avu definition 1010 ⟩ +≡
errno = 0;
fp = popen(cmnd_strm.str().c_str(), "r");
if (fp == 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Irods_AVU_Type::delete_irods_avu':" << endl <<
        "'popen' failed, returning NULL." << endl;
    if (errno != 0) cerr << "'errno' == " << errno << endl << strerror(errno) << endl;
    cerr << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    return 1;
} /* if (fp == 0) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "'popen' succeeded." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1015.**Log**

[LDF 2013.08.22.] BUG FIX: Now calling *fread* and *sscanf* instead of *fscanf*. Previously, this sometimes caused a “broken pipe” error in the shell created by *popen*.

```

⟨ Irods_AVU_Type::delete_irods_avu definition 1010 ⟩ +≡
    memset(buffer, 0, 1024);
    status = fread(buffer, 1, 1024, fp);
    if (status == 0 || status == 1024) {
        memset(buffer, 0, 1024);
        lock_cerr_mutex();
        cerr << "ERROR! In 'Irods_AVU_Type::delete_irods_avu':" << endl << "'fread' returned" <<
            status << ".";
        if (status == 0) cerr << "Failed to read error output from 'imeta' command." << endl;
        else cerr << "Too much error output from 'imeta'. This is not permitted." << endl;
        cerr << "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        pclose(fp);
        return 1;
    } /* if (status == 0 || status == 1024) */
#endif /* DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'Irods_AVU_Type::delete_irods_avu':" << endl <<
            "'fread' succeeded, returning" << status << "." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1016.

```

⟨ Irods_AVU_Type::delete_irods_avu definition 1010 ⟩ +≡
    status = sscanf(buffer, "%d", &temp_val);
    if (status != 1) {
        lock_cerr_mutex();
        cerr << "ERROR! In 'Irods_AVU_Type::delete_irods_avu':" << endl <<
            "'sscanf' failed, returning" << status << ":" << endl << "Failed to set 'temp_val'." <<
            endl << "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        pclose(fp);
        return 1;
    } /* if (status == 0 || status == 1024) */
#endif /* DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'Irods_AVU_Type::delete_irods_avu':" << endl <<
            "'sscanf' succeeded, returning 1." << endl << "'temp_val' == " << temp_val << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1017.

```
< Irods_AVU_Type :: delete_irods_avu definition 1010 > +≡
  if (temp_val == 4) {
    lock_cerr_mutex();
    cerr << "Unknown_file._File_was_probably_deleted_using_`irm'." << endl;
    if (strlen(buffer) > 0) cerr << "'imeta'_error_output:" << endl << buffer << endl;
    cerr << "Exiting_function_successfully_with_return_value_2." << endl;
    unlock_cerr_mutex();
    ret_val = 2;
  } /* if (temp_val == 4) */
```

1018.

```
< Irods_AVU_Type :: delete_irods_avu definition 1010 > +≡
  else
    if (temp_val != 0) {
      lock_cerr_mutex();
      cerr << "ERROR! _In_`Irods_AVU_Type::delete_irods_avu':" << endl <<
        "'imeta'_command_failed,_returning_" << temp_val << "(_!=0)";
      if (strlen(buffer) > 0) cerr << "'imeta'_error_output:" << endl << buffer << endl;
      cerr << "Exiting_function_unsuccessfully_with_return_value_1." << endl;
      unlock_cerr_mutex();
      pclose(fp);
      return 1;
    } /* else if (temp_val != 0) */
```

1019.

```
< Irods_AVU_Type :: delete_irods_avu definition 1010 > +≡
  pclose(fp);
  fp = 0;
```

1020.

Log

[LDF 2013.08.02.] Added this section.

```
< Irods_AVU_Type :: delete_irods_avu definition 1010 > +≡
  if (¬delete_from_database) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "In 'Irods_AVU_Type::delete_irods_avu': 'delete_from_database' == 'false'." ≪
      endl ≪ "Skipping to 'END_DELETEIRODS_AVU'." ≪ endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  goto END_DELETEIRODS_AVU;
} /* if (¬delete_from_database) */
else if (irods_avu.id ≤ 0) {
  lock_cerr_mutex();
  cerr ≪ "WARNING! In 'Irods_AVU_Type::delete_irods_avu': 'irods_avu.id' == " ≪
    irods_avu.id ≪ "(<=0)" ≪ endl ≪ "Can't delete row from 'gwiridsif.Irods_AVUs' database table." ≪ endl ≪ "Skipping to 'END_DELETEIRODS_AVU'." ≪ endl;
  unlock_cerr_mutex();
} /* else if (irods_avu.id ≤ 0) */
#endif DEBUG_COMPILE
else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "In 'Irods_AVU_Type::delete_irods_avu': 'delete_from_database' == 'true'." ≪
      endl ≪ "Will delete row from 'gwiridsif.Irods_AVUs' database table." ≪ endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1021.

```
< Irods_AVU_Type::delete_irods_avu definition 1010 > +≡
    sql_strm.str("");
    sql_strm << "delete_from_gwiridsif.Irods_AVUs_where_irods_avu_id_=_" << irods_avu.id;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "sql_strm.str() == " << sql_strm.str() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    status = submit_mysql_query(sql_strm.str(), result, mysql_ptr, 0, 0, &affected_rows);
    if (status != 0) {
        lock_cerr_mutex();
        cerr << "ERROR! In 'Irods_AVU_Type::delete_irods_avu':"
            << endl <<
            "'submit_mysql_query' failed, returning " << status << "."
            << endl <<
            "Failed to delete row(s) from 'gwiridsif.Irods_AVUs' database table."
            << endl <<
            "Exiting function unsuccessfully with return value 1."
            << endl;
        unlock_cerr_mutex();
        if (result) mysql_free_result(result);
        return 1;
    } /* if (status != 0) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'Irods_AVU_Type::delete_irods_avu':"
            << endl <<
            "'submit_mysql_query' succeeded, returning 0."
            << endl <<
            "Deleted row(s) from 'gwiridsif.Irods_AVUs' database table successfully."
            << endl <<
            "'affected_rows' == " << affected_rows << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
mysql_free_result(result);
result = 0;
```

1022.

```

⟨ Irods_AVU_Type::delete_irods_avu definition 1010 ⟩ +≡
END_DELETEIRODS_AVU:
if (ret_val ≠ 0) {
    lock_cerr_mutex();
    cerr << "In ‘Irods_AVU_Type::delete_irods_avu’:"
    << endl <<
    "Exiting function with return value" << ret_val << "."
    << endl;
    unlock_cerr_mutex();
}
#if DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In ‘Irods_AVU_Type::delete_irods_avu’:"
        << endl <<
        "'imeta' command succeeded, returning 0."
        << endl <<
        "Exiting function successfully with return value 0."
        << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
irods_avu.clear();
return ret_val; } /* End of Irods_AVU_Type::delete_irods_avu definition */

```

1023. Show. [LDF 2012.10.12.]

Log

[LDF 2012.10.12.] Added this function.

```

⟨ Irods_AVU_Type function declarations 997 ⟩ +≡
void show(string s = "", stringstream *strm = 0) const;

```

1024.

```

⟨ Irods_AVU_Type::show definition 1024 ⟩ ≡
void Irods_AVU_Type::show(string s, stringstream *strm) const
{
    char buffer[64];
    memset(buffer, 0, 64);
    stringstream temp_strm;
    temp_strm << time_set;
    struct tm tm;
    if (strptime(temp_strm.str().c_str(), "%s", &tm)) {
        strftime(buffer, 64, "%Y-%m-%d %H:%M:%S %Z", &tm);
    }
    temp_strm.str("");
    if (s.empty()) s = "Irods_AVU_Type:";
    temp_strm << s << endl << "id==XXXXXXXXXXXXXX" << id <<
        endl << "irods_object_id==XXXXXXXXXXXXXX" << irods_object_id << endl <<
        "user_id==XXXXXXXXXXXXXX" << user_id << endl << "irods_object_path==XXXXXXXXXXXXXX" <<
        irods_object_path;
    if (irods_object_path.empty()) temp_strm << "(empty)";
    temp_strm << endl << "attribute==XXXXXXXXXXXXXX" << attribute << endl <<
        "value==XXXXXXXXXXXXXX" << value;
    if (value.empty()) temp_strm << "(empty)";
    temp_strm << endl << "units==XXXXXXXXXXXXXX" << units;
    if (units.empty()) temp_strm << "(empty)";
    temp_strm << endl << "time_set==XXXXXXXXXXXXXX" << time_set << " (seconds since epoch)";
    if (strlen(buffer) > 0) temp_strm << ":" << buffer;
    temp_strm << endl << "deleted_from_archive==XXXXXX" << deleted_from_archive << endl <<
        "deleted_from_gwirdsif_db==XXXXXX" << deleted_from_gwirdsif_db << endl << endl;
    if (*strm) *strm << temp_strm.str();
    else cerr << temp_strm.str();
    return;
} /* End of Irods_AVU_Type::show definition */

```

This code is used in section 1034.

1025. Write to database (*write_to_database*). [LDF 2013.06.06.]

```

⟨ Irods_AVU_Type function declarations 997 ⟩ +≡
int write_to_database(MYSQL *mysql_ptr, int thread_ctr = 0);

```

1026.

```
< Irods_AVU_Type :: write_to_database definition 1026 > ≡
int Irods_AVU_Type :: write_to_database(MYSQL *mysql_ptr, int thread_ctr){ bool DEBUG = false;
    /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    string thread_str;
    stringstream temp_strm;
    if (thread_ctr > 0) {
        temp_strm << "[Thread" << thread_ctr << "] ";
        thread_str = temp_strm.str();
        temp_strm.str("");
    }
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering 'Irods_AVU_Type::write_to_database'." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

See also sections 1027, 1028, 1029, 1030, 1031, 1032, and 1033.

This code is used in section 1034.

1027.

```
< Irods_AVU_Type :: write_to_database definition 1026 > +≡
  MYSQL_RES * result = 0;
  MYSQL_ROW curr_row;
  unsigned int row_ctr;
  unsigned int field_ctr;
  long int affected_rows;
  stringstream sql_strm;
  int ret_val = 0;
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    show();
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  if (irods_object_id == 0) {
    lock_cerr_mutex();
    cerr << thread_str << "WARNING! In 'Irods_AVU_Type::write_to_database':" <<
      endl << "'irods_object_id' == 0. Not writing 'Irods_AVU_Type' to "
      "'gwrdsif.Irods_AVUs' database table." << endl <<
      "Exiting function unsuccessfully with return value 3." << endl;
    unlock_cerr_mutex();
    return 3;
  } /* if (irods_object_id == 0) */
#endif DEBUG_COMPILE
  else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Irods_AVU_Type::write_to_database':" << endl <<
      "'irods_object_id'" << irods_object_id << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1028.

```
< Irods_AVU_Type :: write_to_database definition 1026 > +≡
  status = submit_mysql_query("lock_tables_gwirdsif.Irods_AVUs_write", result, mysql_ptr);
  if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "ERROR! In 'Irods_AVU_Type::write_to_database':"
    endl ≪ "'submit_mysql_query' failed, returning" ≪ status ≪ "."
    endl ≪ "Failed to lock 'gwirdsif.Irods_AVUs' table."
    endl ≪ "Exiting function unsuccessfully with return value 1."
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    return 1;
  } /* if (irods_object.id == 0) */
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr ≪ thread_str ≪ "In 'Irods_AVU_Type::write_to_database':"
      endl ≪ "'submit_mysql_query' succeeded, returning 0."
      endl ≪ "Locked 'gwirdsif.Irods_AVUs' table successfully."
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1029.

```
< Irods_AVU_Type::write_to_database definition 1026 > +≡
status = Scan_Parse_Parameter_Type::get_highest_value(mysql_ptr, "gwiridsif.Irods_AVUs",
    "irods_avu_id", id, true);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "ERROR! In 'Irods_AVU_Type::write_to_database':"
    ≪ endl ≪
        "Scan_Parse_Parameter_Type::get_highest_value' failed, returning "
        ≪ status ≪
        "."
    ≪ endl ≪ "Failed to retrieve highest value of 'irods_avu_id' from the "
    ≪
        "gwiridsif.Irods_AVUs' database table."
    ≪ endl ≪
        "Will try to unlock 'gwiridsif.Irods_AVUs' and exit unsuccessfully with return "
    ≪
        "value 1."
    ≪ endl;
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    id = 0;
    ret_val = 1;
    goto UNLOCK_TABLES;
} /* if (irods_object_id ≡ 0) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ thread_str ≪ "In 'Irods_AVU_Type::write_to_database':"
        ≪ endl ≪
            "'submit_mysql_query' succeeded, returning 0."
        ≪ endl ≪
            "Retrieved highest value of 'irods_avu_id' from the "
        ≪
            "'gwiridsif.Irods_AVUs' database successfully."
        ≪ endl ≪
            "'id' == "
        ≪ id ≪
        endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
mysql_free_result(result);
result = 0;
```

1030.

```

⟨ Irods_AVU_Type::write_to_database definition 1026 ⟩ +≡
  sql_strm << "insert into gwirldsif.Irods_AVUs(irods_avu_id," <<
    "irods_object_id,attribute,value,units,time_set)" << "values(" <<
    "id << ", " << irods_object_id << ", " << " << attribute << ", " << value << ", " << " <<
    "units << ", " << "from_unixtime(" << time_set << "))";
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In `Irods_AVU_Type::write_to_database':" << endl <<
      "'sql_strm.str()' == " << sql_strm.str() << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  status = submit_mysql_query(sql_strm.str(), result, mysql_ptr, 0, 0, &affected_rows);
  if (status != 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In `Irods_AVU_Type::write_to_database':" <<
      endl << "'submit_mysql_query' failed, returning " << status << "." << endl <<
      "Failed to insert row into `gwirldsif.Irods_AVUs` table." << endl <<
      "Will try to unlock tables before exiting function unsuccessfully" <<
      "with return value 1." << endl;
    unlock_cerr_mutex();
    ret_val = 1;
    if (result) {
      mysql_free_result(result);
      result = 0;
    }
    goto UNLOCK_TABLES;
  } /* if (irods_object_id == 0) */
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << thread_str << "In `Irods_AVU_Type::write_to_database':" <<
        endl << "'submit_mysql_query' succeeded, returning 0." << endl <<
        "Inserted row into `gwirldsif.Irods_AVUs` table successfully." << endl <<
        "'affected_rows' == " << affected_rows << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  mysql_free_result(result);
  result = 0;

```

1031.

```

⟨ Irods_AVU_Type::write_to_database definition 1026 ⟩ +≡
UNLOCK_TABLES: status = submit_mysql_query("unlock_tables", result, mysql_ptr);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Irods_AVU_Type::write_to_database':"
    endl << "'submit_mysql_query' failed, returning" << status << "."
    endl << "Failed to unlock 'gwiridsif.Irods_AVUs' table."
    endl << "Exiting function unsuccessfully with return value 1."
    endl;
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    return 1;
} /* if (irods_object.id == 0) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Irods_AVU_Type::write_to_database':"
        endl << "'submit_mysql_query' succeeded, returning 0."
        endl << "Unlocked 'gwiridsif.Irods_AVUs' table successfully."
        endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1032.

```

⟨ Irods_AVU_Type::write_to_database definition 1026 ⟩ +≡
if (ret_val ≠ 0) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Irods_AVU_Type::write_to_database':"
    endl << "Exiting function unsuccessfully with return value 'ret_val' == "
    << ret_val << "."
    endl;
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    return ret_val;
} /* if (ret_val ≠ 0) */

```

1033.

```

⟨ Irods_AVU_Type::write_to_database definition 1026 ⟩ +≡
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "Exiting 'Irods_AVU_Type::write_to_database' successfully with"
    endl << "return value 0."
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
return 0; /* End of Irods_AVU_Type::write_to_database definition */

```

1034. Putting `iRODS_AVU_Type` together. [LDF 2013.01.10.]

The header file `irdsavtp.h++` generated from this CWEB file is included in `<Include files 3>`, so `<Declare class Irods_AVU_Type 996>` is not inserted here. [LDF 2013.06.06.]

```
<Include files 3>
using namespace std;
using namespace gwrdifpk;
class Irods_Object_Type;
<Irods_AVU_Type constructor definitions 998>
<Irods_AVU_Type::operator= definition 1004>
<Irods_AVU_Type::set definition 1006>
<Irods_AVU_Type::clear definition 1008>
<Irods_AVU_Type::delete_irods_avu definition 1010>
<Irods_AVU_Type::show definition 1024>
<Irods_AVU_Type::write_to_database definition 1026>
```

1035. This is what's written to the header file `irdsavtp.h`. [LDF 2008.12.05.]

```
<irdsavtp.h 1035>≡
#ifndef IRDSAVTP_H
#define IRDSAVTP_H 1
using namespace std;
using namespace gwrdifpk;
class Irods_Object_Type;
<Declare class Irods_AVU_Type 996>
#endif
```

1036. `Irods_Object_Type` (`irdsobtp.web`).

Log

[LDF 2012.10.11.] Added this file.

1037. Include files.

```
<Include files 3> +≡
#ifndef _XOPEN_SOURCE
#define _XOPEN_SOURCE
#endif
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <unistd.h>
#include <limits.h>
#include <time.h>
#if 0
#include <sys/time.h>
#include <sys/stat.h>
#endif
#include <sys/types.h>
#if 0
#include <dirent.h>
#endif
#include <string.h>
#if 0
#include <pwd.h>
#endif
#include <errno.h>
#include <pthread.h>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <string>
#include <sstream>
#include <deque>
#include <map>
#include <set>
#include <vector>
#include <stack>
#include <gcrypt.h> /* for gcry_control */
#include <gnutls/gnutls.h>
#include <gnutls/x509.h>
#include <mysql.h>
#include <expat.h>
#if HAVE_CONFIG_H
#include <config.h>
#endif
#undef NAME_LEN
#undef LOCAL_HOST
#include "glblcnst.h++"
#include "glblvrbl.h++"
#include "excptntp.h++"
#include "utilfncts.h++"
#include "parser.h++"
#include "scanner.h++"
```

```
#include "grouppt.h++"
#include "hndlvtpp.h++"
#include "rspnstp.h++"
#include "irdsavtp.h++"
#include "irdsobtp.h++"
#include "hndltype.h++"
#include "dcmtddtp.h++"
#include "x509cert.h++"
#include "dstngnmt.h++"
#include "scprpmtp.h++"
#include "usrinftp.h++"
```

1038. class Irods_Object_Type.

Log

[LDF 2012.10.11.] Added this **class** declaration.
 [LDF 2013.01.07.] Changed **string** *name* to **string** *path*. Added the data members **bool** *deleted*, **time_t** *created*, **time_t** *last_modified*, **string** *created_str* and **string** *last_modified_str*.
 [LDF 2013.01.08.] Added data members **vector***<unsigned long int>* *handle_id_vector* and **vector***<unsigned long int>* *handle_value_id_vector*.
 [LDF 2013.01.31.] Added **vector***<string>* *handle_name_string_vector*.
 [LDF 2013.03.07.] Changed **vector***<Handle_Value_Type>* *handle_value_vector* to **vector***<Handle_Type>* *handle_vector*.
 [LDF 2013.07.21.] Replaced **bool** *deleted* with **bool** *deleted_from_archive* and **bool** *deleted_from_gwirdsif_db*.
 [LDF 2013.08.07.] Added the private data members **bool** *marked_for_deletion_from_archive* and **bool** *marked_for_deletion_from_gwirdsif_db*.
 [LDF 2013.08.07.] Removed **string** *created_str* and **string** *last_modified_str*.
 [LDF 2013.08.08.] Added the private data members **time_t** *delete_from_archive_timestamp* and **time_t** *delete_from_gwirdsif_db_timestamp*.
 [LDF 2013.08.08.] Added **friend** declaration for *purge_irods_archive*.

```
{ Declare class Irods_Object_Type 1038 } ≡
class Irods_Object_Type {
    friend class Scan_Parse_Parameter_Type;
    friend int zzparse(yyscan_t);
    friend void *purge_irods_archive(void *v);
    unsigned long int id;
    int user_id;
    int irods_server_id;
    string path;
    bool marked_for_deletion_from_archive;
    bool marked_for_deletion_from_gwirdsif_db;
    bool deleted_from_archive;
    bool deleted_from_gwirdsif_db;
    time_t delete_from_archive_timestamp;
    time_t delete_from_gwirdsif_db_timestamp;
    time_t created;
    time_t last_modified;
    vector<Handle_Type> handle_vector;
    vector<Irods_AVU_Type> avu_vector;
    vector<unsigned long int> handle_id_vector;
    vector<unsigned long int> handle_value_id_vector;
```

```

vector<string> handle_name_string_vector;
public: <Irods_Object_Type function declarations 1039>
};

```

This code is used in section 1259.

1039. Default constructor. [LDF 2012.10.11.]

Log

[LDF 2012.10.11.] Added this function.

```

<Irods_Object_Type function declarations 1039> ≡
Irods_Object_Type(void);

```

See also sections 1041, 1043, 1045, 1058, 1060, 1062, 1076, 1119, 1132, 1140, 1149, 1153, 1173, 1205, 1223, 1229, and 1231.

This code is used in section 1038.

1040.

```

<Irods_Object_Type constructor definitions 1040> ≡
Irods_Object_Type::Irods_Object_Type(void)
{
    id = 0;
    user_id = 0;
    irods_server_id = 0;
    marked_for_deletion_from_archive = marked_for_deletion_from_gwiridsif_db = deleted_from_archive =
        deleted_from_gwiridsif_db = false;
    delete_from_archive_timestamp = delete_from_gwiridsif_db_timestamp = created = last_modified =
        static_cast<time_t>(0);
}

```

See also section 1042.

This code is used in section 1258.

1041. Non-default constructor. [LDF 2013.01.07.]

Log

[LDF 2013.01.07.] Added this function.

```

<Irods_Object_Type function declarations 1039> +≡
Irods_Object_Type(unsigned int uuser_id, string ppath);

```

1042.

```

⟨ Irods_Object_Type constructor definitions 1040 ⟩ +≡
Irods_Object_Type::Irods_Object_Type(unsigned int uuser_id, string ppath)
{
    bool DEBUG = false;      /* true */
    set_debug_level(DEBUG, 0, 0);
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Entering 'Irods_Object_Type' constructor (non-default)." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    user_id = uuser_id;
    path = ppath;
    id = 0;
    irods_server_id = 1;
    marked_for_deletion_from_archive = marked_for_deletion_from_gwirdsif_db = deleted_from_archive =
        deleted_from_gwirdsif_db = false;
    delete_from_archive_timestamp = delete_from_gwirdsif_db_timestamp = last_modified =
        static_cast<time_t>(0);

    char outstr[200];
    struct tm tmp;
    errno = 0;
    created = time(0);
    if (created == static_cast<time_t>(-1)) {
        lock_cerr_mutex();
        cerr << "ERROR! In 'Irods_Object_Type' constructor: 'time' failed, "
            "returning -1." << endl << "Error: " << strerror(errno) << endl <<
            "Exiting function unsuccessfully." << "Throwing 'Initialize_Exception_Type'." <<
            endl;
        unlock_cerr_mutex();
        throw Initialize_Exception_Type();
    } /* if */
    if ((gmtime_r(&created, &tmp)) == 0) {
        lock_cerr_mutex();
        cerr << "ERROR! In 'Irods_Object_Type' constructor: 'gmtime_r' failed, "
            "returning 0." << endl << "Error: " << strerror(errno) << endl <<
            "Exiting function unsuccessfully." << "Throwing 'Initialize_Exception_Type'." <<
            endl;
        unlock_cerr_mutex();
        throw Initialize_Exception_Type();
    }
    if (strftime(outstr, sizeof(outstr), "%Y-%m-%d %H:%M:%S UTC", &tmp) == 0) {
        lock_cerr_mutex();
        cerr << "ERROR! In 'Irods_Object_Type' constructor: 'strftime' failed, "
            "returning 0." << endl << "Exiting function unsuccessfully." <<
            "Throwing 'Initialize_Exception_Type'." << endl;
        unlock_cerr_mutex();
        throw Initialize_Exception_Type();
    }
}

```

```
#if DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Exiting 'Irods_Object_Type' constructor successfully." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return;
} /* End of non-default Irods_Object_Type constructor definition */
```

1043. Setting function. [LDF 2013.01.08.]

Log

[LDF 2013.01.08.] Added this function.

⟨ Irods_Object_Type function declarations 1039 ⟩ +≡
int set(unsigned int uuser_id, string ppath);

1044.

```

⟨ Irods_Object_Type::set definitions 1044 ⟩ ≡
int Irods_Object_Type::set(unsigned int user_id, string ppath)
{
    bool DEBUG = false;      /* true */
    set_debug_level(DEBUG, 0, 0);
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Entering Irods_Object_Type::set" . " << endl;
        unlock_cerr_mutex();
    }      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */
    user_id = user_id;
    path = ppath;
    id = 0;
    irods_server_id = 1;
    marked_for_deletion_from_archive = marked_for_deletion_from_gwirdsif_db = deleted_from_archive =
        deleted_from_gwirdsif_db = false;
    delete_from_archive_timestamp = delete_from_gwirdsif_db_timestamp = last_modified =
        static_cast<time_t>(0);
    handle_vector.clear();
    avu_vector.clear();
    handle_id_vector.clear();
    handle_value_id_vector.clear();
    handle_name_string_vector.clear();
    char outstr[200];
    struct tm tmp;
    errno = 0;
    created = time(0);
    if (created ≡ static_cast<time_t>(-1)) {
        lock_cerr_mutex();
        cerr << "ERROR! In 'Irods_Object_Type::set': 'time' failed, "
            "returning '(time_t)-1'" << endl << "Error:" << strerror(errno) << endl <<
            "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        return 1;
    }      /* if */
    if ((gmtime_r(&created, &tmp)) ≡ 0) {
        lock_cerr_mutex();
        cerr << "ERROR! In 'Irods_Object_Type::set': 'gmtime_r' failed, "
            "returning 0." << endl << "Error:" << strerror(errno) << endl <<
            "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        return 1;
    }
    if (strftime(outstr, sizeof(outstr), "%Y-%m-%d %H:%M:%S UTC", &tmp) ≡ 0) {
        lock_cerr_mutex();
        cerr << "ERROR! In 'Irods_Object_Type::set': 'strftime' failed, "
            "returning 0." << endl << "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
    }
}

```

```

        return 1;
    }
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Exiting 'Irods_Object_Type::set' successfully with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0;
} /* End of Irods_Object_Type::set definition */

```

This code is used in section 1258.

1045. *get_avus_from_irods_system*. [LDF 2012.10.11.]

Log

[LDF 2012.10.11.] Added this function.

[LDF 2012.10.12.] Replaced argument **FILE** *fp with **string** command. Now opening pipe in this function instead of **Scan_Parse_Parameter_Type**::*get_metadata*.

⟨Irods_Object_Type function declarations 1039⟩ +≡

```
int get_avus_from_irods_system(string command, string filename, Scan_Parse_Parameter_Type
&param, int *ctr = 0);
```

1046.

⟨Irods_Object_Type::*get_avus_from_irods_system* definition 1046⟩ ≡

```
int Irods_Object_Type::get_avus_from_irods_system(string command, string filename,
Scan_Parse_Parameter_Type &param, int *ctr){ bool DEBUG = false; /* true */
set_debug_level(DEBUG, 0, 0);
int status;
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "Entering 'Irods_Object_Type::get_avus_from_irods_system'." << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

See also sections 1047, 1048, 1049, 1050, 1051, 1052, 1053, 1054, 1055, 1056, and 1057.

This code is used in section 1258.

1047.

```
< Irods_Object_Type::get_avus_from_irods_system definition 1046 > +≡
    stringstream temp_strm;
    char buffer[4][512]; /* 29 */
    memset(buffer, 0, 512);
    char *temp_ptr = 0;
    bool first_time = true;
    string temp_str;
    size_t pos;
    Irods_AVU_Type avu;
    struct tm tm;
```

1048.

```
< Irods_Object_Type::get_avus_from_irods_system definition 1046 > +≡
FILE *fp;
fp = popen(command.c_str(), "r");
if (fp == 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Irods_Object_Type::get_avus_from_irods_system':" << endl <<
        "'popen' failed, returning NULL." << endl << "Failed to execute 'imeta' command." <<
        endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    return 1;
} /* if (fp == 0) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'Irods_Object_Type::get_avus_from_irods_system':" << endl <<
            "'popen' succeeded." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1049.

```
< Irods_Object_Type::get_avus_from_irods_system definition 1046 > +≡
do { avu.clear(); for (int i = 0; i < 4; ++i) { memset(buffer[i], 0, 512);
errno = 0;
temp_ptr = fgets(buffer[i], 512, fp);
if (temp_ptr == 0 & first_time) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Irods_Object_Type::get_avus_from_irods_system':" <<
        endl << "'fgets' failed, returning 0:" << endl << "Error: " <<
        strerror(errno) << endl << "Failed to read output of 'imeta' command." << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    pclose(fp);
    fp = 0;
    return 1;
} /* if */
```

1050.

```
< Irods_Object_Type::get_avus_from_irods_system definition 1046 > +≡
else
    if (strlen(buffer[i]) ≡ 512 - 1) {
        lock_cerr_mutex();
        cerr ≡ "ERROR! In 'Irods_Object_Type::get_avus_from_irods_system':"
            ≡ endl ≡
            " fgets' read "
            ≡ (512 - 1) ≡ " characters. Output of 'imeta' command "
            ≡ " exceeds the maximum permitted length "
            ≡ (512 - 2) ≡ " (512-2) characters ). "
            ≡ endl ≡ " Exiting function unsuccessfully with return value 1. "
            ≡ endl;
        unlock_cerr_mutex();
        pclose(fp);
        fp = 0;
        return 1;
    } /* else if */
```

1051.

```
< Irods_Object_Type::get_avus_from_irods_system definition 1046 > +≡
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≡ " In 'Irods_Object_Type::get_avus_from_irods_system':"
            ≡ endl ≡
            " fgets' succeeded, reading "
            ≡ strlen(buffer[i]) ≡ " characters. "
            ≡ endl ≡
            " buffer["
            ≡ i ≡ "] == "
            ≡ endl ≡ buffer[i] ≡ endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
first_time = false;
temp_str = buffer[i];
```

1052.

```

⟨ Irods_Object_Type::get_avus_from_irods_system definition 1046 ⟩ +≡
    if (i == 0)      /* “attribute” line */
    {
        pos = temp_str.find("attribute");
        if (pos != 0) {
#ifndef DEBUG_COMPILE
            if (DEBUG) {
                lock_cerr_mutex();
                cerr << "'i' == 0 and not \"attribute\" line." << endl << "Skipping." << endl;
                unlock_cerr_mutex();
            }      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */
            break;
        }      /* if (pos != 0) */
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "Found \"attribute\" line:" << buffer[i] << endl;
            unlock_cerr_mutex();
        }      /* else if (DEBUG) */
#endif      /* DEBUG_COMPILE */
    avu.attribute = temp_str;
    avu.attribute.erase(0, strlen("attribute:"));
    if (avu.attribute[avu.attribute.length() - 1] == '\n') avu.attribute.erase(avu.attribute.length() - 1);
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "'avu.attribute'" << avu.attribute << '"' << endl;
        unlock_cerr_mutex();
    }      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */
}      /* if (i == 0) (“attribute” line） */

```

1053.

```
< Irods_Object_Type::get_avus_from_irods_system definition 1046 > +≡
else
  if (i ≡ 1) /* "value" line */
  {
    pos = temp_str.find("value");
    if (pos ≠ 0) {
#if DEBUG_COMPILE
      if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ "'i' == 1 and not \"value\" line." ≪ endl ≪ "Skipping." ≪ endl;
        unlock_cerr_mutex();
      } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
      break;
    } /* if (pos ≠ 0) */
#endif DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr ≪ "Found \"value\" line:" ≪ buffer[i] ≪ endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  avu.value = temp_str;
  avu.value.erase(0, strlen("value:"));
  if (avu.value[avu.value.length() - 1] ≡ '\n') avu.value.erase(avu.value.length() - 1);
#endif DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "'avu.value' == " ≪ avu.value ≪ "," ≪ endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* else if (i ≡ 1) ("value" line) */
```

1054.

```

⟨ Irods_Object_Type::get_avus_from_irods_system definition 1046 ⟩ +≡
else
  if (i ≡ 2) /* “units” line */
  {
    pos = temp_str.find("units");
    if (pos ≠ 0) {
#if DEBUG_COMPILE
      if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ "‘i’==2andnot\“units\“line." ≪ endl ≪ "Skipping." ≪ endl;
        unlock_cerr_mutex();
      } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
      break;
    } /* if (pos ≠ 0) */
#endif DEBUG_COMPILE
    else
      if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ "Found\“units\“line:\“" ≪ buffer[i] ≪ endl;
        unlock_cerr_mutex();
      } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    avu.units = temp_str;
    avu.units.erase(0, strlen("units:\“"));
    if (avu.units[avu.units.length() - 1] ≡ '\n') avu.units.erase(avu.units.length() - 1);
#endif DEBUG_COMPILE
    if (DEBUG) {
      lock_cerr_mutex();
      cerr ≪ "‘avu.units’==\“" ≪ avu.units ≪ ",\“" ≪ endl;
      unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  } /* else if (i ≡ 2) (“units” line) */

```

1055.

```

⟨ Irods_Object_Type::get_avus_from_irods_system definition 1046 ⟩ +≡
else
  if (i ≡ 3) /* "time_set" line */
  {
    pos = temp_str.find("time_set");
    if (pos ≠ 0) {
#if DEBUG_COMPILE
      if (DEBUG) {
        lock_cerr_mutex();
        cerr << "'i'==3 and not \"time_set\"\\line." << endl << "Skipping." << endl;
        unlock_cerr_mutex();
      } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
      break;
    } /* if (pos ≠ 0) */
#endif /* DEBUG_COMPILE */
    else
      if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Found\\\"time_set\\\"\\line:\\\" << buffer[i] << endl;
        unlock_cerr_mutex();
      } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    temp_str.erase(0, strlen("time_set:\\"));
    if (temp_str[temp_str.length() - 1] ≡ '\n') temp_str.erase(temp_str.length() - 1);
    if (strptime(temp_str.c_str(), "%Y-%m-%d.%H:%M:%S", &tm))
      avu.time_set = static_cast<unsigned int>(mktime(&tm));
#endif /* DEBUG_COMPILE */
      if (DEBUG) {
        lock_cerr_mutex();
        cerr << "temp_str==" << temp_str << endl << "'avu.time_set'" << avu.time_set <<
          endl;
        unlock_cerr_mutex();
      } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
      avu_vector.push_back(avu);
    } /* else if (i ≡ 3) ("time_set" line) */
  } /* for */
} /* do */
while (temp_ptr ∧ ¬feof(fp) ∧ ¬ferror(fp)) ;

```

1056.

```
< Irods_Object_Type::get_avus_from_irods_system definition 1046 > +≡
    pclose(fp);
    fp = 0;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'Irods_Object_Type::get_avus': " << "avu_vector.size() == " <<
            avu_vector.size() << endl;
#endif 0
    for (vector<Irods_AVU_Type>::const_iterator iter = avu_vector.begin(); iter != avu_vector.end();
        ++iter) {
        iter->show();
    }
#endif
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
if (ctr) *ctr = avu_vector.size();
```

1057.

```
< Irods_Object_Type::get_avus_from_irods_system definition 1046 > +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Exiting 'Irods_Object_Type::get_avus_from_irods_system' successfully" <<
            "with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
return 0; } /* End of Irods_Object_Type::get_avus_from_irods_system definition */
```

1058. Clear. [LDF 2012.10.12.]

Log

[LDF 2012.10.12.] Added this function.

```
< Irods_Object_Type function declarations 1039 > +≡
void clear(void);
```

1059.

```
< Irods_Object_Type :: clear definition 1059 > ≡
void Irods_Object_Type :: clear( void )
{
    id = 0;
    user_id = 0;
    irods_server_id = 0;
    marked_for_deletion_from_archive = marked_for_deletion_from_gwirdsif_db = deleted_from_archive =
        deleted_from_gwirdsif_db = false;
    delete_from_archive_timestamp = delete_from_gwirdsif_db_timestamp = created = last_modified =
        static_cast<time_t>(0);
    path = "";
    handle_vector.clear();
    avu_vector.clear();
    handle_id_vector.clear();
    handle_value_id_vector.clear();
    handle_name_string_vector.clear();
    return;
} /* End of Irods_Object_Type :: clear definition */
```

This code is used in section 1258.

1060. Show. [LDF 2012.10.12.]

Log

[LDF 2012.10.12.] Added this function.
[LDF 2013.03.14.] Made this function **const**.

```
< Irods_Object_Type function declarations 1039 > +≡
void show(string s = "", stringstream *strm = 0) const;
```

1061.

```

⟨ Irods_Object_Type :: show definition 1061 ⟩ ≡
void Irods_Object_Type :: show(string s, stringstream *strm) const
{
    stringstream temp_strm;
    if (s.empty()) s = "Irods_Object_Type:";

    temp_strm << s << endl << "id==ooooooooooooooo" <<
        id << endl << "path==ooooooooooooooo" <<
        path << endl << "user_id==ooooooooooooooo" <<
        user_id << endl << "irods_server_id==ooooooooooooo" <<
        irods_server_id << endl << "marked_for_deletion_from_archive==uuuu" <<
        marked_for_deletion_from_archive << endl << "deleted_from_archive==ooooooooooooo" <<
        deleted_from_archive << endl << "delete_from_archive_timestamp==oooooooooo" <<
        delete_from_archive_timestamp << "u" << (delete_from_archive_timestamp >
        0 ? convert_seconds(delete_from_archive_timestamp) : "") << endl <<
        "marked_for_deletion_from_gwirdsif_db==u" << marked_for_deletion_from_gwirdsif_db <<
        endl << "deleted_from_gwirdsif_db==ooooooooooooo" << deleted_from_gwirdsif_db << endl <<
        "delete_from_gwirdsif_db_timestamp==uuu" << delete_from_gwirdsif_db_timestamp << "u" <<
        (delete_from_gwirdsif_db_timestamp > 0 ? convert_seconds(delete_from_gwirdsif_db_timestamp) :
        "") << endl << "created==ooooooooooooooo" << created << "u" << (created >
        0 ? convert_seconds(created) : "") << endl << "last_modified==ooooooooooooooo" <<
        last_modified << "u" << (last_modified > 0 ? convert_seconds(last_modified) : "") << endl;

    if (avu_vector.size() > 0) {
        temp_strm << "avu_vector.size()==u" << avu_vector.size() << endl << "Showing_avu_vector:" <<
            endl;
        for (vector<Irods_AVU_Type>::const_iterator iter = avu_vector.begin(); iter != avu_vector.end();
            ++iter) {
            iter->show("", &temp_strm);
        }
        temp_strm << endl;
    }
    else temp_strm << "avu_vector is empty." << endl;
    if (handle_vector.size() > 0) {
        temp_strm << "handle_vector.size()==u" << handle_vector.size() << endl <<
            "Showing_handle_vector:" << endl;
        for (vector<Handle_Type>::const_iterator iter = handle_vector.begin(); iter != handle_vector.end();
            ++iter) {
            iter->show("", &temp_strm);
        }
        temp_strm << endl;
    }
    else temp_strm << "handle_vector is empty." << endl;
    if (handle_id_vector.size() > 0) {
        temp_strm << "handle_id_vector.size()==u" << handle_id_vector.size() << endl <<
            "Showing_handle_id_vector:" << endl;
        for (vector<unsigned long int>::const_iterator iter = handle_id_vector.begin();
            iter != handle_id_vector.end(); ++iter) {
            temp_strm << *iter << endl;
        }
        temp_strm << endl;
    }
    else temp_strm << "handle_id_vector is empty." << endl;
}

```

```

if (handle_value_id_vector.size() > 0) {
    temp_strm << "handle_value_id_vector.size() == " << handle_value_id_vector.size() << endl <<
        "Showing handle_value_id_vector:" << endl;
    for (vector<unsigned long int>::const_iterator iter = handle_value_id_vector.begin();
        iter != handle_value_id_vector.end(); ++iter) {
        temp_strm << *iter << endl;
    }
    temp_strm << endl;
}
else temp_strm << "handle_value_id_vector is empty." << endl;
temp_strm << endl;
if (handle_name_string_vector.size() > 0) {
    temp_strm << "handle_name_string_vector.size() == " << handle_name_string_vector.size() <<
        endl << "Showing handle_name_string_vector:" << endl;
    for (vector<string>::const_iterator iter = handle_name_string_vector.begin();
        iter != handle_name_string_vector.end(); ++iter) {
        temp_strm << *iter << endl;
    }
    temp_strm << endl;
}
else temp_strm << "handle_name_string_vector is empty." << endl;
temp_strm << endl;
if (strm) *strm << temp_strm.str();
else cerr << temp_strm.str();
return;
} /* End of Irods_Object_Type::show */

```

This code is used in section 1258.

1062. Write to database (*write_to_database*). [LDF 2013.01.07.]

Log

[LDF 2013.01.07.] Added this function.

⟨ Irods_Object_Type function declarations 1039 ⟩ +≡
int write_to_database(MYSQL *mysql_ptr);

1063.

⟨ Irods_Object_Type::write_to_database definition 1063 ⟩ ≡
int Irods_Object_Type::write_to_database(MYSQL *mysql_ptr){ **bool** DEBUG = *false*; /* true */
 set_debug_level(DEBUG, 0, 0);
int status;
#if DEBUG_COMPILE
if (DEBUG) {
 lock_cerr_mutex();
 cerr << "Entering 'Irods_Object_Type::write_to_database'." << endl;
 unlock_cerr_mutex();
 } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

See also sections 1064, 1065, 1066, 1067, 1068, 1069, 1070, 1071, 1072, 1073, 1074, and 1075.

This code is used in section 1258.

1064.

```
⟨ Irods_Object_Type::write_to_database definition 1063 ⟩ +≡
    MYSQL_RES * result = 0;
    MYSQL_ROW curr_row;
    unsigned int row_ctr;
    unsigned int field_ctr;
    long int affected_rows;
    stringstream sql_strm;
    int ret_val = 0;
    stringstream temp_strm;
    string temp_str;
```

1065.

```
⟨ Irods_Object_Type::write_to_database definition 1063 ⟩ +≡
    status = mysql_select_db(mysql_ptr, "gwiridsif");
    if (status ≡ 0) {
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "In 'Irods_Object_Type::write_to_database': " << "'mysql_select_db' succeeded'" << endl << "Selected 'gwiridsif' database successfully." << endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    } /* if (status ≡ 0) */
    else /* status ≠ 0 */
    {
        lock_cerr_mutex();
        cerr << "In 'Irods_Object_Type::write_to_database': " <<
            "'mysql_select_db' failed, returning " << status << endl <<
            "Failed to select 'gwiridsif' database:" << endl << "Error: " << mysql_error(mysql_ptr) <<
            endl << "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        return 1;
    } /* else (status ≠ 0) */
```

1066.

```
< Irods_Object_Type::write_to_database definition 1063 > +≡
temp_str = "lock_tables_Irods_Objects_write,Irods_AVUs_write,Irods_Objects_Handles_write";
status = submit_mysql_query(temp_str, result, mysql_ptr);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ "ERROR! In 'Irods_Object_Type::write_to_database':"
    ≪ endl ≪ "'submit_mysql_query' failed, returning "
    ≪ status ≪ "."
    ≪ endl ≪ "Failed to lock 'Irods_Objects' and 'Irods_Objects_Handles' tables."
    ≪ endl ≪ "Exiting function unsuccessfully with return value 1."
    ≪ endl;
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    return 1;
} /* if (status ≠ 0) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ "In 'Irods_Object_Type::write_to_database':"
        ≪ endl ≪ "'submit_mysql_query' succeeded, returning 0."
        ≪ endl ≪ "Locked 'Irods_Objects', 'Irods_AVUs' and 'Irods_Objects_Handles' "
        ≪ "tables successfully."
        ≪ endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
temp_str = "";
mysql_free_result(result);
result = 0;
sql_strm.str("");
```

1067.

Log

[LDF 2013.06.06.] Now calling **Scan_Parse_Parameter_Type**::*get_highest_value*. Removed explicit code for querying the database and handling the result.

```
< Irods_Object_Type::write_to_database definition 1063 > +≡
    status = Scan_Parse_Parameter_Type::get_highest_value(mysql_ptr, "Irods_Objects",
        "irods_object_id", id, true);
    if (status ≠ 0) {
        cerr ≪ "ERROR! In 'Irods_Object_Type::write_to_database':"
            ≪ endl ≪
            "'Scan_Parse_Parameter_Type::get_highest_value' failed, returning "
            ≪ status ≪ "."
            ≪ endl ≪ "Failed to retrieve highest value of 'Irods_Objects.irods_object_id'."
            ≪ endl ≪ "Will try to unlock tables"
            ≪ "before exiting unsuccessfully."
            ≪ endl;
        unlock_cerr_mutex();
    if (result) {
        mysql_free_result(result);
        result = 0;
    }
    ret_val = 1;
    goto UNLOCK_TABLES_WTD;
} /* if (status ≠ 0) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ "In 'Irods_Object_Type::write_to_database':"
            ≪ endl ≪
            "'submit_mysql_query' succeeded, returning 0."
            ≪ endl ≪
            "Retrieved highest value of 'Irods_Objects.irods_object_id' successfully"
            ≪
            "and incremented it."
            ≪ endl ≪ "'id' == "
            ≪ id ≪ endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
mysql_free_result(result);
result = 0;
sql_strm.str("");
```

1068.

Log

[LDF 2013.06.06.] Added this section.

```
< Irods_Object_Type::write_to_database definition 1063 > +≡
unsigned long avu_id;
status = Scan_Parse_Parameter_Type::get_highest_value(mysql_ptr, "Irods_AVUs", "irods_avu_id",
    avu_id, true);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ "ERROR! In 'Irods_Object_Type::write_to_database': " ≪ endl ≪
        "'Scan_Parse_Parameter_Type::get_highest_value' failed, returning " ≪ status ≪ ". " ≪
        endl ≪ "Failed to retrieve highest value of 'Irods_AVUs.irods_avu_id' from " ≪
        "database. Failed to set 'avu_id'." ≪ endl ≪
        "Will try to unlock tables before exiting unsuccessfully." ≪ endl;
    unlock_cerr_mutex();
    if (result) {
        mysql_free_result(result);
        result = 0;
    }
    ret_val = 1;
    goto UNLOCK_TABLES_WTD;
} /* if (status ≠ 0) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ "In 'Irods_Object_Type::write_to_database': " ≪ endl ≪
            "'Scan_Parse_Parameter_Type::get_highest_value' succeeded, returning 0. " ≪
            endl ≪ "Retrieved highest value of 'Irods_AVUs.irods_avu_id' from " ≪
            "database successfully and incremented it." ≪ endl ≪ "'avu_id' == " ≪ avu_id ≪
            endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1069.

```

⟨ Irods_Object_Type::write_to_database definition 1063 ⟩ +≡
    temp_strm.str("");
    if (created > 0) {
        temp_strm ≪ "from_unixtime(" ≪ created ≪ "),";
    }
    else temp_strm ≪ "0,";
    if (last_modified > 0) {
        temp_strm ≪ "from_unixtime(" ≪ last_modified ≪ ")";
    }
    else temp_strm ≪ "0";
    sql_strm ≪ "insert into Irods_Objects(irods_object_id, user_id, irods_server_id, "
    ≪ "irods_object_path, marked_for_deletion_from_archive, " ≪ "marked_for_deletion\\"
    ≪ "from_gwirdsif_db, " ≪ "deleted_from_archive, created, last_modified) "
    ≪ "values(" ≪ id ≪ ", " ≪ user_id ≪ ", 1, " ≪ path ≪ ", " ≪ "false, false, false, "
    ≪ temp_strm.str() ≪ ")";
    temp_strm.str("");
#endif /* DEBUG_COMPILE */
#if DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ "sql_strm.str() == " ≪ sql_strm.str() ≪ endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    status = submit_mysql_query(sql_strm.str().c_str(), result, mysql_ptr, 0, 0, &affected_rows);
    if (status ≠ 0) {
        lock_cerr_mutex();
        cerr ≪ "ERROR! In 'Irods_Object_Type::write_to_database': " ≪ endl ≪
            "'submit_mysql_query' failed, returning " ≪ status ≪ ". " ≪ endl ≪
            "Failed to insert row into 'Irods_Objects' database table. " ≪ endl ≪
            "Will try to unlock tables before exiting unsuccessfully. " ≪ endl;
        unlock_cerr_mutex();
        if (result) {
            mysql_free_result(result);
            result = 0;
        }
        ret_val = 1;
        goto UNLOCK_TABLES_WTD;
    } /* if (status ≠ 0) */
#endif /* DEBUG_COMPILE */
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ "In 'Irods_Object_Type::write_to_database': " ≪ endl ≪
            "'submit_mysql_query' succeeded, returning 0. " ≪ endl ≪
            "Inserted row into 'Irods_Objects' database table successfully. " ≪
            endl ≪ "'affected_rows' == " ≪ affected_rows ≪ endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
mysql_free_result(result);
result = 0;

```

```
sqlStrm.str("");
```

1070.

```
<Irods_Object_Type::write_to_database definition 1063> +≡
  if (handle_id_vector.size() > 0) { string comma_str;
    sqlStrm << "insert into Irods_Objects_Handles(irods_object_id, handle_id) values";
    for (vector<unsigned long int>::const_iterator iter = handle_id_vector.begin();
         iter != handle_id_vector.end(); ++iter) {
      sqlStrm << comma_str << "(" << id << ", " << *iter << ")";
      comma_str = ", ";
    }
  #if DEBUG_COMPILE
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "sql_strm.str() == " << sqlStrm.str() << endl;
      unlock_cerr_mutex();
    } /* if (DEBUG) */
  #endif /* DEBUG_COMPILE */
```

1071.

```

⟨ Irods_Object_Type::write_to_database definition 1063 ⟩ +≡
status = submit_mysql_query(sql_strm.str().c_str(), result, mysql_ptr, 0, 0, &affected_rows);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ "ERROR! In 'Irods_Object_Type::write_to_database':"
    ≪ endl ≪
        "submit_mysql_query' failed, returning" ≪ status ≪ "."
    ≪ endl ≪
        "Failed to insert row(s) into 'Irods_Objects_Handles' database table."
    ≪ endl ≪
        "Will try to unlock tables before exiting unsuccessfully."
    ≪ endl;
    unlock_cerr_mutex();
}
if (result) {
    mysql_free_result(result);
    result = 0;
}
ret_val = 1;
goto UNLOCK_TABLES_WTD;
} /* if (status ≠ 0) */
#if DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ "In 'Irods_Object_Type::write_to_database':"
        ≪ endl ≪
            "submit_mysql_query' succeeded, returning 0."
        ≪ endl ≪
            "Inserted row(s) into 'Irods_Objects_Handles' database table successfully."
        ≪ endl ≪
            "'affected_rows' == " ≪ affected_rows ≪ endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
mysql_free_result(result);
result = 0;
sql_strm.str("");
#if DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ "In 'Irods_Object_Type::write_to_database'"
        ≪ endl ≪
            "handle_id_vector.size() == 0."
        ≪ endl ≪
            "Not writing rows to 'Irods_Objects_Handles' database table."
        ≪ endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1072.

Log

[LDF 2013.06.06.] Added this section.

```

⟨ Irods_Object_Type::write_to_database definition 1063 ⟩ +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In `Irods_Object_Type::write_to_database' : " << "avu_vector.size() == " <<
            avu_vector.size() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    if (avu_vector.size() > 0) { string comma_str;
        sql_strm.str("");
        sql_strm << "insert into Irods_AVUs" << "(irods_avu_id, irods_object_id, attribute,
            value, units, time_set)" << "values";
        for (vector<Irods_AVU_Type>::const_iterator iter = avu_vector.begin(); iter != avu_vector.end();
            ++iter) {
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "convert_seconds(iter->time_set) == " << convert_seconds(iter->time_set) << endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        sql_strm << comma_str << "(" << id++ << ", " << id << ", " << " " << iter->attribute << ", "
            << iter->value << ", " << " " << iter->units << ", " << "from_unixtime(" << iter->time_set << ")";
            comma_str = ", ";
        } /* for */
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "sql_strm.str() == " << sql_strm.str() << endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1073.

```

⟨ Irods_Object_Type::write_to_database definition 1063 ⟩ +≡
    status = submit_mysql_query(sql_strm.str().c_str(), result, mysql_ptr, 0, 0, &affected_rows);
    if (status ≠ 0) {
        lock_cerr_mutex();
        cerr ≪ "ERROR! In 'Irods_Object_Type::write_to_database':"
        ≪ endl ≪ "'submit_mysql_query' failed, returning"
        ≪ status ≪ "."
        ≪ endl ≪ "Failed to insert row(s) into 'Irods_AVUs' database table."
        ≪ endl ≪ "Will try to unlock tables before exiting unsuccessfully."
        ≪ endl;
        unlock_cerr_mutex();
    }
    if (result) {
        mysql_free_result(result);
        result = 0;
    }
    ret_val = 1;
    goto UNLOCK_TABLES_WTD;
} /* if (status ≠ 0) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ "In 'Irods_Object_Type::write_to_database':"
        ≪ endl ≪ "'submit_mysql_query' succeeded, returning 0."
        ≪ endl ≪ "Inserted row(s) into 'Irods_AVUs' database table successfully."
        ≪ endl ≪ "'affected_rows' == "
        ≪ affected_rows ≪ endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
mysql_free_result(result);
result = 0; /* if (avu_vector.size() > 0) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ "In 'Irods_Object_Type::write_to_database':"
        ≪ endl ≪ "'avu_vector.size()' == 0."
        ≪ endl ≪ "Not writing rows to 'Irods_AVUs' database table."
        ≪ endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1074.

```

⟨ Irods_Object_Type::write_to_database definition 1063 ⟩ +≡
UNLOCK_TABLES_WTD: status = submit_mysql_query("unlock_tables", result, mysql_ptr);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ "ERROR! In 'Irods_Object_Type::write_to_database':"
    ≪ endl ≪ "'submit_mysql_query' failed, returning"
    ≪ status ≪ "."
    ≪ endl ≪ "Failed to unlock_tables ('Irods_Objects', 'Irods_AVUs'"
    ≪ "and 'Irods_Objects_Handles'" ≪ "were locked)."
    ≪ endl ≪ "Exiting function unsuccessfully with return value 1."
    ≪ endl;
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    return 1;
} /* if (status ≠ 0) */
#ifndef DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "In 'Irods_Object_Type::write_to_database':"
    ≪ endl ≪ "'submit_mysql_query' succeeded, returning 0."
    ≪ endl ≪ "Unlocked tables successfully. ('Irods_Objects' and 'Irods_Objects_Handles'"
    ≪ "tables were locked.)"
    ≪ endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
mysql_free_result(result);
result = 0;

```

1075.

```

⟨ Irods_Object_Type::write_to_database definition 1063 ⟩ +≡
if (ret_val ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ "Exiting 'Irods_Object_Type::write_to_database' unsuccessfully with"
    ≪ "return_value" ≪ ret_val ≪ "."
    ≪ endl;
    unlock_cerr_mutex();
    return ret_val;
} /* if (ret_val ≠ 0) */
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "Exiting 'Irods_Object_Type::write_to_database' successfully"
    ≪ "with return_value 0."
    ≪ endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
return 0; /* End of Irods_Object_Type::write_to_database definition */

```

1076. Get from database (*get_from_database*). [LDF 2013.01.08.]

Returns number of rows retrieved or a negative integer upon error. The number of rows should be either 0 or 1. Currently, only -1 is returned in the error cases. [LDF 2013.01.09.]

Log

[LDF 2013.01.08.] Added this function.

[LDF 2013.01.09.] Changed name of argument **bool row_count_only** to *id_only*: **Irods_Object_Type::id** must always be set, but the other data from the **Irods_Objects** table may not be needed. Added code for fetching the value.

[LDF 2013.01.31.] Added code for retrieving **handle_id** values from the **Irods_Objects_Handles** database table.

< Irods_Object_Type function declarations 1039 > +≡

```
int get_from_database(MYSQL *mysql_ptr, bool id_only = false);
```

1077.

< Irods_Object_Type::get_from_database definition 1077 > ≡

```
int Irods_Object_Type::get_from_database(MYSQL *mysql_ptr, bool id_only){ bool DEBUG = false;
    /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    int ret_val;
    unsigned long int curr_handle_id = 0;
    Handle_Value_Type curr_handle;
    unsigned long int temp_val = 0UL;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Entering 'Irods_Object_Type::get_from_database'." << endl <<
            "'id_only' == " << id_only << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

See also sections 1078, 1079, 1080, 1081, 1082, 1083, 1084, 1085, 1086, 1087, 1088, 1089, 1090, 1091, 1092, 1093, 1094, 1095, 1096, 1097, 1098, 1099, 1100, 1101, 1102, 1103, 1104, 1105, 1106, 1107, 1108, 1109, 1110, 1111, 1112, 1113, 1114, 1115, 1116, 1117, and 1118.

This code is used in section 1258.

1078.

< Irods_Object_Type::get_from_database definition 1077 > +≡

```
MYSQL_RES *result = 0;
MYSQL_ROW curr_row;
unsigned int row_ctr;
unsigned int field_ctr;
long int affected_rows;
stringstream sql_strm;
```

1079.

```

⟨ Irods_Object_Type::get_from_database definition 1077 ⟩ +≡
    status = mysql_select_db(mysql_ptr, "gwirldsif");
    if (status == 0) {
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "In 'Irods_Object_Type::get_from_database': " << 'mysql_select_db' su\
                cceeded'." << endl << "Selected 'gwirldsif' database successfully." << endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    } /* if (status == 0) */
    else /* status != 0 */
    {
        lock_cerr_mutex();
        cerr << "In 'Irods_Object_Type::get_from_database': " <<
            "'mysql_select_db' failed, returning " << status << endl <<
            "Failed to select 'gwirldsif' database:" << endl << "Error: " << mysql_error(mysql_ptr) <<
            endl << "Exiting function unsuccessfully with return value -1." << endl;
        unlock_cerr_mutex();
        return -1;
    } /* else (status != 0) */
}

```

1080. Check whether a row already exists for *irods_object_filename* in the **Irods_Objects** database table.
[LDF 2013.01.07.]

There should only be one row, so the “order” clause in the “select” query is just for insurance. [LDF 2013.01.09.] ■

Log

[LDF 2013.08.12.] Added code for retrieving data from the database if *id* > 0.

```

⟨ Irods_Object_Type::get_from_database definition 1077 ⟩ +≡
    sql_strm.str("");
    sql_strm << "select_irods_object_id, irods_server_id, " <<
        "marked_for_deletion_from_archive, marked_for_deletion_from_gwirldsif_db, " <<
        "deleted_from_archive, unix_timestamp(created), " << "unix_timestamp(last\
        _modified), " << "unix_timestamp(delete_from_archive_timestamp), " <<
        "unix_timestamp(delete_from_gwirldsif_db_timestamp), irods_object_path " <<
        "from_gwirldsif.Irods_Objects";
    if (id > 0) sql_strm << "where_irods_object_id=" << id << " ";
    else sql_strm << "where_user_id=" << user_id << " and_irods_object_path=" << " " << path <<
        " ";
    sql_strm << "order_by_irods_object_id_desc";
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "sql_strm.str() == " << sql_strm.str() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    status = submit_mysql_query(sql_strm.str(), result, mysql_ptr, &row_ctr, &field_ctr);
}

```

1081.

```
< Irods_Object_Type::get_from_database definition 1077 > +≡
  if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ "ERROR! In 'Irods_Object_Type::get_from_database':" ≪ endl ≪
      "'Scan_Parse_Parameter_Type::submit_mysql_query' failed, returning " ≪
      status ≪ "." ≪ endl ≪ mysql_error(mysql_ptr) ≪ endl ≪
      "Failed to retrieve data from 'Irods_Objects' table for " ≪ "'path'" ≪
      path ≪ "." ≪ endl ≪ "Exiting function unsuccessfully with return value -1." ≪ endl;
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    return -1;
  } /* if (status ≠ 0) */
```

1082.

```
< Irods_Object_Type::get_from_database definition 1077 > +≡
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr ≪ "In 'Irods_Object_Type::get_from_database':" ≪ endl ≪
        "'Scan_Parse_Parameter_Type::submit_mysql_query' succeeded, returning 0." ≪ endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1083.

Log

[LDF 2013.01.10.] BUG FIX: Added this section. Previously, the value of *row_ctr* wasn't tested, so *mysql_fetch_row* was called even if no rows had been returned from the query.

[LDF 2013.01.31.] Now setting *ret_val* = *row_ctr*.

```
< Irods_Object_Type::get_from_database definition 1077 > +≡
  if (row_ctr ≡ 0) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
      lock_cerr_mutex();
      cerr ≪ "In 'Irods_Object_Type::get_from_database':" ≪
        endl ≪ "'row_ctr' == 0. No rows retrieved." ≪ endl ≪
        "Exiting function successfully with return value 0." ≪ endl;
      unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    mysql_free_result(result);
    return 0;
  } /* if (row_ctr ≡ 0) */
  else if (row_ctr > 0) ret_val = row_ctr;
```

1084.

```

⟨ Irods_Object_Type::get_from_database definition 1077 ⟩ +≡
  if ((curr_row = mysql_fetch_row(result)) == 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Irods_Object_Type::get_from_database':"
    << endl << "'mysql_fetch_row' failed:" << endl << mysql_error(mysql_ptr) << endl <<
    "Exiting function unsuccessfully with return value -1." << endl;
    unlock_cerr_mutex();
  if (result) {
    mysql_free_result(result);
    result = 0;
  }
  return -1;
} /* if (curr_row = mysql_fetch_row(result) == 0) */
#ifndef DEBUG_COMPILE
else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'Irods_Object_Type::get_from_database':"
    << endl << "'mysql_fetch_row' succeeded."
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1085. irods_object_id. [LDF Undated.]

```

⟨ Irods_Object_Type::get_from_database definition 1077 ⟩ +≡
  if (curr_row[0] == 0 || strlen(curr_row[0]) == 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Irods_Object_Type::get_from_database':"
    << endl << "'curr_row[0]' is NULL or empty." << endl << "Failed to read 'irods_objects.irods_object_id' from "
    << "database. Can't set 'id'." << endl <<
    "Exiting function unsuccessfully with return value -1." << endl;
    unlock_cerr_mutex();
  if (result) {
    mysql_free_result(result);
    result = 0;
  }
  return -1;
} /* if (curr_row[0] == 0 || strlen(curr_row[0]) == 0) */
#ifndef DEBUG_COMPILE
else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "curr_row[0] == " << curr_row[0] << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1086.

```
< Irods_Object_Type::get_from_database definition 1077 > +≡
    errno = 0;
    id = strtoul(curr_row[0], 0, 10);
    if (id == ULONG_MAX) {
        lock_cerr_mutex();
        cerr << "ERROR! In 'Irods_Object_Type::get_from_database':"
            << endl <<
            "'strtoul' failed, returning 'ULONG_MAX':"
            << endl << "Error: "
            << strerror(errno) <<
            endl << "Failed to convert 'Irods_Objects.irods_object_id' from"
            << "database. Failed to set 'id'. Will set to 0."
            << endl <<
            "Exiting function unsuccessfully with return value -1."
            << endl;
        unlock_cerr_mutex();
        id = 0;
    }
    if (result) {
        mysql_free_result(result);
        result = 0;
    }
    return -1;
} /* if (id == ULONG_MAX) */
#endif /* DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'Irods_Object_Type::get_from_database':"
        << " 'id': "
        << id << endl;
    unlock_cerr_mutex();
}
/* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1087.

```
< Irods_Object_Type::get_from_database definition 1077 > +≡
    if (id_only) {
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "In 'Irods_Object_Type::get_from_database':"
                << endl <<
                "'id_only' == 'true'. Only fetching 'irods_object_id', not other data."
                << endl << "Exiting function successfully with return value 'row_ctr' == "
                << row_ctr << "."
                << endl;
            unlock_cerr_mutex();
        }
        /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        mysql_free_result(result);
        return ret_val;
    }
    /* if (id_only) */
```

1088. Fetch other fields from *curr_row* and set data members. [LDF 2013.08.07.]

```
< Irods_Object_Type::get_from_database definition 1077 > +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "'id_only'=="'false'.uuFetchinguumoreuu"data." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1089. *irods_server_id*. [LDF 2013.08.07.]

```
< Irods_Object_Type::get_from_database definition 1077 > +≡
temp_val = strtoul(curr_row[1], 0, 10);
if (temp_val == ULONG_MAX) {
    lock_cerr_mutex();
    cerr << "ERROR!uuInuu'Irods_Object_Type::get_from_database':"
    << endl <<
        "'strtoul'failed, returning 'ULONG_MAX':"
    << endl << "Error:uu" << strerror(errno) <<
        endl << "Faileduuconvertuu'gwirdsif.Irods_Objects.irods_server_id'uuvalueuufromuu"
    << "database.uuFaileduusetuu'irods_server_id'.uuWilluusetuu'id'uu0."
    << endl <<
        "Exitinguufunctionuuunsuccessfullyuuwithuureturnuuvalueuu-1."
    << endl;
    unlock_cerr_mutex();
    id = 0;
    if (result) {
        mysql_free_result(result);
        result = 0;
    }
    return -1;
} /* if (temp_val == ULONG_MAX) */
irods_server_id = temp_val;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Inuu'Irods_Object_Type::get_from_database':uu"
        << "'irods_server_id':uu" <<
            irods_server_id << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1090. *marked_for_deletion_from_archive*. [LDF 2013.08.07.]

```
< Irods_Object_Type::get_from_database definition 1077 > +≡
temp_val = strtoul(curr_row[2], 0, 10);
if (temp_val ≡ ULONG_MAX) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Irods_Object_Type::get_from_database':"
    endl << "'strtoul' failed, returning 'ULONG_MAX':"
    endl << "Error: " << strerror(errno) << endl <<
    "Failed to convert 'gwiridsif.Irods_Objects.marked_for_deletion_from_archive' "
    "value from " << "database.Failed to set 'marked_for_deletion_from_archive'."
    "Will set 'id' to 0." << endl << "Exiting function unsuccessfully with re\"
    turn value -1." << endl;
unlock_cerr_mutex();
id = 0;
if (result) {
    mysql_free_result(result);
    result = 0;
}
return -1;
} /* if (temp_val ≡ ULONG_MAX) */
marked_for_deletion_from_archive = (temp_val ≡ 0UL) ? false : true;
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'Irods_Object_Type::get_from_database':"
    "'marked_for_deletion_from_archive': " << marked_for_deletion_from_archive << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1091. *marked_for_deletion_from_gwirdsif_db*. [LDF 2013.08.07.]

```
< Irods_Object_Type::get_from_database definition 1077 > +≡
temp_val = strtoul(curr_row[3], 0, 10);
if (temp_val ≡ ULONG_MAX) {
    lock_cerr_mutex();
    cerr ≪ "ERROR! In 'Irods_Object_Type::get_from_database':"
    ≪ endl ≪ "'strtoul' failed, returning 'ULONG_MAX':"
    ≪ endl ≪ "Error: "
    strerror(errno) ≪ endl ≪ "Failed to convert 'gwirdsif.Irods_Objec\
ts.marked_for_deletion_from_gwirdsif_db' to value from database. Failed to\
set 'marked_for_deletion_from_gwirdsif_db' to 'id' to 0."
    ≪ endl ≪ "Will set 'id' to 0." ≪ endl ≪
    "Exiting function unsuccessfully with return value -1."
    unlock_cerr_mutex();
    id = 0;
    if (result) {
        mysql_free_result(result);
        result = 0;
    }
    return -1;
} /* if (temp_val ≡ ULONG_MAX) */
marked_for_deletion_from_gwirdsif_db = (temp_val ≡ 0UL) ? false : true;
#if DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "In 'Irods_Object_Type::get_from_database':"
    ≪ "'marked_for_deletion_from_gwirdsif_db':"
    ≪ marked_for_deletion_from_gwirdsif_db ≪
    endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1092. *deleted_from_archive*. [LDF 2013.08.07.]

```
< Irods_Object_Type::get_from_database definition 1077 > +≡
temp_val = strtoul(curr_row[4], 0, 10);
if (temp_val ≡ ULONG_MAX) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Irods_Object_Type::get_from_database':"
    endl << "'strtoul' failed, returning 'ULONG_MAX':"
    endl << "Error:" << strerror(errno) << endl <<
    "Failed to convert 'gwiridsif.Irods_Objects.deleted_from_archive' value from"
    "database. Failed to set 'deleted_from_archive'. Will set 'id' to 0." << endl <<
    "Exiting function unsuccessfully with return value -1." << endl;
unlock_cerr_mutex();
id = 0;
if (result) {
    mysql_free_result(result);
    result = 0;
}
return -1;
} /* if (temp_val ≡ ULONG_MAX) */
deleted_from_archive = (temp_val ≡ 0UL) ? false : true;
#if DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'Irods_Object_Type::get_from_database':"
    endl << "'deleted_from_archive' << "
    deleted_from_archive << endl;
    unlock_cerr_mutex();
}
/* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1093. *created.* [LDF 2013.08.07.]

```
< Irods_Object_Type::get_from_database definition 1077 > +≡
temp_val = strtoul(curr_row[5], 0, 10);
if (temp_val ≡ ULONG_MAX) {
    lock_cerr_mutex();
    cerr ≪ "ERROR! In 'Irods_Object_Type::get_from_database':"
    ≪ endl ≪ "'strtoul' failed, returning 'ULONG_MAX':"
    ≪ endl ≪ "Error: " ≪ strerror(errno) ≪ endl
    ≪ "Failed to convert 'gwirdsif.Irods_Objects.created' value from "
    ≪ "database. Failed to set 'created'. Will set 'id' to 0."
    ≪ endl ≪ "Exiting function unsuccessfully with return value -1."
    unlock_cerr_mutex();
    id = 0;
    if (result) {
        mysql_free_result(result);
        result = 0;
    }
    return -1;
} /* if (temp_val ≡ ULONG_MAX) */
created = static_cast<time_t>(temp_val);
#if DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "In 'Irods_Object_Type::get_from_database':"
    ≪ "'created': " ≪ created ≪ endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1094. *last_modified*. [LDF 2013.08.07.]

```

⟨ Irods_Object_Type::get_from_database definition 1077 ⟩ +≡
temp_val = strtoul(curr_row[6], 0, 10);
if (temp_val == ULONG_MAX) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Irods_Object_Type::get_from_database':"
        << endl <<
        "'strtoul' failed, returning ULONG_MAX'" << endl << "Error: "
        << strerror(errno) <<
        endl << "Failed to convert gwiridsif.Irods_Objects.last_modified"
        << endl << "database.Failed to set 'last_modified'. Will set 'id' to 0."
        << endl << "Exiting function unsuccessfully with return value -1."
        << endl;
unlock_cerr_mutex();
id = 0;
if (result) {
    mysql_free_result(result);
    result = 0;
}
return -1;
} /* if (temp_val == ULONG_MAX) */
last_modified = static_cast<time_t>(temp_val);
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'Irods_Object_Type::get_from_database':"
        << endl << "'last_modified':"
        << last_modified << endl;
    unlock_cerr_mutex();
}
/* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1095. *delete_from_archive_timestamp*. [LDF 2013.08.08.]

```
< Irods_Object_Type::get_from_database definition 1077 > +≡
temp_val = strtoul(curr_row[7], 0, 10);
if (temp_val ≡ ULONG_MAX) {
    lock_cerr_mutex();
    cerr ≪ "ERROR! In 'Irods_Object_Type::get_from_database':"
    ≪ endl ≪ "'strtoul' failed, returning 'ULONG_MAX':"
    ≪ endl ≪ "Error: "
    strerror(errno) ≪ endl ≪ "Failed to convert 'gwiridsif.Irods_Objec\
ts.delete_from_archive_timestamp' to value from"
    ≪ "database. Failed to set 'delete_from_archive_timestamp'. Will set 'id' to 0."
    ≪ endl ≪ "Exiting function unsuccessfully with return value -1."
    ≪ endl;
unlock_cerr_mutex();
id = 0;
if (result) {
    mysql_free_result(result);
    result = 0;
}
return -1;
} /* if (temp_val ≡ ULONG_MAX) */
delete_from_archive_timestamp = static_cast<time_t>(temp_val);
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "In 'Irods_Object_Type::get_from_database':"
    ≪ "'delete_from_archive_timestamp':"
    ≪ delete_from_archive_timestamp ≪ endl;
unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1096. *delete_from_gwirdsif_db_timestamp.* [LDF 2013.08.08.]

```

⟨ Irods_Object_Type::get_from_database definition 1077 ⟩ +≡
temp_val = strtoul(curr_row[8], 0, 10);
if (temp_val ≡ ULONG_MAX) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Irods_Object_Type::get_from_database':"
    endl << "'strtoul' failed, returning 'ULONG_MAX':"
    endl << "Error: " << strerror(errno) << endl <<
    "Failed to convert 'gwirdsif.Irods_Objects.delete_from_gwirdsif_db_timestamp' "
    "value from " << "database.Failed to set 'delete_from_gwirdsif_db_timestamp'."
    "Will set 'id' to 0." << endl << "Exiting function unsuccessfully with re\"
    turn value -1." << endl;
    unlock_cerr_mutex();
    id = 0;
    if (result) {
        mysql_free_result(result);
        result = 0;
    }
    return -1;
} /* if (temp_val ≡ ULONG_MAX) */
delete_from_gwirdsif_db_timestamp = static_cast<time_t>(temp_val);
#if DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'Irods_Object_Type::get_from_database':"
        "delete_from_gwirdsif_db_timestamp': "
        << delete_from_gwirdsif_db_timestamp << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1097. *path.* [LDF 2013.08.16.]

Log

[LDF 2013.08.16.] Added this section.

```

⟨ Irods_Object_Type::get_from_database definition 1077 ⟩ +≡
if (curr_row[9] ≠ 0 ∧ strlen(curr_row[9]) > 0) path = curr_row[9];
else path = "";

```

1098.

```

⟨ Irods_Object_Type::get_from_database definition 1077 ⟩ +≡
mysql_free_result(result);
result = 0;
#if DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        show("*this:");
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1099.

```
< Irods_Object_Type::get_from_database definition 1077 > +≡
  if (result) {
    mysql_free_result(result);
    result = 0;
  }
  sql_strm.str("");
  sql_strm << "select distinct h.handle_id, h.handle from handlesystem_stal
  ndalone.handles as h, " << "gwiridsif.Irods_Objects_Handles as ioh" <<
  "where ioh.irods_object_id = " << id << " and h.handle_id = ioh.handle_id" <<
  "order by h.handle_id";
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "sql_strm.str() == " << sql_strm.str() << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  status = submit_mysql_query(sql_strm.str(), result, mysql_ptr, &row_ctr, &field_ctr);
```

1100.

```
< Irods_Object_Type::get_from_database definition 1077 > +≡
  if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Irods_Object_Type::get_from_database':" << endl <<
      "'Scan_Parse_Parameter_Type::submit_mysql_query' failed, returning " <<
      status << "." << endl << mysql_error(mysql_ptr) << endl <<
      "Failed to retrieve data from 'Irods_Objects_Handle' table for " << "'id' == " <<
      id << "." << endl << "Exiting function unsuccessfully with return value -1." << endl;
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    return -1;
  } /* if (status ≠ 0) */
```

1101.

```
< Irods_Object_Type::get_from_database definition 1077 > +≡
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "In 'Irods_Object_Type::get_from_database':" << endl <<
        "'Scan_Parse_Parameter_Type::submit_mysql_query' succeeded, returning 0." << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1102.

```
< Irods_Object_Type::get_from_database definition 1077 > +≡
    if (row_ctr == 0) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'Irods_Object_Type::get_from_database':" << endl <<
            "'row_ctr'==0.No rows retrieved from 'Irods_Objects_Handles'." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    mysql_free_result(result);
    result = 0;
} /* if (row_ctr == 0) */
```

1103.

```
< Irods_Object_Type::get_from_database definition 1077 > +≡
    for (int i = 0; i < row_ctr; ++i) {
        if ((curr_row = mysql_fetch_row(result)) == 0) {
            lock_cerr_mutex();
            cerr << "ERROR! In 'Irods_Object_Type::get_from_database':" << endl <<
                "'mysql_fetch_row' failed:" << endl << mysql_error(mysql_ptr) << endl <<
                "Exiting function unsuccessfully with return value -1." << endl;
            unlock_cerr_mutex();
        if (result) {
            mysql_free_result(result);
            result = 0;
        }
        return -1;
    } /* if (curr_row = mysql_fetch_row(result) == 0) */
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "In 'Irods_Object_Type::get_from_database':" << endl <<
                "'mysql_fetch_row' succeeded." << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1104.

```
< Irods_Object_Type::get_from_database definition 1077 > +≡
  if (curr_row[0] == 0 ∨ strlen(curr_row[0]) == 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Irods_Object_Type::get_from_database':"
    << endl << "'curr_row[0]' is NULL or empty."
    << endl << "Failed to read 'Irods_Objects_Handles.handle_id' from"
    << "database. Can't set 'id'." << endl <<
    "Exiting function unsuccessfully with return value -1." << endl;
    unlock_cerr_mutex();
  }
  if (result) {
    mysql_free_result(result);
    result = 0;
  }
  return -1;
} /* if (curr_row[0] == 0 ∨ strlen(curr_row[0]) == 0) */
#ifndef DEBUG_COMPILE
else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "curr_row[0] == "
    << curr_row[0] << endl;
    cerr << "curr_row[1] == "
    << curr_row[1] << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1105.

```

⟨ Irods_Object_Type::get_from_database definition 1077 ⟩ +≡
errno = 0;
curr_handle_id = strtoul(curr_row[0], 0, 10);
if (curr_handle_id == ULONG_MAX) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Irods_Object_Type::get_from_database':"
        << endl <<
        "'strtoul' failed, returning 'ULONG_MAX':"
        << endl << "Error: "
        << strerror(errno) <<
        endl << "Failed to convert 'Irods_Objects_Handles.handle_id' from"
        << "database. Failed to set 'curr_handle_id'." << endl <<
        "Exiting function unsuccessfully with return value -1." << endl;
unlock_cerr_mutex();
if (result) {
    mysql_free_result(result);
    result = 0;
}
return -1;
} /* if (curr_handle_id == ULONG_MAX) */
#endif DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'Irods_Object_Type::get_from_database':"
        << "'curr_handle_id':"
        << curr_handle_id << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
handle_id_vector.push_back(curr_handle_id);
handle_name_string_vector.push_back(curr_row[1]); } /* for */
if (result) {
    mysql_free_result(result);
    result = 0;
}
sql_strm.str("");

```

1106.

```

⟨ Irods_Object_Type::get_from_database definition 1077 ⟩ +≡
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "handle_id_vector.size() == "
        << handle_id_vector.size() << endl;
    cerr << "handle_name_string_vector.size() == "
        << handle_name_string_vector.size() << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1107. Retrieve AVUs. [LDF 2013.06.06.]**Log**

[LDF 2013.06.06.] Added this section.

```

⟨Irods_Object_Type::get_from_database definition 1077⟩ +≡
  if (result) {
    mysql_free_result(result);
    result = 0;
  }
  sql_strm.str("");
  Irods_AVU_Type curr_avu;
  sql_strm << "select_irods_avu_id,attribute,value,units,time_set_from"
  << "Irods_AVUs where irods_object_id=" << id << " order by irods_avu_id";
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In `Irods_Object_Type::get_from_database': "
    << 'sql_strm.str()' :<< endl
    << sql_strm.str() << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  status = submit_mysql_query(sql_strm.str(), result, mysql_ptr, &row_ctr, &field_ctr);
  if (status != 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In `Irods_Object_Type::get_from_database':"
    << endl <<
      "Scan_Parser_Parameter_Type::submit_mysql_query' failed, returning"
    << endl << status << "."
    << endl << mysql_error(mysql_ptr) << endl <<
      "Failed to retrieve data from `Irods_AVUs' table for"
    << " `irods_object_id'=" << id << "."
    << endl << "Exiting function unsuccessfully with return value -1."
    << endl;
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    return -1;
  } /* if (status != 0) */

```

1108.

```

⟨Irods_Object_Type::get_from_database definition 1077⟩ +≡
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "In `Irods_Object_Type::get_from_database':"
      << endl <<
        "Scan_Parser_Parameter_Type::submit_mysql_query' succeeded, returning 0."
      << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1109.

```
< Irods_Object_Type::get_from_database definition 1077 > +≡
    if (row_ctr == 0) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'Irods_Object_Type::get_from_database':" << endl <<
            "'row_ctr'==0.No rows retrieved from 'Irods_AVUs'." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    mysql_free_result(result);
    result = 0;
} /* if (row_ctr == 0) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'Irods_Object_Type::get_from_database':" << endl << "'row_ctr'==" <<
            row_ctr << "." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1110.

```

⟨ Irods_Object_Type::get_from_database definition 1077 ⟩ +≡
  for (int i = 0; i < row_ctr; ++i) {
    if ((curr_row = mysql_fetch_row(result)) == 0) {
      lock_cerr_mutex();
      cerr << "ERROR! In 'Irods_Object_Type::get_from_database':" << endl <<
        "'mysql_fetch_row' failed:" << endl << mysql_error(mysql_ptr) << endl <<
        "Exiting function unsuccessfully with return value -1." << endl;
      unlock_cerr_mutex();
    }
    if (result) {
      mysql_free_result(result);
      result = 0;
    }
  }
  return -1;
} /* if (curr_row = mysql_fetch_row(result) == 0) */
#endif DEBUG_COMPILE
else
if (DEBUG) {
  lock_cerr_mutex();
  cerr << "In 'Irods_Object_Type::get_from_database':" << endl <<
    "'mysql_fetch_row' succeeded."
  for (int j = 0; j < field_ctr; ++j) {
    if (curr_row[j] & strlen(curr_row[j]) > 0)
      cerr << "curr_row[" << j << "] == " << curr_row[j] << endl;
    else cerr << "'curr_row[" << j << "]' is NULL or empty." << endl;
  }
  unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1111. *id.* [LDF 2013.06.06.]

```

⟨ Irods_Object_Type::get_from_database definition 1077 ⟩ +≡
  if (curr_row[0] == 0 ∨ strlen(curr_row[0]) == 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Irods_Object_Type::get_from_database':" << endl <<
      "'curr_row[0]' is NULL or empty." << endl << "Failed to read 'iro\"
      ds_AVUs.irods_avu_id' from database. Can't set 'curr_avu.id'." << endl <<
      "Exiting function unsuccessfully with return value -1." << endl;
    unlock_cerr_mutex();
  }
  if (result) {
    mysql_free_result(result);
    result = 0;
  }
  return -1;
} /* if (curr_row[0] == 0 ∨ strlen(curr_row[0]) == 0) */

```

1112.

```

⟨ Irods_Object_Type::get_from_database definition 1077 ⟩ +≡
errno = 0;
curr_avu.id = strtoul(curr_row[0], 0, 10);
if (curr_avu.id == ULONG_MAX) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Irods_Object_Type::get_from_database':"
        << endl <<
        "'strtoul' failed, returning 'ULONG_MAX':"
        << endl << "Error: "
        << strerror(errno) <<
        "Failed to convert the value of 'Irods_AVUs.irods_avu_id' to"
        "retrieved from the database." << endl <<
        "Failed to set 'curr_avu.id'." << endl <<
        "Exiting function unsuccessfully with return value -1." << endl;
unlock_cerr_mutex();
if (result) {
    mysql_free_result(result);
    result = 0;
}
return -1;
} /* if (curr_avu.id == ULONG_MAX) */
#endif /* DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'Irods_Object_Type::get_from_database':"
        << curr_avu.id <<
        curr_avu.id << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1113. attribute. [LDF 2013.06.06.]

```

⟨ Irods_Object_Type::get_from_database definition 1077 ⟩ +≡
if (curr_row[1] == 0 || strlen(curr_row[1]) == 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Irods_Object_Type::get_from_database':"
        << endl <<
        "'curr_row[1]' is NULL or empty."
        << endl << "Failed to read 'Irods_AVUs.attribute' from"
        "database. Can't set 'curr_avu.attribute'." << endl <<
        "Exiting function unsuccessfully with return value -1." << endl;
unlock_cerr_mutex();
if (result) {
    mysql_free_result(result);
    result = 0;
}
return -1;
} /* if (curr_row[1] == 0 || strlen(curr_row[1]) == 0) */
else {
    curr_avu.attribute = curr_row[1];
}

```

1114. *value*. [LDF 2013.06.06.]

```
<Irods_Object_Type::get_from_database definition 1077> +≡
  if (curr_row[2] ≡ 0) {
    lock_cerr_mutex();
    cerr ≪ "ERROR! In 'Irods_Object_Type::get_from_database':" ≪ endl ≪
      "'curr_row[2]' is NULL." ≪ endl ≪ "Failed to read 'Irods_AVUs.value' from"
      "database. Can't set 'curr_avu.value'." ≪ endl ≪
      "Exiting function unsuccessfully with return value -1." ≪ endl;
    unlock_cerr_mutex();
  }
  if (result) {
    mysql_free_result(result);
    result = 0;
  }
  return -1;
} /* if (curr_row[2] ≡ 0) */
else if (strlen(curr_row[2]) > 0) {
  curr_avu.value = curr_row[2];
}
else curr_avu.value = "";
```

1115. *units*. [LDF 2013.06.06.]

```
<Irods_Object_Type::get_from_database definition 1077> +≡
  if (curr_row[3] ≡ 0) {
    lock_cerr_mutex();
    cerr ≪ "ERROR! In 'Irods_Object_Type::get_from_database':" ≪ endl ≪
      "'curr_row[3]' is NULL." ≪ endl ≪ "Failed to read 'Irods_AVUs.units' from"
      "database. Can't set 'curr_avu.units'." ≪ endl ≪
      "Exiting function unsuccessfully with return units -1." ≪ endl;
    unlock_cerr_mutex();
  }
  if (result) {
    mysql_free_result(result);
    result = 0;
  }
  return -1;
} /* if (curr_row[3] ≡ 0) */
else if (strlen(curr_row[3]) > 0) {
  curr_avu.units = curr_row[3];
}
else curr_avu.units = "";
```

1116. *time_set*. [LDF 2013.06.06.]

```
< Irods_Object_Type::get_from_database definition 1077 > +≡
if (curr_row[4] ≡ 0 ∨ strlen(curr_row[4]) ≡ 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Irods_Object_Type::get_from_database':" <<
        endl << "'curr_row[4]' is NULL or empty." << endl <<
        "Failed to read 'Irods_AVUs.irods_avu_time_set' from "
        "database. Can't set 'curr_avu.time_set'." << endl <<
        "Exiting function unsuccessfully with return value -1." << endl;
    unlock_cerr_mutex();
    if (result) {
        mysql_free_result(result);
        result = 0;
    }
    return -1;
} /* if (curr_row[4] ≡ 0 ∨ strlen(curr_row[4]) ≡ 0) */
```

1117.

```

⟨ Irods_Object_Type::get_from_database definition 1077 ⟩ +≡
time_t temp_time_val;
status = get_seconds_since_epoch(curr_row[4], temp_time_val);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Irods_Object_Type::get_from_database':"
        << endl <<
        "'get_seconds_since_epoch' failed, returning "
        << status << "." << endl <<
        "Failed to convert the value of 'Irods_AVUs.irods_avu_time_set' "
        << endl << "retrieved from the database." << endl <<
        "Failed to set 'curr_avu.time_set'." <<
        endl << "Exiting function unsuccessfully with return value -1." << endl;
    unlock_cerr_mutex();
    if (result) {
        mysql_free_result(result);
        result = 0;
    }
    return -1;
} /* if (curr_avu.time_set ≡ ULONG_MAX) */
else {
    curr_avu.time_set = static_cast<unsigned long int>(temp_time_val);
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'Irods_Object_Type::get_from_database':"
            << "'curr_avu.time_set':"
            << curr_avu.time_set << " "
            << convert_seconds(curr_avu.time_set) << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    /* else */
    curr_avu.irods_object_id = id;
    avu_vector.push_back(curr_avu);
    curr_avu.clear(); } /* for */
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'Irods_Object_Type::get_from_database':"
            << "'avu_vector.size()':"
            << avu_vector.size() << endl;
        if (avu_vector.size() > 0) {
            cerr << "'avu_vector':"
            << endl;
            for (vector<Irods_AVU_Type>::iterator iter = avu_vector.begin();
                iter ≠ avu_vector.end();
                ++iter) {
                iter->show();
            }
            cerr << endl;
        } /* if (avu_vector.size() > 0) */
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif DEBUG_COMPILE
    if (result) {
        mysql_free_result(result);
        result = 0;
    }
}

```

```
sqlStrm.str("");
```

1118.

```
<Irods_Object_Type::get_from_database definition 1077> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Exiting 'Irods_Object_Type::get_from_database' successfully\
               with return value " << 'ret_val' <= " << ret_val <= ". " << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return ret_val; } /* End of Irods_Object_Type::get_from_database definition */
```

1119. Update database entry (*update*). [LDF 2013.01.09.]

Log

[LDF 2013.01.09.] Added this function.

```
<Irods_Object_Type function declarations 1039> +≡
int update(MYSQL *mysql_ptr);
```

1120.

```
<Irods_Object_Type::update definition 1120> ≡
int Irods_Object_Type::update(MYSQL *mysql_ptr){
    /* !! TODO: LDF 2013.03.20. Add code for writing to 'Irods_AVUs' table. */
    bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    size_t temp_val = 0;
    stringstream temp_strm;
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Entering 'Irods_Object_Type::update'." << endl;
        show(*this);
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

See also sections 1121, 1122, 1123, 1124, 1125, 1126, 1127, 1128, 1129, 1130, and 1131.

This code is used in section 1258.

1121. Error handling: $id \equiv 0$. [LDF 2013.01.09.]

```
< Irods_Object_Type::update definition 1120 > +≡
  if (id == 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Irods_Object_Type::update':"
    << endl <<
    "'id'==0. Can't update database entry."
    << endl <<
    "Use 'Irods_Object_Type::get_from_database' to set 'id'."
    << endl <<
    "Exiting function unsuccessfully with return value 1."
    << endl;
    unlock_cerr_mutex();
    return 1;
  } /* if (id == 0) */
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "'id'==" << id << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1122.

```
< Irods_Object_Type::update definition 1120 > +≡
  MYSQL_RES * result = 0;
  MYSQL_ROW curr_row;
  unsigned int row_ctr;
  unsigned int field_ctr;
  long int affected_rows;
  stringstream sql_strm;
  int ret_val = 0;
  string temp_str;
  char outstr[200];
  struct tm tmp;
```

1123.

```

⟨ Irods_Object_Type::update definition 1120 ⟩ +≡
    status = mysql_select_db(mysql_ptr, "gwirdsif");
    if (status == 0) {
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "In 'Irods_Object_Type::update': " << "'mysql_select_db' succeeded." << endl <<
                "Selected 'gwirdsif' database successfully." << endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    } /* if (status == 0) */
    else /* status != 0 */
    {
        lock_cerr_mutex();
        cerr << "In 'Irods_Object_Type::update': " << "'mysql_select_db' failed, returning" <<
            status << endl << "Failed to select 'gwirdsif' database:" <<
            endl << "Error: " << mysql_error(mysql_ptr) << endl <<
            "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        return 1;
    } /* else (status != 0) */
}

```

1124.

```

⟨ Irods_Object_Type::update definition 1120 ⟩ +≡
    status = submit_mysql_query("lock_tables_Irods_Objects_write, Irods_Objects_Handles_write",
        result, mysql_ptr);
    if (status != 0) {
        lock_cerr_mutex();
        cerr << "ERROR! In 'Irods_Object_Type::update':" << endl <<
            "'submit_mysql_query' failed, returning" << status << "." << endl <<
            "Failed to lock 'Irods_Objects' and 'Irods_Objects_Handles' tables." << endl <<
            "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        if (result) mysql_free_result(result);
        return 1;
    } /* if (status != 0) */
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "In 'Irods_Object_Type::update':" << endl <<
                "'submit_mysql_query' succeeded, returning 0." << endl <<
                "Locked 'Irods_Objects' and 'Irods_Objects_Handles' tables successfully." << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    mysql_free_result(result);
    result = 0;
}

```

1125.

```
< Irods_Object_Type::update definition 1120 > +≡
  errno = 0;
  last_modified = time(0);
  if (last_modified ≡ static_cast<time_t>(-1)) {
    lock_cerr_mutex();
    cerr ≪ "ERROR! In 'Irods_Object_Type::update': time failed, "
    ≪ "returning (time_t)-1:" ≪ endl ≪ "Error: " ≪ strerror(errno) ≪ endl ≪
    "Will try to unlock tables before exiting function unsuccessfully." ≪ endl;
    unlock_cerr_mutex();
    last_modified = 0;
    ret_val = 1;
    goto UNLOCK_TABLES_UPDATE;
  } /* if */
  if ((gmtime_r(&last_modified, &tmp)) ≡ 0) {
    lock_cerr_mutex();
    cerr ≪ "ERROR! In 'Irods_Object_Type::update': gmtime_r failed, "
    ≪ "returning 0." ≪ endl ≪ "Error: " ≪ strerror(errno) ≪ endl ≪
    "Will try to unlock tables before exiting function unsuccessfully." ≪ endl;
    unlock_cerr_mutex();
    last_modified = 0;
    ret_val = 1;
    goto UNLOCK_TABLES_UPDATE;
  }
  if (strftime(outstr, sizeof(outstr), "%Y-%m-%d %H:%M:%S UTC", &tmp) ≡ 0) {
    lock_cerr_mutex();
    cerr ≪ "ERROR! In 'Irods_Object_Type::update': strftime failed, "
    ≪ "returning 0." ≪ endl ≪ "Will try to unlock tables before exiting "
    ≪ "function unsuccessfully." ≪ endl;
    unlock_cerr_mutex();
    last_modified = 0;
    ret_val = 1;
    goto UNLOCK_TABLES_UPDATE;
  }
```

1126.

```

⟨ Irods_Object_Type::update definition 1120 ⟩ +≡
    temp_strm.str("");
    if (last_modified ≡ created) {
        /* Unset last_modified if it has the same value as created. [LDF 2013.07.21.] */
        last_modified = 0;
    }
    if (last_modified > 0) temp_strm ≪ "from_unixtime(" ≪ last_modified ≪ ")";
    else temp_strm ≪ "0";
    sql_strm ≪ "update_Irods_Objects_set_last_modified_= " ≪ temp_strm.str() ≪ " "
                  ≪ "where_irods_object_id_= " ≪ id;
#endif      /* DEBUG_COMPILE */
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "sql_strm.str()= " ≪ sql_strm.str() ≪ endl;
    unlock_cerr_mutex();
}      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */

```

1127.

```

⟨ Irods_Object_Type::update definition 1120 ⟩ +≡
    temp_strm.str("");
    status = submit_mysql_query(sql_strm.str().c_str(), result, mysql_ptr, 0, 0, &affected_rows);
    if (status ≠ 0) {
        lock_cerr_mutex();
        cerr ≪ "ERROR! In 'Irods_Object_Type::update':"
              ≪ endl ≪
              "'submit_mysql_query' failed, returning " ≪ status ≪ "."
              ≪ endl ≪ "Failed to update 'Irods_Objects' table."
              ≪ endl ≪
              "Will try to unlock tables before exiting function unsuccessfully." ≪ endl;
        unlock_cerr_mutex();
        ret_val = 1;
        if (result) {
            mysql_free_result(result);
            result = 0;
        }
        goto UNLOCK_TABLES_UPDATE;
    }      /* if (status ≠ 0) */
#endif      /* DEBUG_COMPILE */
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ "In 'Irods_Object_Type::update':"
              ≪ endl ≪
              "'submit_mysql_query' succeeded, returning 0."
              ≪ endl ≪
              "Updated 'Irods_Objects' table successfully."
              ≪ endl ≪ "'affected_rows'= "
              ≪ affected_rows ≪ endl;
        unlock_cerr_mutex();
    }      /* else if (DEBUG) */
#endif      /* DEBUG_COMPILE */
mysql_free_result(result);
result = 0;
sql_strm.str("");

```

1128.

```
< Irods_Object_Type::update definition 1120 > +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "handle_id_vector.size() == " << handle_id_vector.size() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    if (handle_id_vector.size() > 0) { sql_strm << "insert into Irods_Objects_Handles(irod"
        s_object_id, handle_id) values ";
    string comma_str = "";
    for (vector<unsigned long int>::const_iterator iter = handle_id_vector.begin();
        iter != handle_id_vector.end(); ++iter) {
        sql_strm << comma_str << "(" << id << ", " << *iter << ")";
        comma_str = ", ";
    } /* for */
#endif /* DEBUG_COMPILE */
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "sql_strm.str() == " << sql_strm.str() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1129.

```
< Irods_Object_Type::update definition 1120 > +≡
status = submit_mysql_query(sql_strm.str().c_str(), result, mysql_ptr, 0, 0, &affected_rows);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ "ERROR! In 'Irods_Object_Type::update':"
    ≪ endl ≪
    "'submit_mysql_query' failed, returning "
    ≪ status ≪ "."
    ≪ endl ≪
    "Failed to insert into 'Irods_Objects_Handles' table."
    ≪ endl ≪
    "Will try to unlock tables before exiting function unsuccessfully."
    ≪ endl;
unlock_cerr_mutex();
ret_val = 1;
if (result) {
    mysql_free_result(result);
    result = 0;
}
goto UNLOCK_TABLES_UPDATE;
} /* if (status ≠ 0) */
#endif DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "In 'Irods_Object_Type::update':"
    ≪ endl ≪
    "'submit_mysql_query' succeeded, returning 0."
    ≪ endl ≪
    "Inserted into 'Irods_Objects_Handles' table successfully."
    ≪ endl ≪
    "'affected_rows' == "
    ≪ affected_rows ≪ endl;
unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
mysql_free_result(result);
result = 0;
sql_strm.str("");
/* if (handle_id_vector.size() > 0) */
```

1130.

```

⟨ Irods_Object_Type::update definition 1120 ⟩ +≡
UNLOCK_TABLES_UPDATE: status = submit_mysql_query("unlock_tables", result, mysql_ptr);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Irods_Object_Type::update':"
        << endl <<
        "'submit_mysql_query' failed, returning"
        << status << "."
        << endl <<
        "Failed to unlock_tables ('Irods_Objects' and 'Irods_Objects_Handles' were locked"
        << d)."
        << endl << "Exiting function unsuccessfully with return value 1."
        << endl;
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    return 1;
} /* if (status ≠ 0) */
#endif /* DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'Irods_Object_Type::update':"
        << endl <<
        "'submit_mysql_query' succeeded, returning 0."
        << endl <<
        "Unlocked_tables successfully. ('Irods_Objects' and 'Irods_Objects_Handles' "
        << "tables were locked.)"
        << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
mysql_free_result(result);
result = 0;

```

1131.

```

⟨ Irods_Object_Type::update definition 1120 ⟩ +≡
if (ret_val ≠ 0) {
    lock_cerr_mutex();
    cerr << "Exiting 'Irods_Object_Type::update' unsuccessfully with"
        << "return value"
        << ret_val << "."
        << endl;
    unlock_cerr_mutex();
    return ret_val;
} /* if (ret_val ≠ 0) */
#endif /* DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "Exiting 'Irods_Object_Type::update' successfully with return value 0."
        << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
return 0; } /* End of Irods_Object_Type::update definition */

```

1132. Put iRODS object (*put_irods_object*). [LDF 2013.03.07.]

Log

[LDF 2013.03.07.] Added this function.
 [LDF 2013.03.22.] Added argument **string** *irods_env_filename*.

<Irods_Object_Type function declarations 1039> +≡

```
int put_irods_object(string filename, string irods_env_filename, bool force = false);
```

1133.

<Irods_Object_Type::put_irods_object definition 1133> ≡

```
int Irods_Object_Type::put_irods_object(string filename, string irods_env_filename, bool force){
    bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    stringstream temp_strm;
    char buffer[1024];
    memset(buffer, 0, 1024);
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Entering 'Irods_Object_Type::put_irods_object'." << endl;
        show(*this);
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

See also sections 1134, 1135, 1136, 1137, 1138, and 1139.

This code is used in section 1258.

1134.

```

⟨ Irods_Object_Type::put_irods_object definition 1133 ⟩ +≡
    temp_strm << "env_irodsEnvFile=" << irods_env_filename << "input";
    if (force) temp_strm << "-f";
    temp_strm << "," << filename << ',' << path << ",";
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'Irods_Object_Type::put_irods_object':" << endl << "'temp_strm.str()'" <<
            temp_strm.str() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    status = system(temp_strm.str().c_str());
    if (status == -1 || !WIFEXITED(status)) {
        lock_cerr_mutex();
        cerr << "ERROR! In 'Irods_Object_Type::put_irods_object':"
            << "system' failed, returning" << status << "." << endl;
        if (WIFEXITED(status)) cerr << "WEXITSTATUS(status)" << "=" << WEXITSTATUS(status) << endl;
        else cerr << "Process failed to exit." << endl;
        cerr << "Exiting 'Irods_Object_Type::put_irods_object'" <<
            "unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        return 1;
    } /* (status == -1 || !WIFEXITED(status)) */
}

```

1135.

```

⟨ Irods_Object_Type::put_irods_object definition 1133 ⟩ +≡
else
    if (WEXITSTATUS(status) != 0) {
        lock_cerr_mutex();
        cerr << "ERROR! In 'Irods_Object_Type::put_irods_object':"
            << "input" << command << "(called via 'system') failed, "
            << "returning" << WEXITSTATUS(status) << "." << endl <<
            "Exiting 'Irods_Object_Type::put_irods_object'" <<
            "unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        return 1;
    } /* else if (WEXITSTATUS(status) != 0) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'Irods_Object_Type::put_irods_object':" << endl <<
            "'system' succeeded, returning 0." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1136.

```
< Irods_Object_Type::put_irods_object definition 1133 > +≡
  if (avu_vector.size() > 0) { temp_strm.str("");
  temp_strm << "echo\"";
  for (vector<Irods_AVU_Type>::iterator iter = avu_vector.begin(); iter != avu_vector.end(); ++iter)
  {
    temp_strm << "add-d" << path << ',' << iter->attribute << ',' << " " << iter->value << ",";
    if (!iter->units.empty()) {
      temp_strm << " " << iter->units << ",";
    }
    temp_strm << "\n";
  } /* for */
  temp_strm << "\"|envirodsEnvFile=" << irods_env_filename << "imeta>/dev/null2>/dev/null";
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "temp_strm.str() == " << temp_strm.str() << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1137.

```
< Irods_Object_Type::put_irods_object definition 1133 > +≡
  status = system(temp_strm.str().c_str());
  if (status == -1 || !WIFEXITED(status)) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Irods_Object_Type::put_irods_object': "
    << "'system' failed, returning" << status << "." << endl;
    if (WIFEXITED(status)) cerr << "WEXITSTATUS(status) == " << WEXITSTATUS(status) << endl;
    else cerr << "Process failed to exit." << endl;
    cerr << "Exiting 'Irods_Object_Type::put_irods_object' "
    << "unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    return 1;
  } /* (status == -1 || !WIFEXITED(status)) */
```

1138.

```

⟨ Irods_Object_Type::put_irods_object definition 1133 ⟩ +≡
else
  if (WEXITSTATUS(status) ≠ 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Irods_Object_Type::put_irods_object':"
    << "'imeta' command via 'system' failed, "
    << "returning " << WEXITSTATUS(status) << "."
    << endl <<
    "Exiting 'Irods_Object_Type::put_irods_object':"
    << "unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    return 1;
  } /* else if (WEXITSTATUS(status) ≠ 0) */
#endif DEBUG_COMPILE
else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'Irods_Object_Type::put_irods_object':"
    << endl <<
    "'system' succeeded, returning 0."
    << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if */

```

1139.

```

⟨ Irods_Object_Type::put_irods_object definition 1133 ⟩ +≡
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "Exiting 'Irods_Object_Type::put_irods_object':"
    << " successfully\n"
    << " with return value 0."
    << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  return 0; } /* End of Irods_Object_Type::put_irods_object definition */

```

1140. Add AVU (*add_avu*). [LDF 2013.03.08.]

Log

[LDF 2013.03.08.] Added this function.

[LDF 2013.03.22.] Added argument **string irods_env_filename**.

[LDF 2013.08.16.] Added arguments and code for pushing the AVU onto *avu_vector* and/or writing it to the database.

⟨ Irods_Object_Type function declarations 1039 ⟩ +≡

```

int add_avu(Irods_AVU_Type avu, string irods_env_filename, bool call_imeta = true, bool
push_onto_vector = true, bool database = true, MYSQL * mysql_ptr = 0, int thread_ctr = 0);

```

1141.

```

⟨ Irods_Object_Type::add_avu definition 1141 ⟩ ≡
int Irods_Object_Type::add_avu(Irods_AVU_Type avu, string irods_env_filename, bool
    call_imeta, bool push_onto_vector, bool database, MYSQL * mysql_ptr, int thread_ctr){ bool
    DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    stringstream temp_strm;
    char buffer[1024];
    memset(buffer, 0, 1024);
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Entering 'Irods_Object_Type::add_avu'." << endl;
        show(*this);
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

See also sections 1142, 1143, 1144, 1145, 1146, 1147, and 1148.

This code is used in section 1258.

1142.

```

⟨ Irods_Object_Type::add_avu definition 1141 ⟩ +≡
if (database ≡ true ∧ mysql_ptr ≡ 0) {
    lock_cerr_mutex();
    cerr << "WARNING! In 'Irods_Object_Type::add_avu':"
    << endl <<
    "'database' == 'true', but 'mysql_ptr' == NULL." << endl <<
    "Resetting 'database' to 'false' and continuing." << endl;
    unlock_cerr_mutex();
    database = false;
} /* if (database ≡ true ∧ mysql_ptr ≡ 0) */

```

1143.

Log

[LDF 2013.08.16.] Added this section.

```

⟨ Irods_Object_Type::add_avu definition 1141 ⟩ +≡
if (call_imeta ≡ false) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'Irods_Object_Type::add_avu':"
        << endl <<
        "'call_imeta' == 'false'. Skipping to 'END_CALL_IMETA'." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    goto END_CALL_IMETA;
} /* if (call_imeta ≡ false) */

```

1144.

```

⟨ Irods_Object_Type::add_avu definition 1141 ⟩ +≡
    temp_strm << "env\irodsEnvFile=" << irods_env_filename << "\imeta\add\d\path" << path << "\"
        << avu.attribute << "\," << avu.value << ",";
#endif /* DEBUG_COMPILE */
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "temp_strm.str() == " << temp_strm.str() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    status = system(temp_strm.str().c_str());
    if (status == -1 || !WIFEXITED(status)) {
        lock_cerr_mutex();
        cerr << "ERROR! In 'Irods_Object_Type::add_avu': system failed, returning" <<
            status << endl;
        if (WIFEXITED(status)) cerr << "WEXITSTATUS(status) == " << WEXITSTATUS(status) << endl;
        else cerr << "Process failed to exit." << endl;
        cerr << "Exiting 'Irods_Object_Type::add_avu'" <<
            "unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        return 1;
    } /* (status == -1 || !WIFEXITED(status)) */
}

```

1145.

```

⟨ Irods_Object_Type::add_avu definition 1141 ⟩ +≡
    else
        if (WEXITSTATUS(status) != 0) {
            lock_cerr_mutex();
            cerr << "ERROR! In 'Irods_Object_Type::add_avu': "
                "\imeta\add\command\called\via\`system'\failed, returning"
                WEXITSTATUS(status) << endl << "Exiting 'Irods_Object_Type::add_avu'" <<
                    "unsuccessfully with return value 1." << endl;
            unlock_cerr_mutex();
            return 1;
        } /* else if (WEXITSTATUS(status) != 0) */
#endif /* DEBUG_COMPILE */
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "In 'Irods_Object_Type::add_avu': "
                "'system' succeeded, returning 0." << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
END_CALL_IMETA:

```

1146.**Log**

[LDF 2013.08.16.] Added this section.

```
< Irods_Object_Type::add_avu definition 1141 > +≡
  if (push_onto_vector) avu_vector.push_back(avu);
```

1147.**Log**

[LDF 2013.08.16.] Added this section.

```
< Irods_Object_Type::add_avu definition 1141 > +≡
  if (database) {
    status = avu.write_to_database(mysql_ptr, thread_ctr);
    if (status ≠ 0) {
      lock_cerr_mutex();
      cerr << "ERROR! In 'Irods_Object_Type::add_avu': "
          << "'Irods_AVU_Type::write_to_database' failed, returning "
          << status << "."
          << "Exiting 'Irods_Object_Type::add_avu' "
          << "unsuccessfully with return value 1."
          << endl;
      unlock_cerr_mutex();
      return 1;
    } /* if (status ≠ 0) */
  #if DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "In 'Irods_Object_Type::add_avu': "
          << "'Irods_AVU_Type::write_to_database' succeeded, returning 0."
          << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
  #endif /* DEBUG_COMPILE */
  } /* if (database) */
```

1148.

```
< Irods_Object_Type::add_avu definition 1141 > +≡
  #if DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "Exiting 'Irods_Object_Type::add_avu' successfully with return value 0."
    unlock_cerr_mutex();
  } /* if (DEBUG) */
  #endif /* DEBUG_COMPILE */
  return 0; } /* End of Irods_Object_Type::add_avu definition */
```

1149. Add AVU conditionally (*add_avu_cond*). [LDF 2013.08.16.]

Log

[LDF 2013.08.16.] Added this function.

```
< Irods_Object_Type function declarations 1039 > +≡
int add_avu_cond(string irods_env_filename, string check_attrib, string check_val, string
new_attrib, MYSQL * mysql_ptr = 0, string new_val = "", bool database = true, bool
push_onto_vector = true, int thread_ctr = 0);
```

1150.

```
< Irods_Object_Type::add_avu_cond definition 1150 > ≡
int Irods_Object_Type::add_avu_cond(string irods_env_filename, string check_attrib, string
check_val, string new_attrib, MYSQL * mysql_ptr, string new_val, bool push_onto_vector, bool
database, int thread_ctr){ bool DEBUG = false; /* true */
set_debug_level(DEBUG, 0, 0);
int status;
#ifndef DEBUG_COMPILE
if (DEBUG) {
lock_cerr_mutex();
cerr << "Entering 'Irods_Object_Type::add_avu_cond'." << endl;
unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

See also sections 1151 and 1152.

This code is used in section 1258.

1151.

```

⟨ Irods_Object_Type::add_avu_cond definition 1150 ⟩ +≡
    vector<Irods_AVU_Type>::iterator iter = find_avu(check_attrib, check_val);
    Irods_AVU_Type curr_avu;
    bool call_imeta;

    if (iter != avu_vector.end()) {
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "In 'Irods_Object_Type::add_avu_cond':"
                << endl <<
                "Matching AVU found. Will call 'Irods_AVU_Type::add_avu'." << endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        curr_avu = *iter;
        curr_avu.attribute = new_attrib;
        curr_avu.value = (new_val.empty()) ? check_val : new_val;
        time_t curr_time = time(0);

        if (deleted_from_archive ∨ (marked_for_deletion_from_archive ∧ delete_from_archive_timestamp <
            curr_time - purge_irods_archive_limit)) call_imeta = false;
        else call_imeta = true;
        status = add_avu(curr_avu, irods_env_filename, call_imeta, push_onto_vector, database, mysql_ptr,
            thread_ctr);
        if (status ≠ 0) {
            lock_cerr_mutex();
            cerr << "ERROR! In 'Irods_Object_Type::add_avu_cond':"
                << endl <<
                "'Irods_AVU_Type::add_avu' failed, returning " << status << "."
                << endl <<
                "Exiting function unsuccessfully with return value -1." << endl;
            unlock_cerr_mutex();
            return -1;
        } /* if (status ≠ 0) */
#endif DEBUG_COMPILE
        else
            if (DEBUG) {
                lock_cerr_mutex();
                cerr << "In 'Irods_Object_Type::add_avu_cond':"
                    << endl <<
                    "'Irods_AVU_Type::add_avu' succeeded, returning 0."
                    << endl <<
                    "Exiting function successfully with return value 1." << endl;
                unlock_cerr_mutex();
            } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
        return 1;
    } /* if (iter ≠ avu_vector.end()) */

```

1152.

```
< Irods_Object_Type::add_avu_cond definition 1150 > +≡
  else {
#define DEBUG_COMPILE
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "In 'Irods_Object_Type::add_avu_cond':" << endl <<
        "No matching AVU found. Not adding AVU." << endl <<
        "Exiting function successfully with return value 0." << endl;
      unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0;
} /* else */
} /* Irods_Object_Type::add_avu_cond */
```

1153. Mark for deletion (*mark_for_deletion*). [LDF 2013.08.07.]

Log

[LDF 2013.08.07.] Added this function.

[LDF 2013.08.08.] Added argument **time_t** &**save_delay**.

```
< Irods_Object_Type function declarations 1039 > +≡
static int mark_for_deletion(vector<Irods_Object_Type> &irods_object_vector, MYSQL * &mysql_ptr,
  Response_Type &response, int user_id, string irods_env_filename, time_t &save_delay, string
  thread_str = "", bool wake_purge_thread = false);
```

1154.

```

⟨Irods_Object_Type::mark_for_deletion definition 1154⟩ ≡
    int Irods_Object_Type::mark_for_deletion(vector<Irods_Object_Type> &irods_object_vector,
                                              MYSQL *&mysql_ptr, Response_Type &response, int user_id, string irods_env_filename, time_t
                                              &save_delay, string thread_str, bool wake_purge_thread){ bool DEBUG = false; /* true */
        set_debug_level(DEBUG, 0, 0);
        stringstream sql_strm;
        stringstream sql_strm_1;
        MYSQL_RES *result = 0;
        MYSQL_RES *result_1 = 0;
#ifndef 0
        MYSQL_ROW curr_row;
        unsigned int row_ctr;
        unsigned int field_ctr;
#endif
#ifndef
        long int affected_rows;
        long int affected_rows_1;
        int status = 0;
        int ret_val = -1;
        FILE *fp = 0;
#endif DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_str << "Entering 'Irods_Object_Type::mark_for_deletion'." << endl;
            response.show("response:");
            cerr << "response.delay_value=" << response.delay_value << endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

See also sections 1155, 1156, 1157, 1158, 1159, 1160, 1161, 1162, 1163, 1164, 1165, 1166, 1167, 1168, 1169, 1170, 1171, and 1172.
This code is used in section 1258.

1155.

```

⟨Irods_Object_Type::mark_for_deletion definition 1154⟩ +≡
    if (irods_object_vector.size() == 0) {
        lock_cerr_mutex();
        cerr << thread_str << "WARNING! In 'Irods_Object_Type::mark_for_deletion':"
        << endl << "'irods_object_vector.size()=='0'. No 'Irods_Object_Type' objects"
        << "to mark for deletion." << endl << "Exiting function unsuccessfully with re"
        << "turn value 2." << endl;
        unlock_cerr_mutex();
        return 2;
    } /* if (irods_object_vector.size() == 0) */

```

1156.

```
< Irods_Object_Type :: mark_for_deletion definition 1154 > +≡
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "irods_object_vector.size() == " << irods_object_vector.size() << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1157. *response.delay_value* will be 1 if the **rm** command has been used with the **delay** option without an argument. In this case, the iRODS object will be deleted by *purge_server_database* after *purge_database_interval* seconds.

It is not possible to distinguish between **delay** with no argument and **delay** with 1 second as its argument. However, a delay of 1 second is of no practical use, so I think it's better to use 1 as described. Otherwise, some other value would have to be "sacrificed" in this way. [LDF 2013.08.08.]

```
< Irods_Object_Type :: mark_for_deletion definition 1154 > +≡
    errno = 0;
    time_t temp_time_val = time(0);
    time_t curr_time;
    if (temp_time_val == static_cast<time_t>(-1)) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'Irods_Object_Type::mark_for_deletion':"
        << endl << "'time' failed, returning (time_t)-1:" << endl <<
        "Error_number: " << errno << ":" << endl << strerror(errno) << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        return 1;
    } /* if (temp_time_val == static_cast<time_t>(-1)) */
```

1158. The value used below 31622400 = Number of seconds in a year + 1 day (366 days). This way, it will be possible to identify iRODS objects entries in the `gwirdsif.Irods_Objects` database that have been marked for immediate deletion easily, because the value in the `delete_from_gwirdsif_db` field will be more than a year older than the current date/time. [LDF 2013.08.12.]

```

⟨ Irods_Object_Type::mark_for_deletion definition 1154 ⟩ +≡
curr_time = temp_time_val;
if (response.delay_value ≡ 1) {
    save_delay = temp_time_val;
}
else if (response.delay_value > 1) {
    temp_time_val += static_cast<time_t>(response.delay_value);
    save_delay = temp_time_val;
}
else /* Mark iRODS objects for immediate deletion */
{
    if (temp_time_val ≤ purge_irods_archive_limit ∨ (temp_time_val - purge_irods_archive_limit) ≤ 31622400)
    {
        /* Error handling. [LDF 2013.08.14.] */
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'Irods_Object_Type::mark_for_deletion':"
        << endl <<
        "'temp_time_val' and/or 'purge_irods_archive_limit' has invalid value:" <<
        endl << "'temp_time_val' <= 'purge_irods_archive_limit'" << endl <<
        "or 'temp_time_val' - 'purge_irods_archive_limit' <= 31622400" << endl <<
        "'temp_time_val' == ....." << temp_time_val << endl <<
        "'purge_irods_archive_limit' == ....." << purge_irods_archive_limit <<
        endl << "'temp_time_val' - 'purge_irods_archive_limit' =="
        (temp_time_val - purge_irods_archive_limit) << endl <<
        "Invalid timestamp value for immediate deletion." <<
        endl << "This shouldn't ever happen." << endl <<
        "'purge_irods_archive_limit' probably has a bad value:" << endl <<
        "Exiting unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        return 1;
    } /* if */
    else {
        temp_time_val -= purge_irods_archive_limit + 31622400;
    }
} /* else */
sql_strm << "update_gwirdsif.Irods_Objects set ";
sql_strm_1 << "update_gwirdsif.Irods_Objects set ";
if (¬(response.options & 4_U)) {
    sql_strm << "marked_for_deletion_from_archive=1," <<
    "delete_from_archive_timestamp=" << "from_unixtime(" << temp_time_val << "),";
}
if (response.options & 2_U ∨ response.options & 4_U) {
    sql_strm_1 << "marked_for_deletion_from_gwirdsif_db=1," <<
    "delete_from_gwirdsif_db_timestamp=" << "from_unixtime(" << temp_time_val << "),";
}
sql_strm << "last_modified=now()" << "where irods_object_id in (";
sql_strm_1 << "last_modified=now()" << "where irods_object_id in (";
string comma_str = "";

```

```

for (vector<Irods_Object_Type>::iterator iter = irods_object_vector.begin();
      iter != irods_object_vector.end(); ++iter) {
#if DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        iter->show();
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    sql_strm << comma_str << iter->id;
    sql_strm_1 << comma_str << iter->id;
    comma_str = ",";
} /* for */
sql_strm << ") and marked_for_deletion_from_archive=0 and deleted_from_archive=0";
sql_strm_1 << ") and marked_for_deletion_from_gwiridsif_db=0";
#if DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "sql_strm.str()=" << sql_strm.str() << endl << "sql_strm_1.str()=" <<
            sql_strm_1.str() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1159.

```

⟨Irods_Object_Type::mark_for_deletion definition 1154⟩ +≡
status = submit_mysql_query(sql_strm.str(), result, mysql_ptr, 0, 0, &affected_rows);
if (status ≡ 0) status = submit_mysql_query(sql_strm_1.str(), result_1, mysql_ptr, 0, 0, &affected_rows_1);
if (status ≠ 0) {
    cerr << thread_str << "ERROR! In 'Irods_Object_Type::mark_for_deletion':" <<
        endl << "'submit_mysql_query' failed, returning" << status << "." << endl <<
        "Failed to update 'gwiridsif.Irods_Objects' database table." << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    if (result_1) mysql_free_result(result_1);
    return 1;
} /* if (status ≠ 0) */

```

1160.

```
< Irods_Object_Type :: mark_for_deletion definition 1154 > +≡
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_str << "In 'Irods_Object_Type::mark_for_deletion':" << endl <<
                "'submit_mysql_query' succeeded, returning 0." << endl << "'affected_rows'" <<
                affected_rows << endl << "'affected_rows_1'" << endl << affected_rows_1 << endl <<
                "Updated 'gwiridsif.Irods_Objects' database table successfully." << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
mysql_free_result(result);
mysql_free_result(result_1);
result = 0;
result_1 = 0;
```

1161. Create AVUs. [LDF 2013.08.09.]

This should be done even if *response.delay_value* \equiv 0 (“immediate deletion”), because the purging might have been suppressed, i.e., the thread for running *purge_irods_archive* might not have been started. [LDF 2013.08.14.]

!! PLEASE NOTE: The rows in *gwirdsif.Irods_Objects* are updated, i.e., the fields *marked_for_deletion_from_archive* and/or *marked_for_deletion_from_gwirdsif_database* are set to true and the corresponding timestamps are updated as applicable, but no rows are added to *Irods_AVUs*. Under normal circumstances, the iRODS objects will either be deleted by *purge_irods_archive* or undeleted by the user, so that any new rows in *Irods_AVUs* would normally only exist for a short time before being deleted. At present, I don’t see any need to create any, especially considering that they would contain no information that isn’t already stored in the *Irods_Objects* table. [LDF 2013.08.19.]

Log

[LDF 2013.08.09.] Added this section.

```

⟨ Irods_Object_Type::mark_for_deletion definition 1154 ⟩ +≡
stringstream cmnd_strm;
cmnd_strm << "export_irodsEnvFile=" << irods_env_filename << ";" _a='';
string save_delay_str = convert_seconds(save_delay, true);
string and_str = "";
bool found = false;
for (vector<Irods_Object_Type>::iterator iter = irods_object_vector.begin();
     iter != irods_object_vector.end(); ++iter) {
    if (iter->deleted_from_archive == false & !(response.options & 4_U)) {
        found = true;
        cmnd_strm << and_str << "imeta_add_d\" " << iter->path << "\" <<
            "\\" MARKED_FOR_DELETION_FROM_ARCHIVE\\" " << "\" << save_delay << " " <<
            save_delay_str << "\"2>&1";
        and_str = " _&& ";
    }
    if (iter->deleted_from_archive == false & (response.options & 2_U <| response.options & 4_U)) {
        found = true;
        cmnd_strm << and_str << "imeta_add_d\" " << iter->path << "\" <<
            "\\" MARKED_FOR_DELETION_FROM_GWIRDSIF_DATABASE\\" " << "\" << save_delay << " " <<
            save_delay_str << "\"2>&1";
        and_str = " _&& ";
    }
} /* for */
cmnd_strm << "' ;_echo $? ;_echo \$a";
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Irods_Object_Type::mark_for_deletion':" << endl <<
        "' cmnd_strm.str()' == " << cmnd_strm.str() << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1162.

Log

[LDF 2013.08.12.] Added this section.

```

⟨ Irods_Object_Type::mark_for_deletion definition 1154 ⟩ +≡
    if (found == false) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Irods_Object_Type::mark_for_deletion':" << endl <<
            "'found' == 'false'." << iRODS_object(s)_already_deleted_from_archive." << endl <<
            "Not creating AVUs." << endl << "Skipping to 'END_MARK_FOR_DELETION'." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    goto END_AVUS_MARK_FOR_DELETION;
} /* if (found == false) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "cmnd_strm.str() == " << cmnd_strm.str() << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
errno = 0;
fp = popen(cmnd_strm.str().c_str(), "r");
if (fp == 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Irods_Object_Type::mark_for_deletion':" << endl <<
        "'popen' failed, returning 0." << endl;
    if (errno != 0)
        cerr << "Error_number: " << errno << endl << "Error: " << strerror(errno) << endl;
    cerr << "Exiting function unsuccessfully with return value 3." << endl;
    unlock_cerr_mutex();
    return 3;
} /* if (fp == 0) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Irods_Object_Type::mark_for_deletion':" << endl <<
            "'popen' succeeded." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
ret_val = -1;
errno = 0;
status = fscanf(fp, "%d", &ret_val);
if (status != 1) {
    lock_cerr_mutex();
}

```

```
cerr << thread_str << "ERROR! In 'Irods_Object_Type::mark_for_deletion':" <<
endl << "'fscanf' failed, returning" << status << "." << endl <<
"Failed to read return value of 'imeta' command." << endl;
if (errno != 0)
    cerr << "Error_number:" << errno << endl << "Error: " << strerror(errno) << endl;
    cerr << "Exiting function unsuccessfully with return value 3." << endl;
    unlock_cerr_mutex();
    pclose(fp);
    return 3;
} /* if (status != 1) */
#if DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "'ret_val' == " << ret_val << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1163.

```

⟨Irods_Object_Type::mark_for_deletion definition 1154⟩ +≡
if (ret_val ≠ 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Irods_Object_Type::mark_for_deletion':"
        << endl <<
        "'imeta' command(s) called via 'popen' failed, returning "
        << ret_val << "." << endl <<
        "Failed to add AVUs to iRODS objects." << endl;
    unlock_cerr_mutex();
    char buffer[1024];
    memset(buffer, 0, 1024);
    status = fread(buffer, 1, 1024, fp);
    if (status ≡ 0) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'Irods_Object_Type::mark_for_deletion':"
            << endl << "'fread' failed, returning 0:" << endl <<
            "Failed to read error output from 'imeta' command(s) called via 'popen'."
            << endl;
        unlock_cerr_mutex();
    }
    else if (status ≡ 1024) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'Irods_Object_Type::mark_for_deletion':"
            << endl << "'fread' returned 1024: Error output exceeded maximum amount (1023 characters)."
            << endl <<
            "Too much error output from 'imeta' command(s) called via 'popen'." <<
            endl << "This isn't permitted." << endl;
        unlock_cerr_mutex();
    }
    else {
        lock_cerr_mutex();
        cerr << "Error output from 'imeta' commands called via 'popen':"
            << endl << buffer << endl;
        unlock_cerr_mutex();
    }
    lock_cerr_mutex();
    cerr << "Exiting function unsuccessfully with return value 3." << endl;
    unlock_cerr_mutex();
    pclose(fp);
    return 3;
} /* if (ret_val ≠ 0) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Irods_Object_Type::mark_for_deletion':"
            << endl <<
            "'imeta' command(s) called via 'popen' succeeded, returning 0."
            << endl <<
            "Added AVUs to iRODS objects successfully." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
pclose(fp);
fp = 0; END_AVUS_MARK_FOR_DELETION:

```

1164.

Log

[LDF 2013.08.12.] Added this section.

```
< Irods_Object_Type::mark_for_deletion definition 1154 > +≡
map<unsigned long int,
    Handle_Value_Type>::iterator hv_iter; for (vector<Irods_Object_Type>::iterator
iter = irods_object_vector.begin(); iter ≠ irods_object_vector.end(); ++iter) {
#if DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ "iter->handle_vector.size() == " ≪ iter->handle_vector.size() ≪ endl;
        cerr ≪ "iter->handle_id_vector.size() == " ≪ iter->handle_id_vector.size() ≪ endl;
        cerr ≪ "iter->handle_name_string_vector.size() == " ≪
            iter->handle_name_string_vector.size() ≪ endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1165.

Log

[LDF 2013.08.21.] BUG FIX: Added this section.

```
< Irods_Object_Type::mark_for_deletion definition 1154 > +≡
if (iter->handle_vector.size() ≡ 0) {
#if DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ thread_str ≪ "In 'Irods_Object_Type::mark_for_deletion':"
            ≪ endl ≪ "'iter->handle_vector.size()' == 0." ≪
        "Not calling Handle_Type::fetch_handles_from_database." ≪ endl ≪
        "Continuing." ≪ endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    continue;
} /* if (iter->handle_vector.size() ≡ 0) */
```

1166.

```
< Irods_Object_Type::mark_for_deletion definition 1154 > +≡
status = Handle_Type::fetch_handles_from_database(mysql_ptr, iter->handle_id_vector,
    iter->handle_vector, "", thread_str);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << thread_str << "WARNING! In 'Irods_Object_Type::mark_for_deletion':"
        endl << "'Handle_Type::fetch_handles_from_database' failed, "
        endl << "returning"
        status << "."
        endl << "Failed to fetch handle(s) from database."
        endl << "Will try to continue."
        endl;
    unlock_cerr_mutex();
    iter->handle_vector.clear();
    continue;
} /* if (status ≠ 0) */
```

1167.

```
< Irods_Object_Type::mark_for_deletion definition 1154 > +≡
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Irods_Object_Type::mark_for_deletion':"
            endl << "'Handle_Type::fetch_handles_from_database' succeeded, "
            endl << "returning 0."
            endl << "Fetched handle(s) from database successfully."
            endl << "'iter->handle_vector.size()' == "
            endl << iter->handle_vector.size() << endl;
        if (iter->handle_vector.size() > 0) cerr << "'iter->handle_vector':"
            endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1168.

```
< Irods_Object_Type::mark_for_deletion definition 1154 > +≡
for (vector<Handle_Type>::iterator iter_1 = iter->handle_vector.begin();
    iter_1 ≠ iter->handle_vector.end(); ++iter_1) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        iter_1->show();
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1169.

```
< Irods_Object_Type::mark_for_deletion definition 1154 > +≡
  hv_iter = iter_1→find("IRODS_OBJECT");
  if (hv_iter ≠ iter_1→handle_value_map.end()) {
    status = 0;
    if (response.options & 2_U ∨ response.options & 4_U) {
      status = iter_1→add_value(mysql_ptr,
        Handle_Value_Type::IRODS_OBJECT_MARKED_FOR_DELETION_FROM_GWIRDSIF_DB_INDEX,
        "IRODS_OBJECT_MARKED_FOR_DELETION_FROM_GWIRDSIF_DB", convert_seconds(save_delay),
        hv_iter→second.created_by_user_id);
    }
    if (status ≡ 0 ∧ ¬(response.options & 4_U)) {
      status = iter_1→add_value(mysql_ptr,
        Handle_Value_Type::IRODS_OBJECT_MARKED_FOR_DELETION_FROM_ARCHIVE_INDEX,
        "IRODS_OBJECT_MARKED_FOR_DELETION_FROM_ARCHIVE", convert_seconds(save_delay),
        hv_iter→second.created_by_user_id);
    }
    if (status ≠ 0) {
      lock_cerr_mutex();
      cerr ≪ thread_str ≪ "WARNING! In 'Irods_Object_Type::mark_for_deletion':"
      endl ≪ "'Handle_Type::add_value' failed, returning"
      ≪ status ≪ "."
      ≪ endl ≪ "Failed to add handle value to handle."
      ≪ endl ≪ "Will try to continue."
      ≪ endl;
      unlock_cerr_mutex();
    }
  } /* if (hv_iter ≠ iter_1→handle_vector.end()) */
```

1170.

```

⟨Irods_Object_Type::mark_for_deletion definition 1154⟩ +≡
    hv_iter = iter_1→find("DC_METADATA_IRODS_OBJECT");
    if (hv_iter ≠ iter_1→handle_value_map.end()) {
        status = 0;
        if (response.options & 2_U ∨ response.options & 4_U) {
            status = iter_1→add_value(mysql_ptr,
                Handle_Value_Type::DC_METADATA_IRODS_OBJECT_MARKED_FOR_DELETION_FROM_GWIRDSIF_DB_INDE█,
                "DC_METADATA_IRODS_OBJECT_MARKED_FOR_DELETION_FROM_GWIRDSIF_DB",
                convert_seconds(save_delay), hv_iter→second.created_by_user_id);
        }
        if (status ≡ 0 ∧ ¬(response.options & 4_U)) {
            status = iter_1→add_value(mysql_ptr,
                Handle_Value_Type::DC_METADATA_IRODS_OBJECT_MARKED_FOR_DELETION_FROM_ARCHIVE_INDE█,
                "DC_METADATA_IRODS_OBJECT_MARKED_FOR_DELETION_FROM_ARCHIVE",
                convert_seconds(save_delay), hv_iter→second.created_by_user_id);
        }
        if (status ≠ 0) {
            lock_cerr_mutex();
            cerr ≪ thread_str ≪ "WARNING! In 'Irods_Object_Type::mark_for_deletion':"
            endl ≪ "'Handle_Type::add_value' failed, returning"
            ≪ status ≪ "."
            ≪ endl ≪ "Failed to add handle value to handle."
            ≪ endl ≪ "Will try to continue."
            ≪ endl;
            unlock_cerr_mutex();
        }
    } /* if (hv_iter ≠ iter_1→handle_vector.end()) */
} /* Inner for */
#endif /* DEBUG_COMPILE */
if (DEBUG) {
    lock_cerr_mutex();
    if (iter→handle_vector.size() > 0) cerr ≪ endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* Outer for */

```

1171.

```

⟨Irods_Object_Type::mark_for_deletion definition 1154⟩ +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "NOTICE: In 'Irods_Object_Type::mark_for_deletion':"
            << endl <<
            "'wake_purge_thread'=="
            << wake_purge_thread << endl;
        if (wake_purge_thread & response.delay_value == 0)
            cerr << "'wake_purge_thread'==true and 'response.delay_value'==0:"
                << endl <<
                "Will_wake_purge_thread_for_immediate_deletion."
            << endl;
        else cerr << "'wake_purge_thread'=="
            << wake_purge_thread << " "
            << "and 'response.delay_value'=="
            << response.delay_value << ":" << endl <<
            "Will_not_wake_purge_thread_for_immediate_deletion."
            << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    if (wake_purge_thread & response.delay_value == 0) {
        if (purge_irods_archive_thread_id == static_cast<pthread_t>(0)) {
            lock_cerr_mutex();
            cerr << thread_str << "NOTICE: In 'Irods_Object_Type::mark_for_deletion':"
                << endl <<
                "'purge_irods_archive_thread_id'==0:"
                << endl <<
                "\\"Purge_iRODS_archive_thread\\not_running."
                << endl <<
                "iRODS_objects_marked_for_immediate_deletion_will_"
                <<
                "be_deleted_the_next_time_gwirdsif_is_started_with"
                << endl <<
                "purging_the_iRODS_archive_enabled."
                << endl <<
                "Not_calling_pthread_cond_signal'. Continuing."
                << endl;
            unlock_cerr_mutex();
        } /* if */
        else {
#ifndef DEBUG_COMPILE
            if (DEBUG) {
                lock_cerr_mutex();
                cerr << thread_str << "In 'Irods_Object_Type::mark_for_deletion':"
                    << endl <<
                    "'purge_irods_archive_thread_id'!=0:"
                    << endl <<
                    "\\"Purge_iRODS_archive_thread\\running\\"
                    <<
                    "Calling_pthread_cond_signal_to_wake_it_up."
                    << endl;
                unlock_cerr_mutex();
            } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
            pthread_mutex_lock(&purge_irods_archive_mutex);
            status = pthread_cond_signal(&purge_irods_archive_cond);
            if (status != 0) {
                lock_cerr_mutex();
                cerr << thread_str << "ERROR! In 'Irods_Object_Type::mark_for_deletion':"
                    << endl <<
                    "'pthread_cond_signal'_failed,_returning_"
                    << status << ":" << endl <<
                    "Error:\\"
                    << strerror(status) << endl <<
                    "Exiting_function_unsuccessfully_with_return_value_1."
                    << endl;
                unlock_cerr_mutex();
                return 1;
            }
#ifndef DEBUG_COMPILE
            else

```

```

if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Irods_Object_Type::mark_for_deletion':" << endl <<
        "'pthread_cond_signal' succeeded, returning 0." << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    pthread_mutex_unlock(&purge_irods_archive_mutex);
} /* else */
} /* if (wake_purge_thread & response.delay_value == 0) */

```

1172.

```

⟨Irods_Object_Type::mark_for_deletion definition 1154⟩ +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Exiting 'Irods_Object_Type::mark_for_deletion'" <<
            "successfully with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; } /* End of Irods_Object_Type::mark_for_deletion definition */

```

1173. Delete from archive (*delete_from_archive*). [LDF 2013.08.08.]

This function is called by *purge_irods_archive*. It only requires the *id*, *user_id* and *path* data members to have been set. [LDF 2013.08.08.]

Log

[LDF 2013.08.08.] Added this function.

```

⟨Irods_Object_Type function declarations 1039⟩ +≡
int delete_from_archive(MYSQL *&mysql_ptr, string thread_str = "");

```

1174.

```

⟨ Irods_Object_Type::delete_from_archive definition 1174 ⟩ ≡
int Irods_Object_Type::delete_from_archive(MYSQL * &mysql_ptr, string thread_str){ bool
    DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status = 0;
    stringstream sql_strm;
    stringstream cmnd_strm;
    MYSQL_RES * result = 0;
    MYSQL_ROW curr_row = 0;
    unsigned int row_ctr = 0U;
    unsigned int field_ctr = 0U;
    long int affected_rows = 0UL;
    char buffer[1024];
    memset(buffer, 0, 1024);
    int temp_val = 0;
    unsigned long int temp_val_1 = 0;
    FILE *fp = 0;
    vector<unsigned long int> temp_handle_id_vector;
    vector<Handle_Type> temp_handle_vector;
    vector<unsigned long int> irods_object_id_vector;
    vector<Irods_Object_Type> irods_object_vector;
    Irods_Object_Type curr_irods_object;
    vector<Irods_AVU_Type>::iterator avu_iter;
    Irods_AVU_Type curr_avu;
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering 'Irods_Object_Type::delete_from_archive'." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

See also sections 1175, 1176, 1177, 1178, 1179, 1180, 1181, 1182, 1183, 1184, 1185, 1186, 1187, 1188, 1189, 1190, 1191, 1192, 1193, 1194, 1195, 1196, 1197, 1198, 1199, 1200, 1201, 1202, 1203, and 1204.

This code is used in section 1258.

1175. Retrieve user info. It's required in order to have access to the iRODS environment file for the user.
[LDF 2013.08.16.]

```

⟨ Irods_Object_Type::delete_from_archive definition 1174 ⟩ +≡
if (id == 0UL || user_id == 0 || path.length() == 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Irods_Object_Type::delete_from_archive':"
    << endl << "'id', 'user_id' or 'path' has invalid value:" << endl << "'id' == "
    id << endl << "'user_id' == " << user_id << endl << "'path' == "
    path << endl << "Can't delete iRODS object from archive." << endl <<
    "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    return 1;
} /* if (id == 0UL || user_id == 0 || path.length() == 0) */

```

1176.

```

⟨ Irods_Object_Type::delete_from_archive definition 1174 ⟩ +≡
    Scan_Parse_Parameter_Type param;
    User_Info_Type user_info;
    pthread_mutex_lock(&global_user_info_map_mutex);
    map<int, User_Info_Type>::iterator iter = global_user_info_map.find(user_id);
    if (iter == global_user_info_map.end()) {
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_str << "In 'Irods_Object_Type::delete_from_archive':"
                << endl << "User_info_for_user" << user_id << " not found on "
                << "'global_user_info_map'." << endl << "Will call 'Scan_Parse_Parameter_Type::get_user'." << endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        status = param.get_user(user_id, 0, "", &user_info, true);
        if (status != 0) {
            lock_cerr_mutex();
            cerr << thread_str << "ERROR! In 'Irods_Object_Type::delete_from_archive':"
                << endl << "'Scan_Parse_Parameter_Type::get_user' failed, returning"
                << status << "."
                << endl << "Can't delete iRODS_object from archive."
                << endl << "Exiting function unsuccessfully with return value 1."
                << endl;
            unlock_cerr_mutex();
            pthread_mutex_unlock(&global_user_info_map_mutex);
            return 1;
        } /* if (status != 0) */
#ifndef DEBUG_COMPILE
        else
            if (DEBUG) {
                lock_cerr_mutex();
                cerr << thread_str << "In 'Irods_Object_Type::delete_from_archive':"
                    << endl << "'Scan_Parse_Parameter_Type::get_user' succeeded, returning 0."
                    << endl;
#endif /* DEBUG_COMPILE */
#ifndef 0
        param.show("param:");
        user_info.show("user_info:");
#endif
#ifndef endif
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    global_user_info_map[user_id] = user_info;
} /* if (iter == global_user_info_map.end()) */
else {
    user_info = iter->second;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Irods_Object_Type::delete_from_archive':"
            << endl << "User_info_for_user" << user_id << " found on "
            << "'global_user_info_map'." << endl;
#endif /* DEBUG_COMPILE */
#ifndef 1
    user_info.show("user_info:");
#endif
}

```

```
#endif
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (iter == global_user_info_map.end()) */
pthread_mutex_unlock(&global_user_info_map_mutex);
```

1177.

Log

[LDF 2013.08.14.] BUG FIX: Now calling the shellscript `gw_irm.sh` instead of passing the “raw” commands to `popen`. Formerly, I was getting the following error: `sh: line 0: echo: write error: Broken pipe` I don’t know why this was happening, but using the shellscript seems to have fixed the problem. The problem may have been the `echo` commands and using `bash` rather than `sh` may have fixed the problem (whatever it was). (See below. [LDF 2013.08.16.])

[LDF 2013.08.16.] BUG FIX: Now calling `fread` and `sscanf` instead of `fscanf` followed conditionally by `fread`: This fixes the “broken pipe” error (see above). It probably isn’t necessary to use the shellscript `gw_irm.sh`, but I’m leaving this as it is for the moment, since I don’t see any particular disadvantage to using it. Using it is probably not significantly slower than passing the commands directly to `sh`.

```
<Irods_Object_Type::delete_from_archive definition 1174> +≡
cmnd_strm << config_dir << "/gw_irm.sh\" << user_info.get_irods_env_filename() << "\\\" << "\\\" <
path << "\\";
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "cmnd_strm.str()\\==\\\" << cmnd_strm.str() << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1178.

```

⟨ Irods_Object_Type::delete_from_archive definition 1174 ⟩ +≡
    errno = 0;
    fp = popen(cmnd_strm.str().c_str(), "r");
    if (fp == 0) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'Irods_Object_Type::delete_from_archive':"
            << endl <<
            "'popen' failed, returning NULL." << endl;
        if (errno != 0) cerr << "'errno' == " << errno << endl << strerror(errno) << endl;
        cerr << "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        return 1;
    } /* if (fp == 0) */
#endif /* DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Irods_Object_Type::delete_from_archive':"
            << endl <<
            "'popen' succeeded." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
memset(buffer, 0, 1024);
status = fread(buffer, 1, 1024, fp);
if (status == 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Irods_Object_Type::delete_from_archive':"
        << endl <<
        "'fread' failed, returning 0." << endl << "Failed to read output of 'gw_irm.sh'."
        << endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    pclose(fp);
    return 1;
} /* if (status == 0) */
else if (status == 1024) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Irods_Object_Type::delete_from_archive':"
        << endl <<
        "'fread' read 1024 characters."
        << endl <<
        "Output of 'gw_irm.sh' exceeds maximum amount (1023 characters)."
        << endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    pclose(fp);
    return 1;
} /* if (status == 0) */
#endif /* DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Irods_Object_Type::delete_from_archive':"
            << endl <<
            "'fread' succeeded, returning " << status << "."
            << endl << "'buffer' == " << endl <<
            buffer << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */

```

```
#endif /* DEBUG_COMPILE */
```

1179.

```
<Irods_Object_Type::delete_from_archive definition 1174> +≡
    pclose(fp);
    fp = 0;
    status = sscanf(buffer, "%d", &temp_val);
    if (status != 1) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'Irods_Object_Type::delete_from_archive':"
            << endl << "'sscanf' failed, returning" << status << "."
            << endl << "Failed to read exit status of 'irm' command from the output of 'gw_irm.sh'."
            << endl << "Exiting function unsuccessfully with return value 1."
            << endl;
        unlock_cerr_mutex();
        return 1;
    } /* if (status != 1) */
```

1180.

```
<Irods_Object_Type::delete_from_archive definition 1174> +≡
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Irods_Object_Type::delete_from_archive':"
            << endl << "'sscanf' succeeded, returning 1."
            << endl << "'temp_val' == "
            << temp_val << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1181.

```
<Irods_Object_Type::delete_from_archive definition 1174> +≡
if (temp_val != 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Irods_Object_Type::delete_from_archive':"
        << endl << "'gw_irm.sh' failed, returning" << temp_val << " (!= 0)"
        << endl << "'gw_irm.sh' output:" << endl << buffer << endl;
    return 1;
} /* else if (temp_val != 0) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Irods_Object_Type::delete_from_archive':"
            << endl << "'gw_irm.sh' succeeded, returning 0."
            << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1182.

```

⟨ Irods_Object_Type::delete_from_archive definition 1174 ⟩ +≡
    sql_strm.str("");
    sql_strm << "update_gwirdsif.Irods_Objects.set_deleted_from_archive=1," <<
        "marked_for_deletion_from_archive=0," << "last_modified=now()," <<
        "delete_from_archive_timestamp=now()" << "where_irods_object_id=" << id;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In `Irods_Object_Type::delete_from_archive':" << endl <<
            "'sql_strm.str()' == " << sql_strm.str() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    status = submit_mysql_query(sql_strm.str(), result, mysql_ptr, 0, 0, &affected_rows);
    if (status != 0) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In `Irods_Object_Type::delete_from_archive':" <<
            endl << "'submit_mysql_query' failed, returning " << status << "." << endl <<
            "Failed to update `gwirdsif.Irods_Objects` database table." << endl <<
            "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        if (result) mysql_free_result(result);
        return 1;
    } /* if (status != 0) */
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_str << "In `Irods_Object_Type::delete_from_archive':" <<
                endl << "'submit_mysql_query' succeeded, returning 0." <<
                endl << "'affected_rows' == " << affected_rows << endl <<
                "Updated `gwirdsif.Irods_Objects` database table successfully." << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    mysql_free_result(result);
    result = 0;

```

1183. Update associated handles. [LDF 2013.08.16.]

```

⟨ Irods_Object_Type::delete_from_archive definition 1174 ⟩ +≡
sql_strm.str("");
string handle_database = (standalone_handle) ? "handlesystem_standalone" : "handlesystem";
sql_strm << "select distinct handle_id from " << handle_database << ".handles"
    " where (type='IRODS_OBJECT' or type='DC_METADATA_IRODS_OBJECT' "
    " or type='IRODS_OBJECT_REF' or type='DC_METADATA_IRODS_OBJECT_REF') "
    " and data=" << path << " and created_by_user_id=" << user_id <<
    " order by handle_id";
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "'Irods_Object_Type::delete_from_archive':"
        << endl <<
        "sql_strm.str()=" << sql_strm.str() << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
status = submit_mysql_query(sql_strm.str(), result, mysql_ptr, &row_ctr, &field_ctr);
if (status != 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Irods_Object_Type::delete_from_archive':"
        << endl << "'submit_mysql_query' failed, returning"
        << status <<
        "." << endl << "Failed to retrieve handle ID(s) from "
        " " << handle_database << ".handles.database.table."
        << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    return 1;
} /* if (status != 0) */
#endif DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Irods_Object_Type::delete_from_archive':"
        << endl <<
        "'submit_mysql_query' succeeded, returning 0." << endl << "'row_ctr'=="
        row_ctr << endl << "Retrieved handle ID(s) from "
        " " << handle_database <<
        ".handles.database.table.successfully." << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1184.

```
< Irods_Object_Type::delete_from_archive definition 1174 > +≡
    for (int i = 0; i < row_ctr; ++i) {
        if ((curr_row = mysql_fetch_row(result)) == 0) {
            lock_cerr_mutex();
            cerr << thread_str << "ERROR! In 'Irods_Object_Type::delete_from_archive':" << endl <<
                "'mysql_fetch_row' failed, returning " << status << ":" << endl << mysql_error(mysql_ptr) <<
                endl << "Exiting function unsuccessfully with return value 1." << endl;
            unlock_cerr_mutex();
            mysql_free_result(result);
            return 1;
        } /* if */
    #if DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_str << "In 'Irods_Object_Type::delete_from_archive':" << endl <<
                "'mysql_fetch_row' succeeded." << endl << "'curr_row[0]' == " << curr_row[0] << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
    #endif /* DEBUG_COMPILE */
```

1185.

```
< Irods_Object_Type::delete_from_archive definition 1174 > +≡
    errno = 0;
    temp_val_1 = strtoul(curr_row[0], 0, 10);
    if (temp_val_1 == ULONG_MAX) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'Irods_Object_Type::delete_from_archive':" << endl <<
            "'strtoul' failed, returning 'ULONG_MAX'" << endl << strerror(errno) << endl <<
            "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        mysql_free_result(result);
        return 1;
    } /* if (temp_val_1 == ULONG_MAX) */
```

1186.

```
< Irods_Object_Type::delete_from_archive definition 1174 > +≡
    #if DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_str << "In 'Irods_Object_Type::delete_from_archive':" << endl <<
                "'strtoul' succeeded." << endl << "'temp_val_1' == " << temp_val_1 << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
    #endif /* DEBUG_COMPILE */
    temp_handle_id_vector.push_back(temp_val_1); } /* for */
```

1187.

```
< Irods_Object_Type::delete_from_archive definition 1174 > +≡
  if (result) {
    mysql_free_result(result);
    result = 0;
  }
  sqlLstrm.str("");
```

1188.

```
< Irods_Object_Type::delete_from_archive definition 1174 > +≡
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In `Irods_Object_Type::delete_from_archive':"
      << `temp_handle_id_vector.size()' == " << temp_handle_id_vector.size() << endl;
    if (temp_handle_id_vector.size() > 0) {
      cerr << "temp_handle_id_vector:" << endl;
      for (vector<unsigned long int>::const_iterator iter = temp_handle_id_vector.begin();
           iter != temp_handle_id_vector.end(); ++iter) {
        cerr << *iter << endl;
      }
      cerr << endl;
    } /* if (temp_handle_id_vector.size() > 0) */
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1189.

```

⟨Irods_Object_Type::delete_from_archive definition 1174⟩ +≡
    if (temp_handle_id_vector.size() > 0) { status = param.fetch_handles_from_database(temp_handle_id_vector,
        temp_handle_vector);
    if (status ≠ 0) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'Irods_Object_Type::delete_from_archive':"
            << endl <<
            "'Scan_Parse_Parameter_Type::fetch_handles_from_database' failed, "
            "returning " << status << "." << endl <<
            "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        return 1;
    } /* if (status ≠ 0) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Irods_Object_Type::delete_from_archive':"
            << endl <<
            "'Scan_Parse_Parameter_Type::fetch_handles_from_database' succeeded, "
            "returning 0." << endl << "'temp_handle_vector.size()' == "
            temp_handle_vector.size() << endl;
        if (temp_handle_vector.size() > 0) {
            cerr << "temp_handle_vector:" << endl;
            for (vector<Handle_Type>::const_iterator iter = temp_handle_vector.begin();
                iter ≠ temp_handle_vector.end(); ++iter) {
                iter->show();
            }
            cerr << endl;
        } /* if (temp_handle_vector.size() > 0) */
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1190.

```

⟨Irods_Object_Type::delete_from_archive definition 1174⟩ +≡
    string comma_str = "";
    sql_strm << "delete from " << handle_database << ".handles where handle_id in (";
    int handle_value_ctr = 0; for (vector<Handle_Type>::iterator iter = temp_handle_vector.begin();
        iter ≠ temp_handle_vector.end(); ++iter) {

```

1191.

```

⟨ Irods_Object_Type::delete_from_archive definition 1174 ⟩ +≡
    handle_value_ctr = 0;
    sql_strm << comma_str << iter->handle_id;
    comma_str = ",";
    status = add_handle_value(*iter, mysql_ptr, "IRODS_OBJECT", path,
        Handle_Value_Type::IRODS_OBJECT_DELETED_FROM_ARCHIVE_INDEX,
        "IRODS_OBJECT_DELETED_FROM_ARCHIVE", "", thread_str);
    if (status ≥ 0) {
        handle_value_ctr += status;
        status = add_handle_value(*iter, mysql_ptr, "DC_METADATA_IRODS_OBJECT", path,
            Handle_Value_Type::DC_METADATA_IRODS_OBJECT_DELETED_FROM_ARCHIVE_INDEX,
            "DC_METADATA_IRODS_OBJECT_DELETED_FROM_ARCHIVE", "", thread_str);
    }
    if (status ≥ 0) {
        handle_value_ctr += status;
        status = add_handle_value(*iter, mysql_ptr, "IRODS_OBJECT_REF", path,
            Handle_Value_Type::IRODS_OBJECT_REF_DELETED_FROM_ARCHIVE_INDEX,
            "IRODS_OBJECT_REF_DELETED_FROM_ARCHIVE", "", thread_str);
    }
    if (status ≥ 0) {
        handle_value_ctr += status;
        status = add_handle_value(*iter, mysql_ptr, "DC_METADATA_IRODS_OBJECT_REF", path,
            Handle_Value_Type::DC_METADATA_IRODS_OBJECT_REF_DELETED_FROM_ARCHIVE_INDEX,
            "DC_METADATA_IRODS_OBJECT_REF_DELETED_FROM_ARCHIVE", "", thread_str);
    }
    if (status ≥ 0) {
        handle_value_ctr += status;
    }
    if (status < 0) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'Irods_Object_Type::delete_from_archive':"
            endl << "'Irods_Object_Type::add_handle_value' failed, returning"
            endl << status << "."
            endl << "Failed to add handle value to handle."
            endl << "Exiting function unsuccessfully with return value 1."
            endl;
        unlock_cerr_mutex();
        return 1;
    } /* if (status < 0) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Irods_Object_Type::delete_from_archive':"
            endl << "'Irods_Object_Type::add_handle_value' succeeded (multiple times)."
            endl << "Added"
            endl << handle_value_ctr << " handle value(s) to handle."
            endl << "(0 is a permissible value.)"
            endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1192.

Log

[LDF 2013.08.16.] Added this section.

```
< Irods_Object_Type :: delete_from_archive definition 1174 > +≡
} /* for */
sql_strm << ") and (type='IRODS_OBJECT_MARKED_FOR_DELETION_FROM_ARCHIVE' "
" or type=" << "'DC_METADATA_IRODS_OBJECT_MARKED_FOR_DELETION_FROM_ARCHIVE')";
```

1193.

```
< Irods_Object_Type :: delete_from_archive definition 1174 > +≡
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Irods_Object_Type::delete_from_archive':" << endl <<
        "'sql_strm.str()' == " << sql_strm.str() << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1194.

```
< Irods_Object_Type :: delete_from_archive definition 1174 > +≡
status = submit_mysql_query(sql_strm.str(), result, mysql_ptr, 0, 0, &affected_rows);
if (status != 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR!" << "In 'Irods_Object_Type::delete_from_archive':" <<
        endl << "'submit_mysql_query' failed," << "returning" << status << "."
        << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    return 1;
} /* if (status != 0) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Irods_Object_Type::delete_from_archive':" << endl <<
            "'submit_mysql_query' succeeded," << "returning" << status << "."
            << "'affected_rows' == " << affected_rows << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
mysql_free_result(result);
result = 0;
sql_strm.str(""); /* if (temp_handle_id_vector.size() > 0) */
```

1195.

```
< Irods_Object_Type::delete_from_archive definition 1174 > +≡
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_str << "In 'Irods_Object_Type::delete_from_archive':" << endl <<
                "'temp_handle_id_vector' is empty. No handles to update." << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1196. Update "reference" ("REF") AVUs for other iRODS objects. [LDF 2013.08.16.]

```
< Irods_Object_Type::delete_from_archive definition 1174 > +≡
    sql_strm << "select distinct irods_object_id from gwidrsif.Irods_Info" <<
        "where user_id = " << user_id << " and (attribute = 'IRODS_OBJECT_REF' or
        "or attribute = 'DC_METADATA_IRODS_OBJECT_REF') and value = " << path << ",";
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Irods_Object_Type::delete_from_archive':" << endl <<
            "'sql_strm.str()' == " << sql_strm.str() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    status = submit_mysql_query(sql_strm.str(), result, mysql_ptr, &row_ctr, &field_ctr, 0);
    if (status != 0) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! " << "In 'Irods_Object_Type::delete_from_archive':" <<
            endl << "'submit_mysql_query' failed, " << "returning " << status << "."
        "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        if (result) mysql_free_result(result);
        return 1;
    } /* if (status != 0) */
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_str << "In 'Irods_Object_Type::delete_from_archive':" << endl <<
                "'submit_mysql_query' succeeded, " << "returning " << status << "."
            "row_ctr == " << row_ctr << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1197.

```
< Irods_Object_Type::delete_from_archive definition 1174 > +≡
    if (row_ctr == 0) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Irods_Object_Type::delete_from_archive':" << endl <<
            "'row_ctr'==0. Skipping to 'END_DELETE_FROM_ARCHIVE'." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    mysql_free_result(result);
    result = 0;
    goto END_DELETE_FROM_ARCHIVE;
} /* if (row_ctr == 0) */
```

1198.

```
< Irods_Object_Type::delete_from_archive definition 1174 > +≡
    for (int i = 0; i < row_ctr; ++i) {
        if ((curr_row = mysql_fetch_row(result)) == 0) {
            lock_cerr_mutex();
            cerr << "ERROR! In 'Irods_Object_Type::delete_from_archive':" << endl <<
                "'mysql_fetch_row' failed:" << endl << mysql_error(mysql_ptr) << endl <<
                "Exiting function unsuccessfully with return value 1." << endl;
            unlock_cerr_mutex();
        if (result) {
            mysql_free_result(result);
            result = 0;
        }
        return 1;
    } /* if (curr_row == mysql_fetch_row(result) == 0) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'Irods_Object_Type::delete_from_archive':" << endl <<
            "'mysql_fetch_row' succeeded." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1199.

```

⟨Irods_Object_Type::delete_from_archive definition 1174⟩ +≡
    errno = 0;
    temp_val_1 = strtoul(curr_row[0], 0, 10);
    if (temp_val_1 == ULONG_MAX) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'Irods_Object_Type::delete_from_archive':"
            << endl <<
            "'strtoul' failed, returning 'ULONG_MAX':"
            << endl << strerror(errno) << endl <<
            "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        mysql_free_result(result);
        return 1;
    } /* if (temp_val_1 == ULONG_MAX) */
    else {
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_str << "In 'Irods_Object_Type::delete_from_archive':"
                << endl <<
                "'strtoul' succeeded."
                << endl << "'temp_val_1' == "
                << temp_val_1 << endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        irods_object_id_vector.push_back(temp_val_1);
    } /* else */
    /* for */
    mysql_free_result(result);
    result = 0;
    sqlStrm.str("");

```

1200.

```

⟨Irods_Object_Type::delete_from_archive definition 1174⟩ +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Irods_Object_Type::delete_from_archive':"
            << endl <<
            "'irods_object_id_vector.size()' == "
            << irods_object_id_vector.size() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    if (irods_object_id_vector.size() == 0) {
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_str << "In 'Irods_Object_Type::delete_from_archive':"
                << endl <<
                "'irods_object_id_vector.size()' == 0."
                << endl <<
                "Skipping to 'END_DELETE_FROM_ARCHIVE'." << endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        goto END_DELETE_FROM_ARCHIVE;
    } /* if (irods_object_id_vector.size() == 0) */

```

1201.

```

⟨ Irods_Object_Type::delete_from_archive definition 1174 ⟩ +≡
    for (vector<unsigned long int>::const_iterator iter = irods_object_id_vector.begin();
        iter ≠ irods_object_id_vector.end(); ++iter) { curr_irods_object.clear();
    curr_irods_object.id = *iter;
    status = curr_irods_object.get_from_database(mysql_ptr, false); if (status ≤ 0) { #
if DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "WARNING! In 'Irods_Object_Type::delete_from_archive':"
            << endl <<
            "'Irods_Object::get_from_database' failed, returning" << status << "."
            << endl <<
            "Will try to continue." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    continue; } /* if (status ≤ 0) */
#
if DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Irods_Object_Type::delete_from_archive':"
            << endl <<
            "'Irods_Object::get_from_database' succeeded, returning" << status << "."
            << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    irods_object_vector.push_back(curr_irods_object); } /* for */

```

1202.

```

⟨ Irods_Object_Type::delete_from_archive definition 1174 ⟩ +≡
#ifndef DEBUG_COMPILE
if (irods_object_vector.size() == 0) {
#endif /* DEBUG_COMPILE */
    if (irods_object_vector.size() ≡ 0) {
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Irods_Object_Type::delete_from_archive':"
        << endl <<
        "'irods_object_vector.size()' == 0" << irods_object_vector.size() << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    goto END_DELETE_FROM_ARCHIVE;
} /* if (irods_object_vector.size() ≡ 0) */

```

1203.

```

⟨ Irods_Object_Type::delete_from_archive definition 1174 ⟩ +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Irods_Object_Type::delete_from_archive':" << endl <<
            "'irods_object_vector':" << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "path=" << path << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
for (vector<Irods_Object_Type>::iterator iter = irods_object_vector.begin();
     iter != irods_object_vector.end(); ++iter) {
    status = iter->add_avu_cond(user_info.get_irods_env_filename(), "IRODS_OBJECT_REF", path,
        "IRODS_OBJECT_REF_DELETED_FROM_ARCHIVE", mysql_ptr);
    if (status ≥ 0)
        status = iter->add_avu_cond(user_info.get_irods_env_filename(), "DC_METADATA_IRODS_OBJECT_REF",
            path, "DC_METADATA_IRODS_OBJECT_REF_DELETED_FROM_ARCHIVE", mysql_ptr);
    if (status < 0) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'Irods_Object_Type::delete_from_archive':" << endl <<
            "'Irods_Object::add_avu_cond' failed, returning " << status << "." << endl <<
            "Will try to continue." << endl;
        unlock_cerr_mutex();
    }
} /* for */

```

1204.

```

⟨ Irods_Object_Type::delete_from_archive definition 1174 ⟩ +≡
END_DELETE_FROM_ARCHIVE:
    if (result) mysql_free_result(result);
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Exiting 'Irods_Object_Type::delete_from_archive' successfully" <<
            "with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; } /* End of Irods_Object_Type::delete_from_archive definition */

```

1205. Delete from ‘gwirdsif’ database (*delete_from_gwirdsif_db*). [LDF 2013.08.12.]

```

⟨ Irods_Object_Type function declarations 1039 ⟩ +≡
int delete_from_gwirdsif_db(MYSQL *&mysql_ptr, string thread_str = "");

```

1206.

```

⟨ Irods_Object_Type::delete_from_gwirdsif_db definition 1206 ⟩ ≡
int Irods_Object_Type::delete_from_gwirdsif_db(MYSQL * &mysql_ptr, string thread_str){ bool
    DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    MYSQL_RES * result = 0;
    MYSQL_ROW curr_row;
    unsigned int row_ctr;
    unsigned int field_ctr;
    long int affected_rows;
    stringstream sql_strm;
    string comma_str = "";
#ifndef 0
    int ret_val = 0;
    stringstream temp_strm;
    string temp_str;
#endif
#ifndef
    int status = 0;
    unsigned long temp_val = 0UL;
    vector<unsigned long int> temp_handle_id_vector;
    vector<Handle_Type> temp_handle_vector;
    int handle_value_ctr = 0;
    map<unsigned long int, Handle_Value_Type>::iterator hv_iter;
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering 'Irods_Object_Type::delete_from_gwirdsif_db'." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
See also sections 1207, 1208, 1209, 1210, 1211, 1212, 1213, 1214, 1215, 1216, 1217, 1218, 1219, 1220, 1221, and 1222.
This code is used in section 1258.

```

1207.

```

⟨ Irods_Object_Type::delete_from_gwirdsif_db definition 1206 ⟩ +≡
if (id ≡ 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Irods_Object_Type::delete_from_gwirdsif_db':"
    << endl << "'id' ≡ 0. Can't delete row(s) from 'gwirdsif.Irods_Objects' database table."
    << endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    return 1;
} /* if (id ≡ 0) */

```

1208.

```

⟨ Irods_Object_Type::delete_from_gwirdsif_db definition 1206 ⟩ +≡
    string handle_database = (standalone_handle) ? "handlesystem_standalone" : "handlesystem";
    if (handle_id_vector.size() ≡ 0) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Irods_Object_Type::delete_from_gwirdsif_db':" <<
            endl << "'handle_id_vector.size()' == 0." << endl <<
            "Not calling 'Handle_Type::fetch_handles_from_database'." << endl <<
            "Skipping to 'POST_HANDLES'." << endl;
        unlock_cerr_mutex();
        goto POST_HANDLES;
    } /* if (handle_id_vector.size() ≡ 0) */
    status = Handle_Type::fetch_handles_from_database(mysql_ptr, handle_id_vector, handle_vector, "", thread_str);
    if (status ≡ 2) {
        lock_cerr_mutex();
        cerr << thread_str << "WARNING! In 'Irods_Object_Type::delete_from_gwirdsif_db':" <<
            endl << "'Handle_Type::fetch_handles_from_database' returned 2:" << endl <<
            "No rows retrieved from '" << handle_database << ".handles'" << "database_table." <<
            endl << "Skipping to 'POST_HANDLES'." << endl;
        unlock_cerr_mutex();
        goto POST_HANDLES;
    } /* if (status ≡ 2) */
    else if (status ≠ 0) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'Irods_Object_Type::delete_from_gwirdsif_db':" << endl <<
            "'Handle_Type::fetch_handles_from_database' failed, returning " << status << "." <<
            endl << "Failed to retrieve handles from '" << handle_database << ".handles'" <<
            "database_table." << endl << "Exiting function unsuccessfully with return value 1." <<
            endl;
        unlock_cerr_mutex();
        return 1;
    } /* else if (status ≠ 0) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Irods_Object_Type::delete_from_gwirdsif_db':" << endl <<
            "'Handle_Type::fetch_handles_from_database' succeeded, returning 0." <<
            "'handle_vector.size()' == " << handle_vector.size() << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1209.

```

⟨Irods_Object_Type::delete_from_gwirdsif_db definition 1206⟩ +≡
  sql_strm << "delete_from_" << handle_database << ".handles" << "where_handle_id_in_(";
  comma_str = "";
  for (vector<Handle_Type>::iterator iter = handle_vector.begin(); iter != handle_vector.end();
       ++iter) {
    sql_strm << comma_str << iter->handle_id;
    comma_str = ",";
  } /* for */
  sql_strm << ")") &and (type='IRODS_OBJECT_MARKED_FOR_DELETION_FROM_GWIRDSIF_DB' <<
    "or type='DC_METADATA_IRODS_OBJECT_MARKED_FOR_DELETION_FROM_GWIRDSIF_DB')";
#endif DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In `Irods_Object_Type::delete_from_gwirdsif_db': " << endl <<
      "'sql_strm.str()' == " << sql_strm.str() << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  status = submit_mysql_query(sql_strm.str(), result, mysql_ptr, 0, 0, &affected_rows);
  if (status != 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In `Irods_Object_Type::delete_from_gwirdsif_db': " <<
      endl << "'submit_mysql_query' failed, returning " << status << "." << endl <<
      "Failed to delete rows from the " << handle_database << ".handles" << database_table." <<
      endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    return 1;
  } /* if (status != 0) */
#endif DEBUG_COMPILE
else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In `Irods_Object_Type::delete_from_gwirdsif_db': " << endl <<
      "'submit_mysql_query' succeeded, returning 0." << endl << "'affected_rows' == " <<
      affected_rows << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  mysql_free_result(result);
  result = 0;
  sql_strm.str("");

```

1210.

```

⟨ Irods_Object_Type::delete_from_gwirdsif_db definition 1206 ⟩ +≡
for (vector<Handle_Type>::iterator iter = handle_vector.begin(); iter ≠ handle_vector.end();
     ++iter) {
    handle_value_ctr = 0;
    status = add_handle_value(*iter, mysql_ptr, "IRODS_OBJECT", path,
        Handle_Value_Type::IRODS_OBJECT_DELETED_FROM_GWIRDSIF_DB_INDEX,
        "IRODS_OBJECT_DELETED_FROM_GWIRDSIF_DB", "", thread_str);
    if (status ≥ 0) {
        handle_value_ctr += status;
        status = add_handle_value(*iter, mysql_ptr, "DC_METADATA_IRODS_OBJECT", path,
            Handle_Value_Type::DC_METADATA_IRODS_OBJECT_DELETED_FROM_GWIRDSIF_DB_INDEX,
            "DC_METADATA_IRODS_OBJECT_DELETED_FROM_GWIRDSIF_DB", "", thread_str);
    }
    if (status ≥ 0) {
        handle_value_ctr += status;
    }
    if (status < 0) {
        lock_cerr_mutex();
        cerr ≪ thread_str ≪ "ERROR! In 'Irods_Object_Type::delete_from_gwirdsif_db':"
        endl ≪ "'Irods_Object_Type::add_handle_value' failed, returning"
        status ≪ "."
        endl ≪ "Failed to add handle value to handle."
        endl ≪ "Exiting function unsuccessfully with return value 1."
        endl;
        unlock_cerr_mutex();
        return 1;
    } /* if (status < 0) */
#endif DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "In 'Irods_Object_Type::delete_from_gwirdsif_db':"
    endl ≪ "'Irods_Object_Type::add_handle_value' succeeded (multiple times)."
    endl ≪ "Added"
    endl ≪ handle_value_ctr ≪ " handle value(s) to handle."
    endl ≪ "(0 is a permissible value.)"
    endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* for */

```

1211.

```

⟨ Irods_Object_Type::delete_from_gwirdsif_db definition 1206 ⟩ +≡
POST_HANDLES:
  if (result) {
    mysql_free_result(result);
    result = 0;
  }
  sqlStrm.str("");
  sqlStrm << "delete from gwirdsif.Irods_Objects where irods_object_id=" << id;
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Irods_Object_Type::delete_from_gwirdsif_db':" << endl <<
      "'sql_strm.str()' == " << sqlStrm.str() << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  status = submit_mysql_query(sqlStrm.str(), result, mysql_ptr, 0, 0, &affected_rows);
  if (status != 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Irods_Object_Type::delete_from_gwirdsif_db':" <<
      endl << "'submit_mysql_query' failed, returning" << status << "." << endl <<
      "Failed to delete row from the 'gwirdsif.Irods_Objects' database table." << endl <<
      "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    return 1;
  } /* if (status != 0) */
#endif DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << thread_str << "In 'Irods_Object_Type::delete_from_gwirdsif_db':" << endl <<
        "'submit_mysql_query' succeeded, returning 0." << endl << "'affected_rows'" == " <<
        affected_rows << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1212.

```

⟨ Irods_Object_Type::delete_from_gwirdsif_db definition 1206 ⟩ +≡
  if (result) {
    mysql_free_result(result);
    result = 0;
  }
  sql_strm.str("");
  sql_strm << "delete_from_gwirdsif.Irods_AVUs WHERE irods_object_id=" << id;
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Irods_Object_Type::delete_from_gwirdsif_db':" << endl <<
      "'sql_strm.str()'==" << sql_strm.str() << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  status = submit_mysql_query(sql_strm.str(), result, mysql_ptr, 0, 0, &affected_rows);
  if (status != 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Irods_Object_Type::delete_from_gwirdsif_db':" <<
      endl << "'submit_mysql_query' failed, returning" << status << "." << endl <<
      "Failed to delete row(s) from the 'gwirdsif.Irods_AVUs' database table." << endl <<
      "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    return 1;
  } /* if (status != 0) */
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << thread_str << "In 'Irods_Object_Type::delete_from_gwirdsif_db':" << endl <<
        "'submit_mysql_query' succeeded, returning 0." << endl << "'affected_rows'==" <<
        affected_rows << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1213.

```

⟨ Irods_Object_Type::delete_from_gwirdsif_db definition 1206 ⟩ +≡
    mysql_free_result(result);
    result = 0;
    sql_strm.str("");
    sql_strm << "delete from gwirdsif.Irods_Objects_Handles where irods_object_id=" << id;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Irods_Object_Type::delete_from_gwirdsif_db':" << endl <<
            "'sql_strm.str()' == " << sql_strm.str() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    status = submit_mysql_query(sql_strm.str(), result, mysql_ptr, 0, 0, &affected_rows);
    if (status != 0) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'Irods_Object_Type::delete_from_gwirdsif_db':" <<
            endl << "'submit_mysql_query' failed, returning " << status << "." << endl <<
            "Failed to delete row(s) from the 'gwirdsif.Irods_Objects_Handles' database table";
        e." << endl << "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        if (result) mysql_free_result(result);
        return 1;
    } /* if (status != 0) */
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_str << "In 'Irods_Object_Type::delete_from_gwirdsif_db':" << endl <<
                "'submit_mysql_query' succeeded, returning 0." << endl << "'affected_rows'" == " <<
                affected_rows << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1214.

Log

[LDF 2013.08.15.] Added this section.

```

⟨ Irods_Object_Type :: delete_from_gwirdsif_db definition 1206 ⟩ +≡
  if (result) {
    mysql_free_result(result);
    result = 0;
  }
  sql_strm.str("");
  sql_strm << "select distinct handle_id from " << handle_database << ".handles" <<
  " where (type='IRODS_OBJECT_REF' or type='DC_METADATA_IRODS_OBJECT_REF') and "
  " and data=" << path << ", order by handle_id";
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In `Irods_Object_Type::delete_from_gwirdsif_db':"
    " `sql_strm.str()' == " << sql_strm.str() << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  status = submit_mysql_query(sql_strm.str(), result, mysql_ptr, &row_ctr, &field_ctr);
  if (status != 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In `Irods_Object_Type::delete_from_gwirdsif_db':"
    " `submit_mysql_query' failed, returning " << status << "."
    " Failed to retrieve row(s) from the " << handle_database << ".handles" <<
    " database.table." << endl << "Exiting function unsuccessfully with return value 1."
    " endl;
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    return 1;
  } /* if (status != 0) */
#endif DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << thread_str << "In `Irods_Object_Type::delete_from_gwirdsif_db':"
      " `submit_mysql_query' succeeded, returning 0."
      " << endl << `row_ctr' == "
      " row_ctr << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1215.

```

⟨ Irods_Object_Type::delete_from_gwirdsif_db definition 1206 ⟩ +≡
    if (row_ctr == 0) {
#if DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Irods_Object_Type::delete_from_gwirdsif_db':" << endl <<
            "'row_ctr'==0 Skipping to 'END_DELETE_FROM_GWIRDSIF_DB'." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    mysql_free_result(result);
    result = 0;
    goto END_DELETE_FROM_GWIRDSIF_DB;
} /* if (row_ctr == 0) */

```

1216.

```

⟨ Irods_Object_Type::delete_from_gwirdsif_db definition 1206 ⟩ +≡
    for (int i = 0; i < row_ctr; ++i) {
        if ((curr_row = mysql_fetch_row(result)) == 0) {
            lock_cerr_mutex();
            cerr << thread_str << "ERROR! In 'Irods_Object_Type::delete_from_gwirdsif_db':" <<
                endl << "'mysql_fetch_row' failed:" << endl << mysql_error(mysql_ptr) << endl <<
                "Exiting function unsuccessfully with return value 1." << endl;
            unlock_cerr_mutex();
        if (result) {
            mysql_free_result(result);
            result = 0;
        }
        return 1;
    } /* if (curr_row == mysql_fetch_row(result) == 0) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Irods_Object_Type::delete_from_gwirdsif_db':" << endl <<
            "'mysql_fetch_row' succeeded." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1217.

```

⟨ Irods_Object_Type::delete_from_gwirdsif_db definition 1206 ⟩ +≡
    temp_val = strtoul(curr_row[0], 0, 10);
    errno = 0;
    if (temp_val == ULONG_MAX) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'Irods_Object_Type::delete_from_gwirdsif_db':"
            << endl <<
            "'strtoul' failed, returning 'UNLONG_MAX':"
            << endl << strerror(errno) << endl <<
            "Exiting function unsuccessfully with return value 1."
            << endl;
        unlock_cerr_mutex();
        mysql_free_result(result);
        return 1;
    } /* if (temp_val == ULONG_MAX) */
    else {
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_str << "In 'Irods_Object_Type::delete_from_gwirdsif_db':"
                << endl <<
                "'strtoul' succeeded."
                << endl << "'temp_val' == "
                << temp_val << endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        temp_handle_id_vector.push_back(temp_val);
    } /* else */
} /* for */
mysql_free_result(result);
result = 0;

```

1218.

```

⟨ Irods_Object_Type::delete_from_gwirdsif_db definition 1206 ⟩ +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Irods_Object_Type::delete_from_gwirdsif_db':"
            << endl <<
            "'temp_handle_id_vector.size()' == "
            << temp_handle_id_vector.size() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    if (temp_handle_id_vector.size() == 0) {
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_str << "In 'Irods_Object_Type::delete_from_gwirdsif_db':"
                << endl <<
                "'temp_handle_id_vector.size()' == 0. No handles to fetch."
                << endl <<
                "Skipping to 'END_DELETE_FROM_GWIRDSIF_DB'."
                << endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        goto END_DELETE_FROM_GWIRDSIF_DB;
    } /* if (temp_handle_id_vector.size() == 0) */

```

1219.

```

⟨ Irods_Object_Type::delete_from_gwirdsif_db definition 1206 ⟩ +≡
    status = Handle_Type::fetch_handles_from_database(mysql_ptr, temp_handle_id_vector,
        temp_handle_vector, "", thread_str);
    if (status ≡ 2) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Irods_Object_Type::delete_from_gwirdsif_db':"
            << endl <<
            "'Handle_Type::fetch_handles_from_database' returned 2, i.e.,"
            << endl <<
            "No rows retrieved from "
            << handle_database << ".handles."
            << endl <<
            "Skipping to 'END_DELETE_FROM_GWIRDSIF_DB'."
            << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    goto END_DELETE_FROM_GWIRDSIF_DB;
} /* if (status ≡ 2) */
else if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Irods_Object_Type::delete_from_gwirdsif_db':"
        << endl <<
        "'Handle_Type::fetch_handles_from_database' failed, returning"
        << status << "."
        << endl << "Failed to retrieve handles."
        << endl <<
        "Exiting function unsuccessfully with return value 1."
        << endl;
    unlock_cerr_mutex();
    return 1;
} /* else if (status ≠ 0) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Irods_Object_Type::delete_from_gwirdsif_db':"
            << endl <<
            "'Handle_Type::fetch_handles_from_database' succeeded, returning 0."
            << endl <<
            "'temp_handle_vector.size()' == "
            << temp_handle_vector.size() << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    if (temp_handle_vector.size() ≡ 0) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Irods_Object_Type::delete_from_gwirdsif_db':"
            << endl <<
            "'temp_handle_vector.size()' == 0. No handles."
            << endl <<
            "Skipping to 'END_DELETE_FROM_GWIRDSIF_DB'."
            << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    goto END_DELETE_FROM_GWIRDSIF_DB;
} /* if (temp_handle_vector.size() ≡ 0) */

```

1220.

```
< Irods_Object_Type :: delete_from_gwirdsif_db definition 1206 > +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "temp_handle_vector:" << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    for (vector<Handle_Type>::iterator iter = temp_handle_vector.begin();
         iter != temp_handle_vector.end(); ++iter) { handle_value_ctr = 0;
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        iter->show();
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1221.

```

⟨ Irods_Object_Type::delete_from_gwirdsif_db definition 1206 ⟩ +≡
    status = add_handle_value(*iter, mysql_ptr, "IRODS_OBJECT_REF", path,
        Handle_Value_Type::IRODS_OBJECT_REF_DELETED_FROM_GWIRDSIF_DB_INDEX,
        "IRODS_OBJECT_REF_DELETED_FROM_GWIRDSIF_DB", "", thread_str);
    if (status ≥ 0) {
        handle_value_ctr += status;
        status = add_handle_value(*iter, mysql_ptr, "DC_METADATA_IRODS_OBJECT_REF", path,
            Handle_Value_Type::DC_METADATA_IRODS_OBJECT_REF_DELETED_FROM_GWIRDSIF_DB_INDEX,
            "DC_METADATA_IRODS_OBJECT_REF_DELETED_FROM_GWIRDSIF_DB", "", thread_str);
    }
    if (status ≥ 0) {
        handle_value_ctr += status;
    }
    if (status < 0) {
        lock_cerr_mutex();
        cerr ≪ thread_str ≪ "ERROR! In 'Irods_Object_Type::delete_from_gwirdsif_db':"
            endl ≪ "'Irods_Object_Type::add_handle_value' failed, returning"
            status ≪ "."
            endl ≪ "Failed to add handle value to handle."
            endl ≪ "Exiting function unsuccessfully with return value 1."
            endl;
        unlock_cerr_mutex();
        return 1;
    } /* if (status < 0) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ thread_str ≪ "In 'Irods_Object_Type::delete_from_gwirdsif_db':"
            endl ≪ "'Irods_Object_Type::add_handle_value' succeeded (multiple times)."
            endl ≪ "Added"
            handle_value_ctr ≪ " handle value(s) to handle."
            endl ≪ "(0 is a permissible value.)"
            endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* for */
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1222.

```
< Irods_Object_Type::delete_from_gwirdsif_db definition 1206 > +≡
END_DELETE_FROM_GWIRDSIF_DB:
    if (result) {
        mysql_free_result(result);
        result = 0;
    }
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Exiting 'Irods_Object_Type::delete_from_gwirdsif_db' "
            << "successfully with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; } /* End of Irods_Object_Type::delete_from_gwirdsif_db definition */
```

1223. Add handle value. [LDF 2013.08.15.]

Return values:

- 0: No handle value with type *old_type_str*.
- 1: Handle value with type *old_type_str* exists, added one with type *new_type_str*.
- 1: **Handle_Type** :: *add_value* failed.

Log

[LDF 2013.08.15.] Added this function. It's called in and **Irods_Object_Type** :: *delete_from_archive* and **Irods_Object_Type** :: *delete_from_gwirdsif_db*.

< Irods_Object_Type function declarations 1039 > +≡

```
int add_handle_value(Handle_Type &handle, MYSQL * &mysql_ptr, string old_type_str, string
path, unsigned int index, string new_type_str, string data_str, string thread_str = "");
```

1224.

```
< Irods_Object_Type::add_handle_value definition 1224 > ≡
int Irods_Object_Type::add_handle_value(Handle_Type &handle, MYSQL * &mysql_ptr, string
old_type_str, string path, unsigned int index, string new_type_str, string data_str, string
thread_str){ bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
```

See also sections 1225, 1226, 1227, and 1228.

This code is used in section 1258.

1225.

```
< Irods_Object_Type::add_handle_value definition 1224 > +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering 'Irods_Object_Type::add_handle_value'." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    int status;
    map<unsigned long int,Handle_Value_Type>::iterator hv_iter;
    hv_iter = handle.find(old_type_str); if (hv_iter != handle.handle_value_map.end()) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Irods_Object_Type::add_handle_value':"
            << endl << "Handle_value_with_type==" << old_type_str << "' present." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1226.

```
< Irods_Object_Type::add_handle_value definition 1224 > +≡
    if (strncmp(hv_iter->second.data, path.c_str(), hv_iter->second.data_length) ≠ 0) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Irods_Object_Type::add_handle_value':"
            << endl << "'hv_iter->second.data'==";
        fwrite(hv_iter->second.data, 1, hv_iter->second.data_length, stderr);
        cerr << endl << "'path'==XXXXXXXXXXXXXXXXXXXX"
            << path << endl <<
            "'hv_iter->second.data'!= 'path'. Not adding handle." << endl <<
            "Exiting function successfully with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0;
} /* if */
```

1227.

```
< Irods_Object_Type::add_handle_value definition 1224 > +≡
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_str << "In 'Irods_Object_Type::add_handle_value':" << endl <<
                "'hv_iter->second.data'==";
            fwrite(hv_iter->second.data, 1, hv_iter->second.data_length, stderr);
            cerr << endl << "'path'=='" << path << endl <<
                "'hv_iter->second.data'=='path'.Continuing." << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1228.

```
< Irods_Object_Type::add_handle_value definition 1224 > +≡
    if (data_str.empty()) {
        data_str = path;
    }
    status = handle.add_value(mysql_ptr, index, new_type_str, data_str, hv_iter->second.created_by_user_id);
    if (status ≠ 0) {
        lock_cerr_mutex();
        cerr << "ERROR! In 'Irods_Object_Type::add_handle_value':" << endl <<
            "'Handle_Type::add_value' failed, " << "returning " << status << "." << endl <<
            "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        return -1;
    }
#endif DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "In 'Irods_Object_Type::add_handle_value':" << endl <<
                "'Handle_Type::add_value' succeeded, returning 0." << endl <<
                "Exiting function successfully with return value 1." << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 1; } /* if (hv_iter ≠ handle.handle_value_map.end()) */
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "Handle_value with type == 'IRODS_OBJECT' not present." << endl << endl <<
                "Exiting function successfully with return value 0." << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; } /* End of Irods_Object_Type::add_handle_value definition */
```

1229. Find AVU. (*find_avu*). [LDF 2013.08.16.]

Log

[LDF 2013.08.16.] Added this function.

```
<Irods_Object_Type function declarations 1039> +≡  
const vector<Irods_AVU_Type>::iterator find_avu(string attrib, string val, string thread_str = "");
```

1230.

```

⟨ Irods_Object_Type::find_avu definition 1230 ⟩ ≡
    const vector<Irods_AVU_Type>::iterator Irods_Object_Type::find_avu(string attrib, string
                           val, string thread_str)
    {
        bool DEBUG = false;      /* true */
        set_debug_level(DEBUG, 0, 0);
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_str << "Entering 'Irods_Object_Type::find_avu'." << endl;
            unlock_cerr_mutex();
        }      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */
        if (avu_vector.size() == 0) {
#ifndef DEBUG_COMPILE
            if (DEBUG) {
                lock_cerr_mutex();
                cerr << thread_str << "In 'Irods_Object_Type::find_avu':" <<
                    endl << "'avu_vector.size()'==0" << endl <<
                    "Exiting function successfully with return value" << "'avu_vector.end()'" <<
                    endl;
                unlock_cerr_mutex();
            }      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */
            return avu_vector.end();
        }      /* if (avu_vector.size() == 0) */
        vector<Irods_AVU_Type>::iterator iter;
        for (iter = avu_vector.begin(); iter != avu_vector.end(); ++iter) {
            if (iter->attribute == attrib & iter->value == val) {
#ifndef DEBUG_COMPILE
                if (DEBUG) {
                    lock_cerr_mutex();
                    cerr << thread_str << "In 'Irods_Object_Type::find_avu':" << endl <<
                        "AVU with 'attribute'==" << attrib << " and " << "'value'==" << val <<
                        "'found:'" << endl;
                    iter->show();
                    cerr << "Exiting function successfully with return value" << "'iter'." << endl;
                    unlock_cerr_mutex();
                }      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */
                return iter;
            }      /* if (found matching AVU) */
        }      /* for */
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_str << "In 'Irods_Object_Type::find_avu':" << endl <<
                "AVU with 'attribute'==" << attrib << " and " << "'value'==" << val <<
                "'not found.'" << endl << "Exiting function successfully with return value" <<
                "'avu_vector.end()'" << endl;

```

```

    unlock_cerr_mutex();
}
/* if (DEBUG) */
#endif /* DEBUG_COMPILE */
return avu_vector.end();
} /* End of Irods_Object_Type::find_avu definition */

```

This code is used in section 1258.

1231. Undelete iRODS objects (*undelete_irods_objects*). [LDF 2013.08.16.]

Log

[LDF 2013.08.16.] Added this function.

```

⟨Irods_Object_Type function declarations 1039⟩ +≡
static int undelete_irods_objects(vector<Irods_Object_Type> &irods_object_vector,
    MYSQL*&mysql_ptr, bool archive, bool database, vector<Response_Type> &response_vector, string
    irods_env_filename, string thread_str);

```

1232.

```

⟨Irods_Object_Type::undelete_irods_objects definition 1232⟩ ≡
int Irods_Object_Type::undelete_irods_objects(vector<Irods_Object_Type> &irods_object_vector,
    MYSQL * &mysql_ptr, bool archive, bool database, vector<Response_Type>
    &response_vector, string irods_env_filename, string thread_str){ bool DEBUG = false;
/* true */
set_debug_level(DEBUG, 0, 0);
int status;
Response_Type response;
stringstream temp_strm;
unsigned long temp_val;
int temp_val_1;
MYSQL_RES * result_array[4] = {0, 0, 0, 0};
MYSQL_ROW curr_row;
vector<unsigned int *> row_ctr_vector;
vector<unsigned int *> field_ctr_vector;
vector<long int *> affected_rows_vector;
stringstream sql_strm[4];
vector<string> query_vector;
string comma_str[2]; std::set<unsigned long int> handle_id_set;
vector<string> handle_value_type_vector;
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "Entering 'Irods_Object_Type::undelete_irods_objects'." << endl <<
        "archive==" << archive << endl << "database==" << database << endl;
    unlock_cerr_mutex();
}
/* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

See also sections 1233, 1234, 1235, 1236, 1237, 1238, 1239, 1240, 1241, 1242, 1243, 1244, 1245, 1246, 1247, 1248, 1249, 1250, 1251, 1252, 1253, 1254, 1255, and 1256.

This code is used in section 1258.

1233.**Log**

[LDF 2013.08.19.] Added this section.

```
< Irods_Object_Type :: undelete_irods_objects definition 1232 > +≡
if (¬(archive ∨ database)) {
    lock_cerr_mutex();
    cerr << thread_str << "WARNING! In 'Irods_Object_Type::undelete_irods_objects':"
    endl << "'archive' and 'database' are both 'false'." <<
    endl << "Not deleting 'Irods_Object_Type' objects." << endl <<
    "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    return 1;
} /* if (¬(archive ∨ database)) */
```

1234.

```
< Irods_Object_Type :: undelete_irods_objects definition 1232 > +≡
if (irods_object_vector.size() ≡ 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Irods_Object_Type::undelete_irods_objects':"
    endl << "'irods_object_vector.size()' == 0" << endl <<
    "No 'Irods_Object_Type' objects to undelete." << endl <<
    "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    return 1;
} /* if (irods_object_vector.size() ≡ 0) */
```

1235.

```
< Irods_Object_Type :: undelete_irods_objects definition 1232 > +≡
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_str << "In 'Irods_Object_Type::undelete_irods_objects':"
            << endl <<
            "'irods_object_vector.size()' == " << irods_object_vector.size() << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1236.

```

⟨ Irods_Object_Type :: undelete_irods_objects definition 1232 ⟩ +≡
    for (vector<Irods_Object_Type>::iterator iter = irods_object_vector.begin();
         iter ≠ irods_object_vector.end(); ++iter) {
#ifndef 0
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        iter->show(" *iter:");
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
#endif
    if (archive ∧ iter->deleted_from_archive) {
        response.clear();
        response.type = Response_Type::COMMAND_ONLY_TYPE;
        temp_strm.str("");
        temp_strm << "UNDELETE_RESPONSE_1_" << iter->path << "\_" <<
            "\_iRODS\_object\_already\_deleted\_from\_archive\";
        response.command = temp_strm.str();
        temp_strm.str("");
        response_vector.push_back(response);
    }
    else if (archive ∧ iter->marked_for_deletion_from_archive) {
        sql_strm[0] << comma_str[0] << iter->id;
        comma_str[0] = ",_";
    }
    if (database ∧ iter->deleted_from_gwirdsif_db) {
        response.clear();
        response.type = Response_Type::COMMAND_ONLY_TYPE;
        temp_strm.str("");
        temp_strm << "UNDELETE_RESPONSE_1_" << iter->path << "\_" <<
            "\_iRODS\_object\_already\_deleted\_from\_‘gwirdsif’\_database\";
        response.command = temp_strm.str();
        temp_strm.str("");
        response_vector.push_back(response);
    }
    else if (database ∧ iter->marked_for_deletion_from_gwirdsif_db) {
        sql_strm[1] << comma_str[1] << iter->id;
        comma_str[1] = ",_";
    }
} /* for */

```

1237.

```
< Irods_Object_Type :: undelete_irods_objects definition 1232 > +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In `Irods_Object_Type::undelete_irods_objects'" << endl <<
            "'sql_strm[0].str()' == " << sql_strm[0].str() << endl << "'sql_strm[1].str()' == " <<
            sql_strm[1].str() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1238.

```
< Irods_Object_Type :: undelete_irods_objects definition 1232 > +≡
if (!sql_strm[0].str().empty()) {
    sql_strm[2] << sql_strm[0].str();
    temp_strm.str("");
    temp_strm << "update_gwirdsif.Irods_Objects.set_marked_for_deletion_from_\
archive=0," << "delete_from_archive_timestamp=0, last_modified=now()," <<
    "where_irods_object_id_in(" << sql_strm[0].str() << ")";
    sql_strm[0].str("");
    sql_strm[0] << temp_strm.str();
    temp_strm.str("");
    temp_strm << "select_handle_id_from_gwirdsif.Irods_Objects_Handles" <<
        "where_irods_object_id_in(" << sql_strm[2].str() << ")";
    sql_strm[2].str("");
    sql_strm[2] << temp_strm.str();
    temp_strm.str("");
}
if (!sql_strm[1].str().empty()) {
    sql_strm[3] << sql_strm[1].str();
    temp_strm.str("");
    temp_strm << "update_gwirdsif.Irods_Objects.set_marked_for_deletion_from_\
gwirdsif_db=0," << "delete_from_gwirdsif_db_timestamp=0, last_modified=now(\n\
)" << "where_irods_object_id_in(" << sql_strm[1].str() << ")";
    sql_strm[1].str("");
    sql_strm[1] << temp_strm.str();
    temp_strm.str("");
    temp_strm << "select_handle_id_from_gwirdsif.Irods_Objects_Handles" <<
        "where_irods_object_id_in(" << sql_strm[3].str() << ")";
    sql_strm[3].str("");
    sql_strm[3] << temp_strm.str();
    temp_strm.str("");
}
```

1239.

```
< Irods_Object_Type::undelete_irods_objects definition 1232 > +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In `Irods_Object_Type::undelete_irods_objects': " << endl;
        for (int i = 0; i < 4; ++i)
            cerr << 'sql_strm[' << i << "] .str()' <=> sql_strm[i].str() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1240.

Log

[LDF 2013.09.03.] BUG FIX: Added section { Delete pointers and clear vectors 1257 }. Now inserting it to **delete** the heap objects created by the calls to **new** below. This fixes a memory leak.

```
< Irods_Object_Type::undelete_irods_objects definition 1232 > +≡
    temp_strm.str("");
    query_vector.clear();
    query_vector.push_back(sql_strm[0].str());
    query_vector.push_back(sql_strm[1].str());
    query_vector.push_back(sql_strm[2].str());
    query_vector.push_back(sql_strm[3].str());
    affected_rows_vector.push_back(new long int(0UL));
    affected_rows_vector.push_back(new long int(0UL));
    affected_rows_vector.push_back(0);
    affected_rows_vector.push_back(0);
    row_ctr_vector.push_back(0);
    row_ctr_vector.push_back(0);
    row_ctr_vector.push_back(0);
    row_ctr_vector.push_back(0);
    row_ctr_vector.push_back(new unsigned int(0U));
    row_ctr_vector.push_back(new unsigned int(0U));
    field_ctr_vector.push_back(0);
    field_ctr_vector.push_back(0);
    field_ctr_vector.push_back(new unsigned int(0U));
    field_ctr_vector.push_back(new unsigned int(0U));
    status = submit_mysql_queries(query_vector, result_array, mysql_ptr, row_ctr_vector, field_ctr_vector,
        affected_rows_vector, false, thread_str);
```

1241.

```
< Irods_Object_Type::undelete_irods_objects definition 1232 > +≡
  if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "ERROR! In 'Irods_Object_Type::undelete_irods_objects':"
    endl ≪ "'submit_mysql_queries' failed, returning" ≪ status ≪ "."
    endl ≪ "Exiting function unsuccessfully with return value 1."
    endl;
    unlock_cerr_mutex();
  }
  for (int i = 0; i < 4; ++i) {
    if (result_array[i]) mysql_free_result(result_array[i]);
  }
  {Delete pointers and clear vectors 1257}
  return 1;
} /* if (status ≠ 0) */
```

1242.

```
< Irods_Object_Type::undelete_irods_objects definition 1232 > +≡
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr ≪ thread_str ≪ "In 'Irods_Object_Type::undelete_irods_objects':"
      endl ≪ "'submit_mysql_queries' succeeded, returning 0."
      endl;
      for (int i = 0; i < 4; ++i) {
        if (affected_rows_vector[i] ≠ 0)
          cerr ≪ "*affected_rows_vector[" ≪ i ≪ "] == "
          ≪ *affected_rows_vector[i] ≪ endl;
        else cerr ≪ "affected_rows_vector[" ≪ i ≪ "] == 0"
          ≪ endl;
        if (row_ctr_vector[i] ≠ 0)
          cerr ≪ "*row_ctr_vector[" ≪ i ≪ "] == "
          ≪ *row_ctr_vector[i] ≪ endl;
        else cerr ≪ "row_ctr_vector[" ≪ i ≪ "] == 0"
          ≪ endl;
        if (field_ctr_vector[i] ≠ 0)
          cerr ≪ "*field_ctr_vector[" ≪ i ≪ "] == "
          ≪ *field_ctr_vector[i] ≪ endl;
        else cerr ≪ "field_ctr_vector[" ≪ i ≪ "] == 0"
          ≪ endl;
      }
      /* for */
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  if (result_array[0]) {
    mysql_free_result(result_array[0]);
    result_array[0] = 0;
  }
  if (result_array[1]) {
    mysql_free_result(result_array[1]);
    result_array[1] = 0;
  }
  for (int i = 0; i < 4; ++i) {
    sql_strm[i].str("");
  }
  query_vector.clear();
```

1243. Update handles (undelete from archive). [LDF 2013.08.19.]

```

⟨ Irods_Object_Type::undelete_irods_objects definition 1232 ⟩ +≡
  if (*row_ctr_vector[2] > 0) {
#if DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Irods_Object_Type::undelete_irods_objects':" <<
      endl << "'*row_ctr_vector[2]' == " << *row_ctr_vector[2] << "(>0)" << endl <<
      "Will update handles (undeleted from archive)." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  for (int i = 0; i < *row_ctr_vector[2]; ++i) {
    curr_row = mysql_fetch_row(result_array[2]);
    if (curr_row == 0) {
      lock_cerr_mutex();
      cerr << thread_str << "ERROR! In 'Irods_Object_Type::undelete_irods_objects':" <<
        endl << "'mysql_fetch_row' failed, returning 0." << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
      unlock_cerr_mutex();
      mysql_free_result(result_array[2]);
      if (result_array[3]) mysql_free_result(result_array[3]);
      ⟨ Delete pointers and clear vectors 1257 ⟩
      return 1;
    } /* if */
#endif /* DEBUG_COMPILE */
    else
      if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Irods_Object_Type::undelete_irods_objects':" << endl <<
          "'mysql_fetch_row' succeeded." << endl;
        unlock_cerr_mutex();
      } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    temp_val = strtoul(curr_row[0], 0, 10);
    if (temp_val == ULONG_MAX) {
      lock_cerr_mutex();
      cerr << thread_str << "ERROR! In 'Irods_Object_Type::undelete_irods_objects':" <<
        endl << "'strtoul' failed, returning 0." << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
      unlock_cerr_mutex();
      mysql_free_result(result_array[2]);
      if (result_array[3]) mysql_free_result(result_array[3]);
      ⟨ Delete pointers and clear vectors 1257 ⟩
      return 1;
    } /* if (temp_val == ULONG_MAX) */
#endif /* DEBUG_COMPILE */
    else
      if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Irods_Object_Type::undelete_irods_objects':" << endl <<
          "'strtoul' succeeded." << endl << "'temp_val' == " << temp_val << endl;
      }
  }
}

```

```
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    handle_id_set.insert(temp_val);
} /* for */
} /* if (*row_ctr_vector[2] > 0) */
#endif DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In `Irods_Object_Type::undelete_irods_objects':"
        endl << "*row_ctr_vector[2]' == 0" << endl <<
        "Will not update handles (undeleted from archive)." << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
if (result_array[2]) {
    mysql_free_result(result_array[2]);
    result_array[2] = 0;
}
```

1244. Update handles (undelete from gwiridsif database). [LDF 2013.08.19.]

```

⟨ Irods_Object_Type::undelete_irods_objects definition 1232 ⟩ +≡
  if (*row_ctr_vector[3] > 0) {
#if DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Irods_Object_Type::undelete_irods_objects':" <<
      endl << "'*row_ctr_vector[3]' == " << *row_ctr_vector[3] << "(>0)" << endl <<
      "Will update handles (undeleted from 'gwiridsif' database)." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  for (int i = 0; i < *row_ctr_vector[3]; ++i) {
    curr_row = mysql_fetch_row(result_array[3]);
    if (curr_row == 0) {
      lock_cerr_mutex();
      cerr << thread_str << "ERROR! In 'Irods_Object_Type::undelete_irods_objects':" <<
        endl << "'mysql_fetch_row' failed, returning 0." << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
      unlock_cerr_mutex();
      mysql_free_result(result_array[3]);
      if (result_array[3]) mysql_free_result(result_array[3]);
      ⟨ Delete pointers and clear vectors 1257 ⟩
      return 1;
    } /* if */
#endif /* DEBUG_COMPILE */
    else
      if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Irods_Object_Type::undelete_irods_objects':" << endl <<
          "'mysql_fetch_row' succeeded." << endl;
        unlock_cerr_mutex();
      } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    temp_val = strtoul(curr_row[0], 0, 10);
    if (temp_val == ULONG_MAX) {
      lock_cerr_mutex();
      cerr << thread_str << "ERROR! In 'Irods_Object_Type::undelete_irods_objects':" <<
        endl << "'strtoul' failed, returning 0." << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
      unlock_cerr_mutex();
      mysql_free_result(result_array[3]);
      if (result_array[3]) mysql_free_result(result_array[3]);
      ⟨ Delete pointers and clear vectors 1257 ⟩
      return 1;
    } /* if (temp_val == ULONG_MAX) */
#endif /* DEBUG_COMPILE */
    else
      if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Irods_Object_Type::undelete_irods_objects':" << endl <<
          "'strtoul' succeeded." << endl << "'temp_val' == " << temp_val << endl;
      }
  }
}

```

```

        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    handle_id_set.insert(temp_val);
} /* for */
} /* if (*row_ctr_vector[3] > 0) */
#endif DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'Irods_Object_Type::undelete_irods_objects':" <<
        endl << '*row_ctr_vector[3]' == 0" << endl <<
        "Will not update handles (undeleted from 'gwirdsif' database)." << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
if (result_array[3]) {
    mysql_free_result(result_array[3]);
    result_array[3] = 0;
}
}
```

1245.

```

⟨ Irods_Object_Type::undelete_irods_objects definition 1232 ⟩ +≡
if (archive) {
    handle_value_type_vector.push_back("IRODS_OBJECT_MARKED_FOR_DELETION_FROM_ARCHIVE");
    handle_value_type_vector.push_back("DC_METADATA_IRODS_OBJECT_MARKED_FOR_DEL\"
        ETION_FROM_ARCHIVE");
}
if (database) {
    handle_value_type_vector.push_back("IRODS_OBJECT_MARKED_FOR_DELETION_FROM_GWIRDSIF_DB");
    handle_value_type_vector.push_back("DC_METADATA_IRODS_OBJECT_MARKED_FOR_DEL\"
        ETION_FROM_GWIRDSIF_DB");
}
status = Handle_Type::delete_handle_values(handle_id_set, handle_value_type_vector, mysql_ptr,
    thread_str);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Irods_Object_Type::undelete_irods_objects':" << endl <<
        "'Handle_Type::delete_handle_values' failed, returning " << status << "." << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    ⟨ Delete pointers and clear vectors 1257 ⟩
    return 1;
} /* if (status ≠ 0) */
```

1246.

```
< Irods_Object_Type::undelete_irods_objects definition 1232 > +≡
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_str << "In `Irods_Object_Type::undelete_irods_objects':"
                << endl <<
                "'Handle_Type::delete_handle_values' succeeded, returning 0." << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1247.

```
< Irods_Object_Type::undelete_irods_objects definition 1232 > +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        for (vector<Irods_Object_Type>::iterator iter = irods_object_vector.begin();
            iter != irods_object_vector.end(); ++iter) {
            cerr << "iter->avu_vector.size() == "
                << iter->avu_vector.size() << endl;
            if (iter->avu_vector.size() > 0) cerr << "'iter->avu_vector':"
                << endl;
            for (vector<Irods_AVU_Type>::iterator iter_1 = iter->avu_vector.begin();
                iter_1 != iter->avu_vector.end(); ++iter_1) {
                iter_1->show();
            } /* Inner for */
            if (iter->avu_vector.size() > 0) cerr << endl;
        } /* Outer for */
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1248.

```

⟨ Irods_Object_Type::undelete_irods_objects definition 1232 ⟩ +≡
    temp_strm.str("");
    temp_strm << "export_irodsEnvFile=" << irods_env.filename << ";" a='echo -e \\\""; 
    for (vector<Irods_Object_Type>::iterator iter = irods_object_vector.begin();
        iter != irods_object_vector.end(); ++iter) {
        if (archive)
            temp_strm << "rmw_d" << iter->path << "_MARKED_FOR_DELETION_FROM_ARCHIVE" << "%\\n";
        if (database) temp_strm << "rmw_d" << iter->path << "_" <<
            "MARKED_FOR_DELETION_FROM_GWIRDSIF_DATABASE" << "%\\n";
    } /* for */
    temp_strm << "\\|imeta>&1';echo $?;echo $a";
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In `Irods_Object_Type::undelete_irods_objects':"
            << endl <<
            "'temp_strm.str()' == " << temp_strm.str() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
FILE *fp = popen(temp_strm.str().c_str(), "r");
if (fp == 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In `Irods_Object_Type::undelete_irods_objects':"
        << endl <<
        "'popen' failed, returning 0." << endl << "Call to `imeta' in shell failed."
        << endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    ⟨ Delete pointers and clear vectors 1257 ⟩
    return 1;
} /* if (fp == 0) */

```

1249.

```
< Irods_Object_Type :: undelete_irods_objects definition 1232 > +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Irods_Object_Type::undelete_irods_objects':" << endl <<
            "'popen' succeeded." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
char buffer[1024];
memset(buffer, 0, 1024);
status = fread(buffer, 1, 1024, fp);
if (status == 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Irods_Object_Type::undelete_irods_objects':" <<
        endl << "'fread' failed, returning 0." << endl <<
        "Can't read output of call to 'imeta' in shell." << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    pclose(fp);
    { Delete pointers and clear vectors 1257 }
    return 1;
} /* if (status == 0) */
```

1250.

```
< Irods_Object_Type :: undelete_irods_objects definition 1232 > +≡
if (status == 1024) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'Irods_Object_Type::undelete_irods_objects':" <<
        endl << "'fread' returned 1024: output of 'imeta' in shell exceeds \
        maximum length == 1023." << endl << "This is not permitted." << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    pclose(fp);
    { Delete pointers and clear vectors 1257 }
    return 1;
} /* if (status == 1024) */
```

1251.

```

⟨ Irods_Object_Type::undelete_irods_objects definition 1232 ⟩ +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'Irods_Object_Type::undelete_irods_objects':" << endl <<
            "'fread' succeeded, returning" << status << "." << endl << "'buffer' ==" << buffer <<
            endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    pclose(fp);
    fp = 0;

```

1252.

```

⟨ Irods_Object_Type::undelete_irods_objects definition 1232 ⟩ +≡
    errno = 0;
    status = sscanf(buffer, "%d", &temp_val_1);
    if (status != 1) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'Irods_Object_Type::undelete_irods_objects':" << endl <<
            "'sscanf' failed, returning" << status << "." << endl;
        if (status == EOF) cerr << "'status' == EOF:" << endl << strerror(errno) << endl;
        cerr << "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        ⟨ Delete pointers and clear vectors 1257 ⟩
        return 1;
    } /* if (status != 1) */

```

1253.

```

⟨ Irods_Object_Type::undelete_irods_objects definition 1232 ⟩ +≡
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_str << "In 'Irods_Object_Type::undelete_irods_objects':" << endl <<
                "'sscanf' succeeded, returning 1." << endl << "'temp_val_1' ==" << temp_val_1 << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1254.

```
< Irods_Object_Type::undelete_irods_objects definition 1232 > +≡
if (temp_val_1 ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "ERROR! In 'Irods_Object_Type::undelete_irods_objects':"
    endl ≪ "'temp_val_1' == " ≪ temp_val_1 ≪ " (!= 0)" ≪
    endl ≪ "'imeta' command in shell failed:" ≪ endl ≪
    "Output (preceded by return value of 'imeta' command):" ≪ endl ≪
    buffer ≪ endl ≪ "Exiting function unsuccessfully with return value 1." ≪ endl;
unlock_cerr_mutex();
{ Delete pointers and clear vectors 1257 }
return 1;
} /* if (temp_val_1 ≠ 0) */
```

1255.

```
< Irods_Object_Type::undelete_irods_objects definition 1232 > +≡
#ifndef DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "In 'Irods_Object_Type::undelete_irods_objects':"
    ≪ endl ≪ "'temp_val_1' == 0" ≪ endl ≪ "'imeta' command in shell succeeded, returning 0."
    ≪ endl ≪ "Deleted AVU(s) successfully." ≪ endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "Exiting 'Irods_Object_Type::undelete_irods_objects' "
    ≪ "successfully with return value 0." ≪ endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1256.

```
< Irods_Object_Type::undelete_irods_objects definition 1232 > +≡
{ Delete pointers and clear vectors 1257 }
return 0; } /* End of Irods_Object_Type::undelete_irods_objects definition */
```

1257. Delete pointers and clear vectors. [LDF 2013.09.03.]

Log

[LDF 2013.09.03.] Added this section.

```

⟨Delete pointers and clear vectors 1257⟩ ≡
for (vector<unsigned int *>::iterator iter = row_ctr_vector.begin(); iter ≠ row_ctr_vector.end();
    ++iter) {
    delete *iter;
    *iter = 0;
}
row_ctr_vector.clear();
for (vector<unsigned int *>::iterator iter = field_ctr_vector.begin(); iter ≠ field_ctr_vector.end();
    ++iter) {
    delete *iter;
    *iter = 0;
}
field_ctr_vector.clear();
for (vector<long int *>::iterator iter = affected_rows_vector.begin(); iter ≠ affected_rows_vector.end();
    ++iter) {
    delete *iter;
    *iter = 0;
}
affected_rows_vector.clear();

```

This code is cited in section 1240.

This code is used in sections 1241, 1243, 1244, 1245, 1248, 1249, 1250, 1252, 1254, and 1256.

1258. Putting iRODS_Object_Type together. [LDF 2012.10.12.]

```

⟨Include files 3⟩
using namespace std;
using namespace gwrdifpk;

⟨Irods_Object_Type constructor definitions 1040⟩
⟨Irods_Object_Type::set definitions 1044⟩
⟨Irods_Object_Type::get_avus_from_irods_system definition 1046⟩
⟨Irods_Object_Type::clear definition 1059⟩
⟨Irods_Object_Type::show definition 1061⟩
⟨Irods_Object_Type::write_to_database definition 1063⟩
⟨Irods_Object_Type::get_from_database definition 1077⟩
⟨Irods_Object_Type::update definition 1120⟩
⟨Irods_Object_Type::put_irods_object definition 1133⟩
⟨Irods_Object_Type::add_avu definition 1141⟩
⟨Irods_Object_Type::add_avu_cond definition 1150⟩
⟨Irods_Object_Type::mark_for_deletion definition 1154⟩
⟨Irods_Object_Type::delete_from_archive definition 1174⟩
⟨Irods_Object_Type::delete_from_gwirdsif_db definition 1206⟩
⟨Irods_Object_Type::add_handle_value definition 1224⟩
⟨Irods_Object_Type::find_avu definition 1230⟩
⟨Irods_Object_Type::undelete_irods_objects definition 1232⟩

```

1259. This is what's written to the header file irdsobtp.h. [LDF 2008.12.05.]

```
<irdsobtp.h 1259> ==
#ifndef IRDSOBTP_H
#define IRDSOBTP_H 1
using namespace std;
using namespace gwrdifpk;
< Declare class Irods_Object_Type 1038 >
#endif
```

1260. class Dublin_Core_Metadata_Type.

Log

[LDF 2012.12.06.] Added this file.

1261. Include files.

```
<Include files 3> +≡
#include <stdlib.h>      /* Standard Library for C */
#include <stdio.h>
#include <string.h>
#include <climits.h>
#include <errno.h>
#include <expat.h>
#include <fstream>        /* Standard Template Library (STL) for C++ */
#include <iomanip>
#include <iostream>
#include <map>
#include <vector>
#include <stack>
#include <string>
#if 0
#include <time.h>
#include <math.h>
#endif
#include <sstream>
#include <pthread.h>      /* POSIX threads */
#include <mysql.h>
#if HAVE_CONFIG_H
#include <config.h>
#endif
#include "glblcnst.h++"
#include "glblvrbl.h++"
#include "utilfncts.h++"
#include "parser.h++"
#include "scanner.h++"
```

1262. Preprocessor macro definitions. [LDF 2012.12.14.]

Log

[LDF 2013.03.14.] Removed the definition of BUFSIZE.

```
⟨ Preprocessor macro definitions 1262 ⟩ ≡
#define XML_LARGE_SIZE
#if defined (XML_USE_MSC_EXTENSIONS) ∧ _MSC_VER < 1400
#define XML_FMT_INT_MOD "I64"
#else
#define XML_FMT_INT_MOD "11"
#endif
#else
#define XML_FMT_INT_MOD "1"
#endif
```

See also section 1370.

This code is used in sections 1333, 1409, and 1410.

1263.

Log

[LDF 2013.03.14.] Removed the declaration of **static char Buff**.

```
⟨ File-local variables 1263 ⟩ ≡
```

```
static int Depth;
static stringstream last_value;
```

This code is used in section 1333.

1264. class Dublin_Core_Metadata_Type. [LDF 2012.12.06.]

Log

[LDF 2012.12.06.] Added this type declaration.

```
(class Dublin_Core_Metadata_Type declaration 1264) ≡
class Dublin_Core_Metadata_Type { friend int yyparse(yyvsp<parameter>);  

friend class Scan_Parse_Parameter_Type;  

friend int exchange_data_with_server(Scan_Parse_Parameter_Type &);  

friend int exchange_data_with_client(Scan_Parse_Parameter_Type &);  

private: unsigned long int id;  

    string irods_object_path;  

    unsigned int user_id;  

    unsigned long int handle_id;  

    multimap<unsigned int, Dublin_Core_Metadata_Sub_Type> metadata_sub_map; stack <  

        Dublin_Core_Metadata_Sub_Type > metadata_sub_stack;  

public: static map<unsigned int, string> element_map;  

    static map<string, unsigned int> element_ctr_map;  

    static map<unsigned int, string> qualifier_map;  

    static map<string, unsigned int> qualifier_ctr_map;  

    static const unsigned int DUBLIN_CORE_NULL_TYPE;  

    static const unsigned int DUBLIN_CORE_ELEMENT_TYPE;  

    static const unsigned int DUBLIN_CORE_QUALIFIER_TYPE;  

    static const unsigned int DUBLIN_CORE_ATTRIBUTE_TYPE;  

    static const unsigned int DUBLIN_CORE_NULL_ELEMENT;  

    static const unsigned int DUBLIN_CORE_TITLE_ELEMENT;  

    static const unsigned int DUBLIN_CORE_CREATOR_ELEMENT;  

    static const unsigned int DUBLIN_CORE SUBJECT_ELEMENT;  

    static const unsigned int DUBLIN_CORE_DESCRIPTION_ELEMENT;  

    static const unsigned int DUBLIN_CORE_PUBLISHER_ELEMENT;  

    static const unsigned int DUBLIN_CORE_CONTRIBUTOR_ELEMENT;  

    static const unsigned int DUBLIN_CORE_DATE_ELEMENT;  

    static const unsigned int DUBLIN_CORE_TYPE_ELEMENT;  

    static const unsigned int DUBLIN_CORE_FORMAT_ELEMENT;  

    static const unsigned int DUBLIN_CORE_IDENTIFIER_ELEMENT;  

    static const unsigned int DUBLIN_CORE_SOURCE_ELEMENT;  

    static const unsigned int DUBLIN_CORE_LANGUAGE_ELEMENT;  

    static const unsigned int DUBLIN_CORE_RELATION_ELEMENT;  

    static const unsigned int DUBLIN_CORE_COVERAGE_ELEMENT;  

    static const unsigned int DUBLIN_CORE_RIGHTS_ELEMENT;  

    static const unsigned int DUBLIN_CORE_NULL_TERM;  

    static const unsigned int DUBLIN_CORE_ABSTRACT_TERM;  

    static const unsigned int DUBLIN_CORE_ACCESSRIGHTS_TERM;  

    static const unsigned int DUBLIN_CORE_ACCRUALMETHOD_TERM;  

    static const unsigned int DUBLIN_CORE_ACCRUALPERIODICITY_TERM;  

    static const unsigned int DUBLIN_CORE_ACCRUALPOLICY_TERM;  

    static const unsigned int DUBLIN_CORE_ALTERNATIVE_TERM;  

    static const unsigned int DUBLIN_CORE_AUDIENCE_TERM;  

    static const unsigned int DUBLIN_CORE_AVAILABLE_TERM;  

    static const unsigned int DUBLIN_CORE_BIBLIOGRAPHICCITATION_TERM;  

    static const unsigned int DUBLIN_CORE_CONFORMSTO_TERM;  

    static const unsigned int DUBLIN_CORE_CONTRIBUTOR_TERM;
```

```
static const unsigned int DUBLIN_CORE_COVERAGE_TERM;
static const unsigned int DUBLIN_CORE_CREATED_TERM;
static const unsigned int DUBLIN_CORE_CREATOR_TERM;
static const unsigned int DUBLIN_CORE_DATE_TERM;
static const unsigned int DUBLIN_CORE_DATEACCEPTED_TERM;
static const unsigned int DUBLIN_CORE_DATECOPYRIGHTED_TERM;
static const unsigned int DUBLIN_CORE_DATESUBMITTED_TERM;
static const unsigned int DUBLIN_CORE_DESCRIPTION_TERM;
static const unsigned int DUBLIN_CORE_EDUCATIONLEVEL_TERM;
static const unsigned int DUBLIN_CORE_EXTENT_TERM;
static const unsigned int DUBLIN_CORE_FORMAT_TERM;
static const unsigned int DUBLIN_CORE_HASFORMAT_TERM;
static const unsigned int DUBLIN_CORE_HASPART_TERM;
static const unsigned int DUBLIN_CORE_HASVERSION_TERM;
static const unsigned int DUBLIN_CORE_IDENTIFIER_TERM;
static const unsigned int DUBLIN_CORE_INSTRUCTIONALMETHOD_TERM;
static const unsigned int DUBLIN_CORE_ISFORMATOF_TERM;
static const unsigned int DUBLIN_CORE_ISPARTOF_TERM;
static const unsigned int DUBLIN_CORE_ISREFERENCEDBY_TERM;
static const unsigned int DUBLIN_CORE_ISREPLACEDBY_TERM;
static const unsigned int DUBLIN_CORE_ISREQUIREDBY_TERM;
static const unsigned int DUBLIN_CORE_ISSUED_TERM;
static const unsigned int DUBLIN_CORE_ISVERSIONOF_TERM;
static const unsigned int DUBLIN_CORE_LANGUAGE_TERM;
static const unsigned int DUBLIN_CORE_LICENSE_TERM;
static const unsigned int DUBLIN_CORE_MEDIATOR_TERM;
static const unsigned int DUBLIN_CORE_MEDIUM_TERM;
static const unsigned int DUBLIN_CORE_MODIFIED_TERM;
static const unsigned int DUBLIN_CORE_PROVENANCE_TERM;
static const unsigned int DUBLIN_CORE_PUBLISHER_TERM;
static const unsigned int DUBLIN_CORE_REFERENCES_TERM;
static const unsigned int DUBLIN_CORE_RELATION_TERM;
static const unsigned int DUBLIN_CORE_REPLACESTERM;
static const unsigned int DUBLIN_CORE_REQUIRESTERM;
static const unsigned int DUBLIN_CORE_RIGHTS_TERM;
static const unsigned int DUBLIN_CORE_RIGHTSHOLDER_TERM;
static const unsigned int DUBLIN_CORE_SOURCE_TERM;
static const unsigned int DUBLIN_CORE_SPATIAL_TERM;
static const unsigned int DUBLIN_CORE SUBJECT_TERM;
static const unsigned int DUBLIN_CORE_TABLEOFCONTENTS_TERM;
static const unsigned int DUBLIN_CORE_TEMPORAL_TERM;
static const unsigned int DUBLIN_CORE_TITLE_TERM;
static const unsigned int DUBLIN_CORE_TYPE_TERM;
static const unsigned int DUBLIN_CORE_VALID_TERM;
{ Dublin_Core_Metadata_Type function declarations 1266 }
};
```

This code is used in sections 1333 and 1335.

1265.

Log

[LDF 2012.12.12.] Added this section.

```
⟨ Initialize static Dublin_Core_Metadata_Type data members 1265 ⟩ ≡
map<unsigned int, string> Dublin_Core_Metadata_Type::element_map;
map<string, unsigned int> Dublin_Core_Metadata_Type::element_ctr_map;
map<unsigned int, string> Dublin_Core_Metadata_Type::qualifier_map;
map<string, unsigned int> Dublin_Core_Metadata_Type::qualifier_ctr_map;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_NULL_TYPE = 0;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_ELEMENT_TYPE = 1;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_QUALIFIER_TYPE = 2;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_NULL_ELEMENT = 0;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_TITLE_ELEMENT = 1;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_CREATOR_ELEMENT = 2;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE SUBJECT_ELEMENT = 3;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_DESCRIPTION_ELEMENT = 4;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_PUBLISHER_ELEMENT = 5;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_CONTRIBUTOR_ELEMENT = 6;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_DATE_ELEMENT = 7;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_TYPE_ELEMENT = 8;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_FORMAT_ELEMENT = 9;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_IDENTIFIER_ELEMENT = 10;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_SOURCE_ELEMENT = 11;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_LANGUAGE_ELEMENT = 12;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_RELATION_ELEMENT = 13;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_COVERAGE_ELEMENT = 14;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_RIGHTS_ELEMENT = 15;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_NULL_TERM = 100;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_ABSTRACT_TERM = 101;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_ACCESSRIGHTS_TERM = 102;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_ACCRUALMETHOD_TERM = 103;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_ACCRUALPERIODICITY_TERM = 104;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_ACCRUALPOLICY_TERM = 105;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_ALTERNATIVE_TERM = 106;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_AUDIENCE_TERM = 107;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_AVAILABLE_TERM = 108;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_BIBLIOGRAPHICCITATION_TERM =
    109;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_CONFORMSTO_TERM = 110;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_CONTRIBUTOR_TERM = 111;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_COVERAGE_TERM = 112;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_CREATED_TERM = 113;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_CREATOR_TERM = 114;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_DATE_TERM = 115;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_DATEACCEPTED_TERM = 116;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_DATECOPYRIGHTED_TERM = 117;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_DATESUBMITTED_TERM = 118;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_DESCRIPTION_TERM = 119;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_EDUCATIONLEVEL_TERM = 120;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_EXTENT_TERM = 121;
```

```

const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_FORMAT_TERM = 122;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_HASFORMAT_TERM = 123;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_HASPART_TERM = 124;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_HASVERSION_TERM = 125;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_IDENTIFIER_TERM = 126;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_INSTRUCTIONALMETHOD_TERM =
    127;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_ISFORMATOF_TERM = 128;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_ISPARTOF_TERM = 129;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_ISREFERENCEDBY_TERM = 130;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_ISREPLACEDBY_TERM = 131;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_ISREQUIREDBY_TERM = 132;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_ISSUED_TERM = 133;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_ISVERSIONOF_TERM = 134;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_LANGUAGE_TERM = 135;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_LICENSE_TERM = 136;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_MEDIATOR_TERM = 137;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_MEDIUM_TERM = 138;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_MODIFIED_TERM = 139;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_PROVENANCE_TERM = 140;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_PUBLISHER_TERM = 141;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_REFERENCES_TERM = 142;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_RELATION_TERM = 143;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_REPLACESTERM = 144;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_REQUIRESTERM = 145;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_RIGHTS_TERM = 146;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_RIGHTSHOLDER_TERM = 147;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_SOURCE_TERM = 148;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_SPATIAL_TERM = 149;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE SUBJECT_TERM = 150;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_TABLEOFCONTENTS_TERM = 151;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_TEMPORAL_TERM = 152;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_TITLE_TERM = 153;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_TYPE_TERM = 154;
const unsigned int Dublin_Core_Metadata_Type::DUBLIN_CORE_VALID_TERM = 155;

```

This code is used in section 1333.

1266. Default constructor. [LDF 2012.12.06.]

⟨ **Dublin_Core_Metadata_Type** function declarations 1266 ⟩ ≡
Dublin_Core_Metadata_Type(**void**);

See also sections 1268, 1270, 1272, 1274, 1276, 1278, 1282, 1286, 1291, 1295, 1297, and 1313.

This code is used in section 1264.

1267.

⟨ **Dublin_Core_Metadata_Type** constructor definition 1267 ⟩ ≡
Dublin_Core_Metadata_Type::**Dublin_Core_Metadata_Type**(**void**)
{
id = *handle_id* = *user_id* = 0;
return;
}

This code is used in section 1333.

1268. Destructor. [LDF 2012.12.06.]

Log

[LDF 2012.12.06.] Added this function.

`< Dublin_Core_Metadata_Type function declarations 1266 > +≡
 ~Dublin_Core_Metadata_Type(void);`

1269.

`< Dublin_Core_Metadata_Type destructor definition 1269 > ≡
 Dublin_Core_Metadata_Type :: ~Dublin_Core_Metadata_Type(void)
 {
 return;
 }`

This code is used in section 1333.

1270. Clear. [LDF 2012.12.06.]

Log

[LDF 2012.12.06.] Added this function.

`< Dublin_Core_Metadata_Type function declarations 1266 > +≡
 void clear(void);`

1271.

`< Dublin_Core_Metadata_Type :: clear definition 1271 > ≡
 void Dublin_Core_Metadata_Type :: clear(void)
 {
 irods_object_path = "";
 id = handle_id = user_id = 0;
 metadata_sub_map.clear();
 } /* Dublin_Core_Metadata_Type :: clear */`

This code is used in section 1333.

1272. Equality operator. [LDF 2012.12.21.]

Log

[LDF 2012.12.21.] Added this function.

`< Dublin_Core_Metadata_Type function declarations 1266 > +≡
 bool operator=(const Dublin_Core_Metadata_Type &d) const;`

1273.

```

⟨ Dublin_Core_Metadata_Type::operator≡ definition 1273 ⟩ ≡
    bool Dublin_Core_Metadata_Type::operator≡(const Dublin_Core_Metadata_Type &d) const
    {
        bool DEBUG = false;      /* true */
        set_debug_level(DEBUG, 0, 0);
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "Entering " 'Dublin_Core_Metadata_Type::operator==' . " " << endl << "user_id" == " " <<
                user_id << endl << "d.user_id" == " " << d.user_id << endl << "irods_object_path" == " " <<
                irods_object_path << endl << "d.irods_object_path" == " " << d.irods_object_path <<
                endl << "metadata_sub_map.size()" == " " << metadata_sub_map.size() << endl <<
                "d.metadata_sub_map.size()" == " " << d.metadata_sub_map.size() << endl;
            unlock_cerr_mutex();
        }      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */
        if (! (user_id == d.user_id & irods_object_path == d.irods_object_path & metadata_sub_map.size() ==
               d.metadata_sub_map.size())) {
#ifndef DEBUG_COMPILE
            if (DEBUG) {
                lock_cerr_mutex();
                cerr << "Returning false." << endl;
                unlock_cerr_mutex();
            }      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */
            return false;
        }
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "Will check maps." << endl;
            unlock_cerr_mutex();
        }      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */
        multimap<unsigned int,
                  Dublin_Core_Metadata_Sub_Type>::const_iterator iter = metadata_sub_map.begin();
        multimap<unsigned int,
                  Dublin_Core_Metadata_Sub_Type>::const_iterator iter_1 = d.metadata_sub_map.begin();
        for ( ; iter != metadata_sub_map.end() & iter_1 != d.metadata_sub_map.end(); ++iter, ++iter_1) {
            if (iter->second != iter_1->second) {
#ifndef DEBUG_COMPILE
                if (DEBUG) {
                    lock_cerr_mutex();
                    cerr << "Subtype objects don't match:" << endl << "'iter->second'" == " " << endl;
                    iter->second.show();
                    cerr << endl << "'iter_1->second'" == " " << endl;
                    iter_1->second.show();
                    cerr << endl << "Returning false." << endl;
                    unlock_cerr_mutex();
                }      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */
            }
        }
    }

```

```

        return false;
    }
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Subtype\u00d7objects\u00d7match.\u00d7Continuing." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* for */
return true;
} /* End of Dublin_Core_Metadata_Type::operator≡ definition */

```

This code is used in section 1333.

1274. Show. [LDF 2012.12.06.]

Log

[LDF 2012.12.06.] Added this function.

⟨Dublin_Core_Metadata_Type function declarations 1266⟩ +≡
int show(string s = "") const;

1275.

⟨Dublin_Core_Metadata_Type::show definition 1275⟩ ≡
int Dublin_Core_Metadata_Type::show(string s) const
{
 if (s.size() == 0) s = "Dublin_Core_Metadata_Type:";
 cerr << s << endl << "id==\u0000000000000000" << id << endl << "handle_id==\u0000000000" <<
 handle_id << endl << "user_id==\u0000000000" << user_id << endl <<
 "irods_object_path==\u0000" << irods_object_path << endl;
 if (metadata_sub_map.size() == 0) cerr << "metadata_sub_map is empty." << endl;
 else {
 cerr << "metadata_sub_map:" << endl;
 for (multimap<unsigned int,
 Dublin_Core_Metadata_Sub_Type>::const_iterator iter = metadata_sub_map.begin();
 iter != metadata_sub_map.end(); ++iter) {
 iter->second.show();
 }
 }
 cerr << endl;
 return 0;
} /* Dublin_Core_Metadata_Type::show */

This code is used in section 1333.

1276. Initialize maps. [LDF 2012.12.06.]

Log

[LDF 2012.12.06.] Added this function.

⟨ **Dublin_Core_Metadata_Type** function declarations 1266 ⟩ +≡

static int *initialize_maps(void)*;

1277.

```

⟨ Dublin_Core_Metadata_Type::initialize_maps definition 1277 ⟩ ≡
int Dublin_Core_Metadata_Type::initialize_maps(void)
{
    bool DEBUG = false;      /* true */
    set_debug_level(DEBUG, 0, 0);
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Entering " ‘Dublin_Core_Metadata_Type::initialize_maps’ . " << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    element_map[0] = "NULL_DUBLIN_CORE_ELEMENT";
    element_map[1] = "title";
    element_map[2] = "creator";
    element_map[3] = "subject";
    element_map[4] = "description";
    element_map[5] = "publisher";
    element_map[6] = "contributor";
    element_map[7] = "date";
    element_map[8] = "type";
    element_map[9] = "format";
    element_map[10] = "identifier";
    element_map[11] = "source";
    element_map[12] = "language";
    element_map[13] = "relation";
    element_map[14] = "coverage";
    element_map[15] = "rights";
    element_ctr_map["NULL_DUBLIN_CORE_ELEMENT"] = 0;
    element_ctr_map["title"] = 1;
    element_ctr_map["creator"] = 2;
    element_ctr_map["subject"] = 3;
    element_ctr_map["description"] = 4;
    element_ctr_map["publisher"] = 5;
    element_ctr_map["contributor"] = 6;
    element_ctr_map["date"] = 7;
    element_ctr_map["type"] = 8;
    element_ctr_map["format"] = 9;
    element_ctr_map["identifier"] = 10;
    element_ctr_map["source"] = 11;
    element_ctr_map["language"] = 12;
    element_ctr_map["relation"] = 13;
    element_ctr_map["coverage"] = 14;
    element_ctr_map["rights"] = 15;
    qualifier_map[0] = qualifier_map[100] = "NULL_DUBLIN_CORE_TERM";
    qualifier_map[1] = qualifier_map[101] = "abstract";
    qualifier_map[2] = qualifier_map[102] = "accessRights";
    qualifier_map[3] = qualifier_map[103] = "accrualMethod";
    qualifier_map[4] = qualifier_map[104] = "accrualPeriodicity";
    qualifier_map[5] = qualifier_map[105] = "accrualPolicy";
    qualifier_map[6] = qualifier_map[106] = "alternative";
}

```

```
qualifier_map[7] = qualifier_map[107] = "audience";
qualifier_map[8] = qualifier_map[108] = "available";
qualifier_map[9] = qualifier_map[109] = "bibliographicCitation";
qualifier_map[10] = qualifier_map[110] = "conformsTo";
qualifier_map[11] = qualifier_map[111] = "contributor";
qualifier_map[12] = qualifier_map[112] = "coverage";
qualifier_map[13] = qualifier_map[113] = "created";
qualifier_map[14] = qualifier_map[114] = "creator";
qualifier_map[15] = qualifier_map[115] = "date";
qualifier_map[16] = qualifier_map[116] = "dateAccepted";
qualifier_map[17] = qualifier_map[117] = "dateCopyrighted";
qualifier_map[18] = qualifier_map[118] = "dateSubmitted";
qualifier_map[19] = qualifier_map[119] = "description";
qualifier_map[20] = qualifier_map[120] = "educationLevel";
qualifier_map[21] = qualifier_map[121] = "extent";
qualifier_map[22] = qualifier_map[122] = "format";
qualifier_map[23] = qualifier_map[123] = "hasFormat";
qualifier_map[24] = qualifier_map[124] = "hasPart";
qualifier_map[25] = qualifier_map[125] = "hasVersion";
qualifier_map[26] = qualifier_map[126] = "identifier";
qualifier_map[27] = qualifier_map[127] = "instructionalMethod";
qualifier_map[28] = qualifier_map[128] = "isFormatOf";
qualifier_map[29] = qualifier_map[129] = "isPartOf";
qualifier_map[30] = qualifier_map[130] = "isReferencedBy";
qualifier_map[31] = qualifier_map[131] = "isReplacedBy";
qualifier_map[32] = qualifier_map[132] = "isRequiredBy";
qualifier_map[33] = qualifier_map[133] = "issued";
qualifier_map[34] = qualifier_map[134] = "isVersionOf";
qualifier_map[35] = qualifier_map[135] = "language";
qualifier_map[36] = qualifier_map[136] = "license";
qualifier_map[37] = qualifier_map[137] = "mediator";
qualifier_map[38] = qualifier_map[138] = "medium";
qualifier_map[39] = qualifier_map[139] = "modified";
qualifier_map[40] = qualifier_map[140] = "provenance";
qualifier_map[41] = qualifier_map[141] = "publisher";
qualifier_map[42] = qualifier_map[142] = "references";
qualifier_map[43] = qualifier_map[143] = "relation";
qualifier_map[44] = qualifier_map[144] = "replaces";
qualifier_map[45] = qualifier_map[145] = "requires";
qualifier_map[46] = qualifier_map[146] = "rights";
qualifier_map[47] = qualifier_map[147] = "rightsHolder";
qualifier_map[48] = qualifier_map[148] = "source";
qualifier_map[49] = qualifier_map[149] = "spatial";
qualifier_map[50] = qualifier_map[150] = "subject";
qualifier_map[51] = qualifier_map[151] = "tableOfContents";
qualifier_map[52] = qualifier_map[152] = "temporal";
qualifier_map[53] = qualifier_map[153] = "title";
qualifier_map[54] = qualifier_map[154] = "type";
qualifier_map[55] = qualifier_map[155] = "valid";
qualifier_ctr_map["NULL_DUBLIN_CORE_TERM"] = 0;
qualifier_ctr_map["abstract"] = 1;
qualifier_ctr_map["accessRights"] = 2;
```

```
qualifier_ctr_map["accrualMethod"] = 3;
qualifier_ctr_map["accrualPeriodicity"] = 4;
qualifier_ctr_map["accrualPolicy"] = 5;
qualifier_ctr_map["alternative"] = 6;
qualifier_ctr_map["audience"] = 7;
qualifier_ctr_map["available"] = 8;
qualifier_ctr_map["bibliographicCitation"] = 9;
qualifier_ctr_map["conformsTo"] = 10;
qualifier_ctr_map["contributor"] = 11;
qualifier_ctr_map["coverage"] = 12;
qualifier_ctr_map["created"] = 13;
qualifier_ctr_map["creator"] = 14;
qualifier_ctr_map["date"] = 15;
qualifier_ctr_map["dateAccepted"] = 16;
qualifier_ctr_map["dateCopyrighted"] = 17;
qualifier_ctr_map["dateSubmitted"] = 18;
qualifier_ctr_map["description"] = 19;
qualifier_ctr_map["educationLevel"] = 20;
qualifier_ctr_map["extent"] = 21;
qualifier_ctr_map["format"] = 22;
qualifier_ctr_map["hasFormat"] = 23;
qualifier_ctr_map["hasPart"] = 24;
qualifier_ctr_map["hasVersion"] = 25;
qualifier_ctr_map["identifier"] = 26;
qualifier_ctr_map["instructionalMethod"] = 27;
qualifier_ctr_map["isFormatOf"] = 28;
qualifier_ctr_map["isPartOf"] = 29;
qualifier_ctr_map["isReferencedBy"] = 30;
qualifier_ctr_map["isReplacedBy"] = 31;
qualifier_ctr_map["isRequiredBy"] = 32;
qualifier_ctr_map["issued"] = 33;
qualifier_ctr_map["isVersionOf"] = 34;
qualifier_ctr_map["language"] = 35;
qualifier_ctr_map["license"] = 36;
qualifier_ctr_map["mediator"] = 37;
qualifier_ctr_map["medium"] = 38;
qualifier_ctr_map["modified"] = 39;
qualifier_ctr_map["provenance"] = 40;
qualifier_ctr_map["publisher"] = 41;
qualifier_ctr_map["references"] = 42;
qualifier_ctr_map["relation"] = 43;
qualifier_ctr_map["replaces"] = 44;
qualifier_ctr_map["requires"] = 45;
qualifier_ctr_map["rights"] = 46;
qualifier_ctr_map["rightsHolder"] = 47;
qualifier_ctr_map["source"] = 48;
qualifier_ctr_map["spatial"] = 49;
qualifier_ctr_map["subject"] = 50;
qualifier_ctr_map["tableOfContents"] = 51;
qualifier_ctr_map["temporal"] = 52;
qualifier_ctr_map["title"] = 53;
qualifier_ctr_map["type"] = 54;
```

```

    qualifier_ctr_map["valid"] = 55;
#if DEBUG_COMPILE
    if (DEBUG) {
        stringstream s;
        lock_cerr_mutex();
        for (map<unsigned int, string>::const_iterator iter = element_map.begin());
            iter ≠ element_map.end(); ++iter) {
                s ≪ iter-first ≪ :;
                cerr ≪ setw(4) ≪ std::left ≪ s.str() ≪ iter-second ≪ endl;
                s.str(");
            }
            cerr ≪ endl;
        for (map<unsigned int, string>::const_iterator iter = qualifier_map.begin());
            iter ≠ qualifier_map.end(); ++iter) {
                s ≪ iter-first ≪ :;
                cerr ≪ setw(4) ≪ std::left ≪ s.str() ≪ iter-second ≪ endl;
                s.str(");
            }
            cerr ≪ "Exiting ‘Dublin_Core_Metadata_Type::initialize_maps’" ≪
                "successfully with return value 0." ≪ endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
    #endif /* DEBUG_COMPILE */
    return 0;
} /* Dublin_Core_Metadata_Type::initialize_maps */

```

This code is used in section 1333.

1278. Output. [LDF 2012.12.07.]

Log

[LDF 2012.12.07.] Added this function.

⟨ **Dublin_Core_Metadata_Type** function declarations 1266 ⟩ +≡
int output(ofstream &out strm) const;

1279.

⟨ **Dublin_Core_Metadata_Type**::*output* definition 1279 ⟩ ≡
int Dublin_Core_Metadata_Type::output(ofstream &out strm) const { bool DEBUG = false;
 / true */*
 set_debug_level(DEBUG, 0, 0);
 stringstream temp_strm;
#if DEBUG_COMPILE
 if (DEBUG) {
 lock_cerr_mutex();
 cerr ≪ "Entering ‘Dublin_Core_Metadata_Type::output’." ≪ endl ≪
 “metadata_sub_map.size()” ≪ metadata_sub_map.size() ≪ endl;
 unlock_cerr_mutex();
 } / if (DEBUG) */*
#endif / DEBUG_COMPILE */*

See also sections 1280 and 1281.

This code is used in section 1333.

1280.

```

⟨ Dublin_Core_Metadata_Type :: output definition 1279 ⟩ +≡
  if (metadata_sub_map.size() == 0) {
    cerr << "WARNING! In 'Dublin_Core_Metadata_Type::output':"
    << endl << "'metadata_sub_map.size()' == 0. No metadata to output."
    << endl << "Exiting function unsuccessfully with return value 2."
    return 2;
  } /* if (metadata_sub_map.size() == 0) */
out_strm << "<metadata" << endl << "  xmlns=\"http://example.org/myapp/\""
<< endl << "  xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\""
<< endl << "  xsi:schemaLocation=\"http://example.org/myapp/\""
<< "http://example.org/myapp/schema.xsd\""
<< endl << "  xmlns:dc=\"http://purl.org/dc/elements/1.1/\""
<< endl << "  xmlns:dcterms=\"http://purl.org/dc/terms/\">" << endl << endl;
for (multimap<unsigned int,
  Dublin_Core_Metadata_Sub_Type>::const_iterator iter = metadata_sub_map.begin();
  iter != metadata_sub_map.end(); ++iter) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "'iter->first' == "
    << iter->first << endl;
    iter->second.show();
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  temp_strm.str("");
  if (iter->second.element_id > 0) temp_strm << "  <dc:"
  << element_map[iter->second.element_id];
  else if (iter->second.qualifier_id > 0)
    temp_strm << "  <dcterms:"
    << qualifier_map[iter->second.qualifier_id];
  for (multimap<string, string>::const_iterator iter_1 =
    iter->second.attribute_map.begin();
    iter_1 != iter->second.attribute_map.end(); ++iter_1)
    temp_strm << "    <"
    << iter_1->first << "=\""
    << iter_1->second << "\"";
  if (iter->second.element_id > 0) temp_strm << ">" << endl;
  else if (iter->second.qualifier_id > 0) temp_strm << ">" << endl;
  temp_strm << "  <\""
  << iter->second.value << endl;
  if (iter->second.element_id > 0)
    temp_strm << "  </dc:"
    << element_map[iter->second.element_id] << ">" << endl;
  else if (iter->second.qualifier_id > 0)
    temp_strm << "  </dcterms:"
    << qualifier_map[iter->second.qualifier_id] << ">" << endl;
  out_strm << temp_strm.str();
  temp_strm.str("");
} /* for */
out_strm << "</metadata>" << endl << endl;

```

1281.

```
< Dublin_Core_Metadata_Type :: output definition 1279 > +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Exiting 'Dublin_Core_Metadata_Type::output' successfully"
            << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; } /* End of Dublin_Core_Metadata_Type :: output definition */
```

1282. Parse. [LDF 2012.12.20.]

Log

[LDF 2012.12.20.] Added this function.

```
< Dublin_Core_Metadata_Type function declarations 1266 > +≡
int parse(char *buffer);
```

1283.

```
< Dublin_Core_Metadata_Type :: parse definition 1283 > ≡
int Dublin_Core_Metadata_Type :: parse(char *buffer){ bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    stringstream temp_strm;
    int status;
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Entering 'Dublin_Core_Metadata_Type::parse' ."
            << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

See also sections 1284 and 1285.
This code is used in section 1333.

1284.

```

⟨ Dublin_Core_Metadata_Type::parse definition 1283 ⟩ +≡
XML_Parserp = XML_ParserCreate(0);
if (p == 0) {
    cerr << "ERROR! In 'Dublin_Core_Metadata_Type::parse':"
    endl << "Couldn't allocate memory for parser." << endl <<
    "Exiting function unsuccessfully with return value 1." << endl;
    return 1;
} /* if (p == 0) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'Dublin_Core_Metadata_Type::parse':"
        endl << "Allocated memory for parser successfully."
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
XML_SetElementHandler(p, start, end);
XML_SetCharacterDataHandler(p, handle_data);
XML_SetUserData(p, static_cast<void*>(this));
status = XML_Parse(p, buffer, strlen(buffer), true);
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "'XML_Parse' returned 'status' == "
        endl << "'XML_Parse' failed, returning 0."
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    if (status == 0) {
        cerr << "ERROR! In 'Dublin_Core_Metadata_Type::parse':"
        endl << "'XML_Parse' failed, returning 0."
        endl << "Exiting function unsuccessfully with return value 1." << endl;
        XML_ParserFree(p);
        return 1;
    }
    else if (status == XML_STATUS_ERROR) {
        lock_cerr_mutex();
        fprintf(stderr, "Parse error at line %" XML_FMT_INT_MOD "u:\n%s\n",
            XML_GetCurrentLineNumber(p), XML_ErrorString(XML_GetErrorCode(p)));
        unlock_cerr_mutex();
        XML_ParserFree(p);
        return 1;
    }
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'Dublin_Core_Metadata_Type::parse':"
        endl << "'XML_Parse' succeeded, returning "
        << status << "." << endl;
        show("*this:");
        unlock_cerr_mutex();
    }

```

```
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1285.

```
< Dublin_Core_Metadata_Type :: parse definition 1283 > +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Exiting 'Dublin_Core_Metadata_Type::parse' successfully"
            << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    XML_ParserFree(p);
    return 0; } /* End of Dublin_Core_Metadata_Type :: parse definition */
```

1286. Start handler. [LDF 2012.12.20.]

Log

[LDF 2012.12.20.] Added this function.

```
< Dublin_Core_Metadata_Type function declarations 1266 > +≡
static void XMLCALL start(void *data, const char *el, const char **attr);
```

1287.

```

⟨ Dublin_Core_Metadata_Type :: start definition 1287 ⟩ ≡
void XMLCALL Dublin_Core_Metadata_Type :: start( void *data, const char *el, const char **attr) {
    bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    static string item;
    static string element;
    static string qualifier;
    static size_t pos;
    static int i;
    static unsigned int element_ctr;
    static unsigned int qualifier_ctr;
    static map<string, unsigned int>::const_iterator iter;
    Dublin_Core_Metadata_Type *dcm = 0;
    static Dublin_Core_Metadata_Sub_Type dcm_sub;
    dcm = static_cast<Dublin_Core_Metadata_Type *>(data);
    dcm_sub.clear();
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Entering " 'Dublin_Core_Metadata_Type::start' . " " << endl << "Depth == " <<
        Depth << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Element: " << el << endl;
        for (i = 0; attr[i]; i += 2) {
            cerr << "Attribute: " << attr[i] << endl << "Value: " << attr[i + 1] << endl << endl;
        } /* for */
        cerr << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

See also sections 1288, 1289, and 1290.

This code is used in section 1333.

1288. Determine whether *item* is an element or a qualifier. [LDF 2012.12.20.]

```

⟨ Dublin_Core_Metadata_Type :: start definition 1287 ⟩ +≡
  element = qualifier = "";
  element_ctr = qualifier_ctr = 0;
  item = el;
  pos = item.find("dc:");
  if (pos ≠ string::npos) {
    element = item.substr(pos + strlen("dc:"));
  }
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "item is an element: " << item << endl << "element == " << element << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  iter = element_ctr_map.find(element);
  if (iter == element_ctr_map.end()) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "Element " << element << " not found in 'element_ctr_map'." << endl <<
      "Ignoring." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    element = "";
    element_ctr = 0;
    goto AFTER_TESTS;
  }
  else {
    element_ctr = iter->second;
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "element_ctr == " << element_ctr << endl << "element_map[" << element_ctr <<
      "] == " << element_map[element_ctr] << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    } /* else */
    goto AFTER_TESTS;
  } /* if */
  pos = item.find("dcterms:");
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "pos == " << pos << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  if (pos ≠ string::npos) {
    qualifier = item.substr(pos + strlen("dcterms:"));
  }

```

```

#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "item is a qualifier: " << item << endl << "qualifier" == " " << qualifier << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    iter = qualifier_ctr_map.find(qualifier);
    if (iter == qualifier_ctr_map.end()) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Qualifier " << qualifier << " not found in 'qualifier_ctr_map'." << endl <<
            "Ignoring." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        qualifier = "";
        qualifier_ctr = 0;
        goto AFTER_TESTS;
    }
    else {
        qualifier_ctr = iter->second;
#endif DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "qualifier_ctr == " << qualifier_ctr << endl << "qualifier_map[" << qualifier_ctr <<
                "] == " << qualifier_map[qualifier_ctr] << endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    } /* else */
    goto AFTER_TESTS;
} /* if */

```

1289. If namespace not specified, first test whether *item* is an element, then if it's a qualifier. [LDF 2012.12.20.] ■

```

⟨Dublin-Core-Metadata-Type::start definition 1287⟩ +≡
    iter = element_ctr_map.find(item);
    if (iter != element_ctr_map.end()) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "item == " << item << " found in 'element_ctr_map'." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        element = item;
        element_ctr = iter->second;
        goto AFTER_TESTS;
    } /* if */

```

1290.

```

⟨ Dublin_Core_Metadata_Type :: start definition 1287 ⟩ +≡
    iter = qualifier_ctr_map.find(item);
    if (iter != qualifier_ctr_map.end()) {
#if DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "item'"_u==_u" << item << " found in 'qualifier_ctr_map'." << endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        qualifier = item;
        qualifier_ctr = iter->second;
        goto AFTER_TESTS;
    } /* if */
AFTER_TESTS:
    if (element_ctr > 0) {
        dcm_sub.element_id = element_ctr;
    }
    else if (qualifier_ctr > 0) {
        dcm_sub.qualifier_id = qualifier_ctr;
    }
    if (element_ctr > 0 ∨ qualifier_ctr > 0) {
        for (i = 0; attr[i]; i += 2) {
            dcm_sub.attribute_map.insert(make_pair(attr[i], attr[i + 1]));
        }
    } /* if */
dcm_metadata_sub_stack.push(dcm_sub); /* Always push dcm_sub onto the stack so that the right
                                     object is processed in Dublin_Core_Metadata_Type::end. If element_id and qualifier_id are
                                     both 0, it will be ignored. [LDF 2012.12.20.] */
Depth++;
return; /* Dublin_Core_Metadata_Type::start */

```

1291. End handler. [LDF 2012.12.20.]

Log

[LDF 2012.12.20.] Added this function.

```

⟨ Dublin_Core_Metadata_Type function declarations 1266 ⟩ +≡
static void XMLCALLend(void *data, const char *el);

```

1292.

```

⟨ Dublin_Core_Metadata_Type :: end definition 1292 ⟩ ≡
void XMLCALL Dublin_Core_Metadata_Type :: end(void *data, const char *el){ bool DEBUG = false;
    /* true */
    set_debug_level(DEBUG, 0, 0);
    static Dublin_Core_Metadata_Type *dcm = 0;
    static Dublin_Core_Metadata_Sub_Type *dcm_sub = 0;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Entering 'Dublin_Core_Metadata_Type::end'." << endl << "Depth== " << Depth <<
            endl << "last_value.str() == " << last_value.str() << "," << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    dcm = static_cast<Dublin_Core_Metadata_Type *>(data);
    /* !! TODO: LDF 2012.12.20. Add error handling for case that the stack is empty. */
    dcm_sub = &dcm->metadata_sub_stack.top();
    if (dcm_sub->element_id > 0 ∨ dcm_sub->qualifier_id > 0) {
        dcm_sub->value = last_value.str();
        dcm->metadata_sub_map.insert(make_pair(0, *dcm_sub));
    }
#endif /* DEBUG_COMPILE */
    if (DEBUG) {
        lock_cerr_mutex();
        dcm_sub->show("dcm_sub:");
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    dcm->metadata_sub_stack.pop();
    last_value.str("");
    Depth--;
}

```

See also sections 1293 and 1294.

This code is used in section 1333.

1293. Currently, this code is never reached, because the character buffer passed to *XML_Parse* only ever contains a single complete XML “metadata” expression. That is, it starts and ends with a *<metadata>* tag and *</metadata>*, respectively.

If this might ever not be the case, I would need to throw (and catch) an exception, because *XML_Parse* doesn’t test the return value of this function. [LDF 2012.12.20.]

```

⟨ Dublin_Core_Metadata_Type :: end definition 1292 ⟩ +≡
if (Depth ≡ 0) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'end': Depth== 0. Quitting." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return;
} /* if (Depth ≡ 0) */

```

1294.

```
{ Dublin_Core_Metadata_Type::end definition 1292 } +≡  
    return; } /* Dublin_Core_Metadata_Type::end */
```

1295. Handle data. [LDF 2012.12.20.]

Log

[LDF 2012.12.20.] Added this function.

```
{ Dublin_Core_Metadata_Type function declarations 1266 } +≡  
static void handle_data(void *data, const char *content, int length);
```

1296.

```

⟨ Dublin_Core_Metadata_Type :: handle_data definition 1296 ⟩ ≡
void Dublin_Core_Metadata_Type :: handle_data(void *data, const char *content, int length)
{
    bool DEBUG = false;      /* true */
    set_debug_level(DEBUG, 0, 0);
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Entering 'handle_data'." << endl;
        unlock_cerr_mutex();
    }      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */
    static stringstream temp_strm;
    static char temp_cstr[2048];
    static bool space;
    static int i;
    static string temp_str;
    temp_strm.str("");
    temp_strm.clear();
    memset(temp_cstr, 0, 2048);
    temp_str = "";
    temp_strm.write(content, length);
    memcpy(temp_cstr, content, length);
    space = true;
    i = 0;
    for ( ; i < strlen(temp_cstr); ++i) {
        if (!isspace(temp_cstr[i])) {
            space = false;
#ifndef DEBUG_COMPILE
            if (DEBUG) {
                lock_cerr_mutex();
                cerr << "In 'Dublin_Core_Metadata_Type::handle_data':"
                    << endl <<
                    "Non-space character found: '" << i << "' == "
                    << temp_cstr[i] <<
                    ". Breaking." << endl;
                unlock_cerr_mutex();
            }      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */
            break;
        }
    }      /* for */
    if (space) {
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "Content is blank. Exiting 'handle_data'." << endl;
            unlock_cerr_mutex();
        }      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */
        return;
    }      /* if (space) */
}

```

```

#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "After loop i==" << i << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    temp_str = temp_cstr;
    if (last_value.str().length() == 0) temp_str.erase(0, i);
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'Dublin_Core_Metadata_Type::handle_data'" << endl << "temp_cstr==" <<
            temp_cstr << endl << "temp_str==" << temp_str << endl << "length==" << length <<
            endl << "content (between single quotes)==" << endl << ",";
        fwrite(content, 1, length, stderr);
        cerr << "," << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    last_value << temp_str;
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Exiting 'handle_data'." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return;
} /* End of Dublin_Core_Metadata_Type::handle_data definition */

```

This code is used in section 1333.

1297. Write to database. [LDF 2012.12.14.]

Log

[LDF 2012.12.14.] Added this function.

[LDF 2013.01.09.] Moved this function from class Scan_Parse_Parameter_Type to class Dublin_Core_Metadata_Type.

⟨ Dublin_Core_Metadata_Type function declarations 1266 ⟩ +≡

int write_to_database(MYSQL *mysql_ptr);

1298.

```

⟨ Dublin_Core_Metadata_Type :: write_to_database definition 1298 ⟩ ≡
int Dublin_Core_Metadata_Type :: write_to_database(MYSQL * mysql_ptr){ bool DEBUG = false;
    /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    stringstream temp_strm;
    MYSQL_RES * result = 0;
    unsigned int row_ctr;
    unsigned int field_ctr;
    long affected_rows;
    MYSQL_ROW curr_row;
    stringstream sql_strm;
    int ret_val = 0;
    bool attributes_found = false;
    string comma_str;
    string comma_str_1;
    stringstream sql_strm_1;
    unsigned long dc_metadata_id;
    unsigned long dc_metadata_sub_id;
    unsigned long save_dc_metadata_id;
    unsigned long save_dc_metadata_sub_id;
    unsigned long temp_val;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Entering 'Dublin_Core_Metadata_Type::write_to_database'." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

See also sections 1299, 1300, 1301, 1302, 1303, 1304, 1305, 1306, 1307, 1308, 1309, 1310, 1311, and 1312.

This code is used in section 1333.

1299.

```
< Dublin_Core_Metadata_Type::write_to_database definition 1298 > +≡
    status = mysql_select_db(mysql_ptr, "gwirldsif");
    if (status ≡ 0) {
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr ≪ "In ‘Dublin_Core_Metadata_Type::write_to_database’: “
                  “mysql_select_db” succeeded” .” ≪ endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    } /* if (status ≡ 0) */
else /* status ≠ 0 */
{
    lock_cerr_mutex();
    cerr ≪ "In ‘Dublin_Core_Metadata_Type::write_to_database’: “
          “mysql_select_db” failed, returning” ≪ status ≪ endl;
    unlock_cerr_mutex();
    return 1;
} /* else (status ≠ 0) */
```

1300.

```

⟨ Dublin_Core_Metadata_Type::write_to_database definition 1298 ⟩ +≡
  sql_strm << "lock_tables_Dublin_Core_Metadata_write,Dublin_Core_Metadata_Sub_write," <<
    "Dublin_Core_Attributes_write";
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "sql_strm.str() == " << sql_strm.str() << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  status = submit_mysql_query(sql_strm.str(), result, mysql_ptr);
  if (status != 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Dublin_Core_Metadata_Type::write_to_database':" << endl <<
      "'submit_mysql_query' failed, returning " << status << ":" << endl <<
      mysql_error(mysql_ptr) << endl << "Failed to lock database tables." << endl <<
      "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    return 1;
  } /* if (status != 0) */
#endif DEBUG_COMPILE
else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'Dublin_Core_Metadata_Type::write_to_database': Locked "
        "database tables successfully." << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
mysql_free_result(result);
result = 0;

```

1301.

```

⟨ Dublin_Core_Metadata_Type :: write_to_database definition 1298 ⟩ +≡
sql_strm.str("");
sql_strm << "select_(select_dublin_core_metadata_id_from_Dublin_Core_Metadata"
    "order_by_dublin_core_metadata_id_desc_limit_1),_"
    "(select_dublin_core_metadata_sub_id_from_Dublin_Core_Metadata_Sub"
    "order_by_dublin_core_metadata_sub_id_desc_limit_1)";
status = submit_mysql_query(sql_strm.str(), result, mysql_ptr, &row_ctr, &field_ctr);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Dublin_Core_Metadata_Type::write_to_database':"
        << endl <<
        "'Dublin_Core_Metadata_Type::submit_mysql_query' failed, returning"
        status << endl << mysql_error(mysql_ptr) << endl <<
        "Failed to retrieve 'Dublin_Core_Metadata.dublin_core_metadata_id'."
        endl << "Exiting function unsuccessfully with return value 1."
        endl;
    unlock_cerr_mutex();
    if (result) {
        mysql_free_result(result);
        result = 0;
    }
    ret_val = 1;
    goto UNLOCK_TABLES_AND_EXIT;
} /* if (status ≠ 0) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'Dublin_Core_Metadata_Type::write_to_database':"
            << endl <<
            "'Dublin_Core_Metadata_Type::submit_mysql_query' succeeded."
            << endl <<
            "'row_ctr' == " << row_ctr << endl <<
            "'field_ctr' == " << field_ctr << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1302.

```

⟨ Dublin_Core_Metadata_Type :: write_to_database definition 1298 ⟩ +≡
if (row_ctr ≡ 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Dublin_Core_Metadata_Type::write_to_database':"
        << endl <<
        "'Dublin_Core_Metadata_Type::submit_mysql_query' returned 0 rows."
        << endl <<
        "Failed to retrieve 'Dublin_Core_Metadata.dublin_core_metadata_id'."
        endl <<
        "Exiting function unsuccessfully with return value 1."
        endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    ret_val = 1;
    goto UNLOCK_TABLES_AND_EXIT;
} /* if (row_ctr ≡ 0) */

```

1303.

```
< Dublin_Core_Metadata_Type::write_to_database definition 1298 > +≡
  if ((curr_row = mysql_fetch_row(result)) == 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Dublin_Core_Metadata_Type::write_to_database':"
    << endl <<
    "'mysql_fetch_row' failed:" << endl << mysql_error(mysql_ptr) << endl <<
    "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    ret_val = 1;
    goto UNLOCK_TABLES_AND_EXIT;
  } /* if (curr_row = mysql_fetch_row(result) == 0) */
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "In 'Dublin_Core_Metadata_Type::write_to_database':"
      << endl <<
      "'mysql_fetch_row' succeeded." << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1304.

```

⟨ Dublin_Core_Metadata_Type :: write_to_database definition 1298 ⟩ +≡
  if (curr_row[0] == 0 ∨ strlen(curr_row[0]) == 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Dublin_Core_Metadata_Type::write_to_database':"
    endl << "'curr_row[0]' is NULL or empty." << endl <<
    "Failed to retrieve 'Dublin_Core_Metadata.dublin_core_metadata_id'." <<
    endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    ret_val = 1;
    goto UNLOCK_TABLES_AND_EXIT;
  } /* if (curr_row[0] == 0 ∨ strlen(curr_row[0]) == 0) */
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'Dublin_Core_Metadata_Type::write_to_database':"
    << "'curr_row[0]' == "
    << curr_row[0] << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  errno = 0;
  temp_val = strtoul(curr_row[0], 0, 10);
  if (temp_val == ULONG_MAX ∨ errno != 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Dublin_Core_Metadata_Type::write_to_database':"
    << endl <<
    "'strtoul' failed, returning"
    << temp_val;
    if (temp_val == ULONG_MAX) cerr << "(ULONG_MAX)." << endl;
    else cerr << ". " << endl;
    if (errno != 0) cerr << "strtoul error: "
    << strerror(errno) << endl;
    cerr << "Failed to retrieve 'Dublin_Core_Metadata.dublin_core_metadata_id'." << endl <<
    "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    ret_val = 1;
    goto UNLOCK_TABLES_AND_EXIT;
  } /* if */
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "temp_val == "
      << temp_val << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  dc_metadata_id = temp_val + 1;
  save_dc_metadata_id = dc_metadata_id;

```

1305.

```

⟨ Dublin_Core_Metadata_Type :: write_to_database definition 1298 ⟩ +≡
  if (curr_row[1] == 0 ∨ strlen(curr_row[1]) == 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Dublin_Core_Metadata_Type::write_to_database':"
    endl << "'curr_row[1]' is NULL or empty." << endl <<
    "Failed to retrieve 'Dublin_Core_Metadata_Sub.dublin_core_metadata_sub_id'." <<
    endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    ret_val = 1;
    goto UNLOCK_TABLES_AND_EXIT;
  } /* if (curr_row[1] == 0 ∨ strlen(curr_row[1]) == 0) */
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'Dublin_Core_Metadata_Type::write_to_database':"
    endl << "'curr_row[1]' == "
    curr_row[1] << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  errno = 0;
  temp_val = strtoul(curr_row[1], 0, 10);
  if (temp_val == ULONG_MAX ∨ errno != 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Dublin_Core_Metadata_Type::write_to_database':"
    endl <<
    "'strtoul' failed, returning" << temp_val;
    if (temp_val == ULONG_MAX) cerr << "(ULONG_MAX)." << endl;
    else cerr << "." << endl;
    if (errno != 0) cerr << "strtoul error:" << strerror(errno) << endl;
    cerr << "Failed to retrieve 'Dublin_Core_Metadata_Sub.dublin_core_metadata_sub_id'." <<
    endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    ret_val = 1;
    goto UNLOCK_TABLES_AND_EXIT;
  } /* if */
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "temp_val == "
      temp_val << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  dc_metadata_sub_id = temp_val + 1;
  save_dc_metadata_sub_id = dc_metadata_sub_id;
  mysql_free_result(result);
  result = 0;

```

1306.

Log

[LDF 2013.02.28.] Now setting *id* = *dc_metadata_id*.

```

⟨ Dublin_Core_Metadata_Type::write_to_database definition 1298 ⟩ +≡
    sql_strm.str("");
    id = dc_metadata_id;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        show(*this|(Dublin_Core_Metadata_Type):");
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    sql_strm << "insert|into|Dublin_Core_Metadata|(dublin_core_metadata_id,|user_id,|"
    "irods_server_id,|irods_object_path,|deleted,|created,|last_modified)|"
    "values(|" << dc_metadata_id << ",|" << user_id << ",|1,|" << "'|" << irods_object_path <<
    "|',|false,|now(),|0)";
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "sql_strm.str()|==|" << sql_strm.str() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    status = submit_mysql_query(sql_strm.str(), result, mysql_ptr, 0, 0, &affected_rows);
    if (status != 0) {
        lock_cerr_mutex();
        cerr << "ERROR!|In|‘Dublin_Core_Metadata_Type::write_to_database’:|" << endl <<
            “Dublin_Core_Metadata_Type::submit_mysql_query’|failed,|returning|” <<
            status << ".|" << endl << mysql_error(mysql_ptr) << endl <<
            “Failed|to|insert|data|into|‘Dublin_Core_Metadata’|table.” << endl <<
            “Exiting|function|unsuccessfully|with|return|value|1.” << endl;
        unlock_cerr_mutex();
        if (result) {
            mysql_free_result(result);
        }
        ret_val = 1;
        goto DELETE_DATABASE_ENTRIES_AND_EXIT;
    } /* if (status != 0) */
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "In|‘Dublin_Core_Metadata_Type::write_to_database’:|" << endl <<
                “Dublin_Core_Metadata_Type::submit_mysql_query’|succeeded.” << endl <<
                “affected_rows’|==|” << affected_rows << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    mysql_free_result(result);

```

result = 0;

1307.

Log

[LDF 2013.02.28.] Now setting **Dublin_Core_Metadata_Sub_Type**::*metadata_id* = *dc_metadata_id* and **Dublin_Core_Metadata_Sub_Type**::*id* = *dc_metadata_sub_id* in all members of *metadata_sub_map*.

```

⟨ Dublin_Core_Metadata_Type :: write_to_database definition 1298 ⟩ +≡
    sql_strm.str("");
    sql_strm << "insert into Dublin_Core_Metadata_Sub (dublin_core_metadata_sub_id, "
    "dublin_core_metadata_id, dublin_core_element_id, dublin_core_term_id, "
    "value) values ";
    sql_strm_1 << comma_str_1 << "insert into Dublin_Core_Attributes" <<
        "(dublin_core_metadata_id, dublin_core_metadata_sub_id, "
        "attribute, value) values ";
    attributes_found = false;
    for (multimap<unsigned int, Dublin_Core_Metadata_Sub_Type>::iterator
        iter = metadata_sub_map.begin(); iter != metadata_sub_map.end(); ++iter) {
        sql_strm << comma_str << "(" << dc_metadata_sub_id << ", "
        << dc_metadata_id << ", "
        << iter->second.element_id << ", "
        << iter->second.qualifier_id << ", "
        << "\"" << iter->second.value <<
        ")";
        comma_str = ", ";
        comma_str_1 = "";
        for (multimap<string, string>::const_iterator iter_1 = iter->second.attribute_map.begin();
            iter_1 != iter->second.attribute_map.end(); ++iter_1) {
            attributes_found = true;
            sql_strm_1 << comma_str_1 << "(" << dc_metadata_id << ", "
            << dc_metadata_sub_id << ", "
            << ", "
            << iter_1->first << ", "
            << "\"" << iter_1->second << ")";
            comma_str_1 = ", ";
        } /* inner for */
        iter->second.metadata_id = dc_metadata_id;
        iter->second.id = dc_metadata_sub_id;
        ++dc_metadata_sub_id;
    } /* outer for */
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "sql_strm.str() == " << sql_strm.str() << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
status = submit_mysql_query(sql_strm.str(), result, mysql_ptr, 0, 0, &affected_rows);
if (status != 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Dublin_Core_Metadata_Type::write_to_database':"
    << endl <<
        "'Dublin_Core_Metadata_Type::submit_mysql_query' failed, returning"
    << status << "."
    << endl << mysql_error(mysql_ptr) << endl <<
        "Failed to insert data into 'Dublin_Core_Metadata_Sub' table."
    << endl <<
        "Exiting function unsuccessfully with return value 1."
    << endl;
    unlock_cerr_mutex();
    if (result) {
        mysql_free_result(result);
    }
}

```

```
ret_val = 1;
goto DELETE_DATABASE_ENTRIES_AND_EXIT;
} /* if (status != 0) */
#endif DEBUG_COMPILE
else
if (DEBUG) {
lock_cerr_mutex();
cerr << "In 'Dublin_Core_Metadata_Type::write_to_database':" << endl <<
" 'Dublin_Core_Metadata_Type::submit_mysql_query' succeeded." << endl <<
" 'affected_rows'" <== " " << affected_rows << endl;
unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
mysql_free_result(result);
result = 0;
sqlLstrm.str("");
```

1308.

```

⟨ Dublin_Core_Metadata_Type::write_to_database definition 1298 ⟩ +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "sql_strm_1.str() == " << sql_strm_1.str() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
status = submit_mysql_query(sql_strm_1.str(), result, mysql_ptr, 0, 0, &affected_rows);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Dublin_Core_Metadata_Type::write_to_database':"
    << endl << "'Dublin_Core_Metadata_Type::submit_mysql_query' failed, returning"
    << status << "."
    << endl << mysql_error(mysql_ptr) << endl <<
    "Failed to insert data into 'Dublin_Core_Metadata_Sub' table."
    << endl <<
    "Exiting function unsuccessfully with return value 1."
    << endl;
    unlock_cerr_mutex();
    if (result) {
        mysql_free_result(result);
    }
    ret_val = 1;
    goto DELETE_DATABASE_ENTRIES_AND_EXIT;
} /* if (status ≠ 0) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'Dublin_Core_Metadata_Type::write_to_database':"
        << endl << "'Dublin_Core_Metadata_Type::submit_mysql_query' succeeded."
        << endl <<
        "affected_rows == "
        << affected_rows << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
mysql_free_result(result);
result = 0;
sql_strm.str("");
sql_strm_1.str("");

```

1309.

```

⟨ Dublin_Core_Metadata_Type :: write_to_database definition 1298 ⟩ +≡
  if (ret_val == 0) goto UNLOCK_TABLES_AND_EXIT;
DELETE_DATABASE_ENTRIES_AND_EXIT:
  if (result) {
    mysql_free_result(result);
    result = 0;
  }
  sql_strm.str("");
  sql_strm_1.str("");
  sql_strm << "delete from Dublin_Core_Metadata where dublin_core_metadata_id >= "
  << save_dc_metadata_id;
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "sql_strm.str() == " << sql_strm.str() << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  status = submit_mysql_query(sql_strm.str(), result, mysql_ptr, 0, 0, &affected_rows);
  if (status != 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Dublin_Core_Metadata_Type::write_to_database':"
    << endl <<
      "'Dublin_Core_Metadata_Type::submit_mysql_query' failed, returning "
    << status << "."
    << endl << mysql_error(mysql_ptr) << endl <<
      "Failed to delete data from 'Dublin_Core_Metadata' table."
    << endl;
    unlock_cerr_mutex();
    ret_val = 1;
  } /* if (status != 0) */
#endif DEBUG_COMPILE
  else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'Dublin_Core_Metadata_Type::write_to_database':"
    << endl <<
      "'Dublin_Core_Metadata_Type::submit_mysql_query' succeeded."
    << endl <<
      "'affected_rows' == "
    << affected_rows << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  mysql_free_result(result);
  result = 0;
  sql_strm.str("");
  sql_strm << "delete from Dublin_Core_Metadata_Sub where dublin_core_metadata_sub_id >= "
  << save_dc_metadata_sub_id;
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "sql_strm.str() == " << sql_strm.str() << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  status = submit_mysql_query(sql_strm.str(), result, mysql_ptr, 0, 0, &affected_rows);

```

```

if (status != 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Dublin_Core_Metadata_Type::write_to_database':" << endl <<
        "'Dublin_Core_Metadata_Type::submit_mysql_query' failed, returning" <<
        status << "." << endl << mysql_error(mysql_ptr) << endl <<
        "Failed to delete data from 'Dublin_Core_Metadata_Sub' table." << endl;
    unlock_cerr_mutex();
    ret_val = 1;
} /* if (status != 0) */
#endif /* DEBUG_COMPILE */
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'Dublin_Core_Metadata_Type::write_to_database':" << endl <<
        "'Dublin_Core_Metadata_Type::submit_mysql_query' succeeded." << endl <<
        "'affected_rows'" <= "Affected_rows" << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
mysql_free_result(result);
result = 0;
sql_strm.str("");
sql_strm.str("");
sql_strm << "delete from Dublin_Core_Attributes where dublin_core_metadata_id >= " <<
    save_dc_metadata_id << " or dublin_core_metadata_sub_id >= " << save_dc_metadata_sub_id;
#endif /* DEBUG_COMPILE */
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "sql_strm.str() == " << sql_strm.str() << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
status = submit_mysql_query(sql_strm.str(), result, mysql_ptr, 0, 0, &affected_rows);
if (status != 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Dublin_Core_Metadata_Type::write_to_database':" << endl <<
        "'Dublin_Core_Metadata_Type::submit_mysql_query' failed, returning" <<
        status << "." << endl << mysql_error(mysql_ptr) << endl <<
        "Failed to delete data from 'Dublin_Core_Attributes' table." << endl;
    unlock_cerr_mutex();
    ret_val = 1;
} /* if (status != 0) */
#endif /* DEBUG_COMPILE */
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'Dublin_Core_Metadata_Type::write_to_database':" << endl <<
        "'Dublin_Core_Metadata_Type::submit_mysql_query' succeeded." << endl <<
        "'affected_rows'" <= "Affected_rows" << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

```
mysql_free_result(result);
result = 0;
sql_strm.str("");
```

1310.

```
< Dublin_Core_Metadata_Type::write_to_database definition 1298 > +≡
UNLOCK_TABLES_AND_EXIT:
if (result) {
    mysql_free_result(result);
    result = 0;
}
status = submit_mysql_query("unlock_tables", result, mysql_ptr, 0, 0, 0);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ "ERROR! In 'Dublin_Core_Metadata_Type::write_to_database':"
    endl ≪ "'submit_mysql_query' failed, returning" ≪ status ≪
        "." ≪ endl ≪ "Failed to unlock database tables." ≪ endl ≪
        "Exiting function unsuccessfully with return value 1." ≪ endl;
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    return 1;
} /* if (status ≠ 0) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ "In 'Dublin_Core_Metadata_Type::write_to_database':"
        endl ≪ "'submit_mysql_query' succeeded, returning 0." ≪ endl ≪
            "Unlocked database tables successfully." ≪ endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
mysql_free_result(result);
result = 0;
```

1311.

```
< Dublin_Core_Metadata_Type::write_to_database definition 1298 > +≡
if (ret_val ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ "In 'Dublin_Core_Metadata_Type::write_to_database':"
    endl ≪ "'ret_val' == "
    ret_val ≪ " != 0 ." ≪ endl ≪ "Exiting function unsuccessfully with return value "
    ret_val ≪ "." ≪ endl;
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    return ret_val;
} /* if (ret_val ≠ 0) */
```

1312.**Log**

[LDF 2013.02.28.] Now setting `*dc_metadata_id_ptr = dc_metadata_id` if `dc_metadata_id_ptr` is non-null.

```
<Dublin_Core_Metadata_Type::write_to_database definition 1298> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Exiting 'Dublin_Core_Metadata_Type::write_to_database'" <<
            "successfully with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; } /* End of Dublin_Core_Metadata_Type::write_to_database definition */
```

1313. Set handle ID (`set_handle_id`). [LDF 2013.02.28.]**Log**

[LDF 2013.02.28.] Added this function.

```
<Dublin_Core_Metadata_Type function declarations 1266> +≡
int set_handle_id(MYSQL *mysql_ptr, unsigned long int hhandle_id);
```

1314.

```
<Dublin_Core_Metadata_Type::set_handle_id definition 1314> ≡
int Dublin_Core_Metadata_Type::set_handle_id(MYSQL *mysql_ptr, unsigned long int hhandle_id){
    bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    MYSQL_RES *result = 0;
    unsigned int row_ctr;
    unsigned int field_ctr;
    long affected_rows;
    MYSQL_ROW curr_row;
    stringstream sql_strm;
    int ret_val = 0;
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Entering 'Dublin_Core_Metadata_Type::set_handle_id'." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

See also sections 1315, 1316, 1317, and 1318.

This code is used in section 1333.

1315. Lock `gwirdsif.Dublin_Core_Metadata` database table. [LDF 2013.02.28.]

```
< Dublin_Core_Metadata_Type::set_handle_id definition 1314 > +≡
  sql_strm << "lock_table_gwirdsif.Dublin_Core_Metadata_write";
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "sql_strm.str() == " << sql_strm.str() << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  status = submit_mysql_query(sql_strm.str(), result, mysql_ptr);
  if (status != 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Dublin_Core_Metadata_Type::set_handle_id':"
        << endl <<
        "'submit_mysql_query' failed, returning " << status << ":" << endl <<
        mysql_error(mysql_ptr) << endl << "Failed to lock database tables." << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    return 1;
  } /* if (status != 0) */
#endif DEBUG_COMPILE
else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'Dublin_Core_Metadata_Type::set_handle_id': Locked"
        << "database tables successfully." << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  sql_strm.str("");
  mysql_free_result(result);
  result = 0;
```

1316.

```
< Dublin_Core_Metadata_Type :: set_handle_id definition 1314 > +≡
  handle_id = hhandle_id;
  sql_strm ≪ "update_gwirdsif.Dublin_Core_Metadata.set_handle_id=" ≪ handle_id ≪
    " where_dublin_core_metadata_id=" ≪ id;
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "sql_strm.str()=" ≪ sql_strm.str() ≪ endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  status = submit_mysql_query(sql_strm.str(), result, mysql_ptr, 0, 0, &affected_rows);
  if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ "ERROR! In 'Dublin_Core_Metadata_Type::set_handle_id':" ≪ endl ≪
      "'submit_mysql_query' failed, returning" ≪ status ≪ ":" ≪ endl ≪
      mysql_error(mysql_ptr) ≪ endl ≪ "Failed to update 'Dublin_Core_Metadata' table." ≪
      endl;
    unlock_cerr_mutex();
    if (result) {
      mysql_free_result(result);
      result = 0;
    }
    ret_val = 1;
    goto SET_HANDLE_ID_UNLOCK_TABLES;
  } /* if (status ≠ 0) */
#endif DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr ≪ "In 'Dublin_Core_Metadata_Type::set_handle_id': Updated_Dub\
          lin_Core_Metadata" ≪ "table successfully." ≪ endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  sql_strm.str("");
  mysql_free_result(result);
  result = 0;
```

1317. Unlock gwrdsif.Dublin_Core_Metadata database table. [LDF 2013.02.28.]

```

⟨ Dublin_Core_Metadata_Type::set_handle_id definition 1314 ⟩ +≡
SET_HANDLE_ID_UNLOCK_TABLES: sql_strm.str("");
if (result) {
    mysql_free_result(result);
    result = 0;
}
sql_strm << "unlock_tables";
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "sql_strm.str() == " << sql_strm.str() << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
status = submit_mysql_query(sql_strm.str(), result, mysql_ptr);
if (status != 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'Dublin_Core_Metadata_Type::set_handle_id':"
        << endl <<
        "'submit_mysql_query' failed, returning " << status << ":" << endl <<
        mysql_error(mysql_ptr) << endl << "Failed to unlock_database_tables." << endl <<
        "Exiting function unsuccessfully with return_value 1." << endl;
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    return 1;
} /* if (status != 0) */
#ifndef DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'Dublin_Core_Metadata_Type::set_handle_id': Unlocked "
        << "database_tables successfully." << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
sql_strm.str("");
mysql_free_result(result);
result = 0;

```

1318.

```

⟨ Dublin_Core_Metadata_Type::set_handle_id definition 1314 ⟩ +≡
  if (ret_val ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ "Exiting ‘Dublin_Core_Metadata_Type::set_handle_id’ unsuccessfully" ≪
      "with return value" ≪ ret_val ≪ "." ≪ endl;
    unlock_cerr_mutex();
    return ret_val;
  } /* if (ret_val ≠ 0) */
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr ≪ "Exiting ‘Dublin_Core_Metadata_Type::set_handle_id’ successfully" ≪
        "with return value 0." ≪ endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  return ret_val; } /* End of Dublin_Core_Metadata_Type::set_handle_id definition */

```

1319. class Dublin_Core_Metadata_Sub_Type. [LDF 2012.12.06.]**Log**

[LDF 2012.12.06.] Added this type declaration.

[LDF 2012.12.12.] Renamed this **class**. Old name: *Dublin_Core_Qualifier_Type*. New name: **Dublin_Core_Metadata_Sub_Type**.

```

⟨ class Dublin_Core_Metadata_Sub_Type declaration 1319 ⟩ ≡
class Dublin_Core_Metadata_Sub_Type {
  friend int yyparse(yyscan_t parameter);
  friend class Scan_Parse_Parameter_Type;
  friend class Dublin_Core_Metadata_Type;
  friend int exchange_data_with_server(Scan_Parse_Parameter_Type &);
  friend int exchange_data_with_client(Scan_Parse_Parameter_Type &);

private: unsigned int id;
  unsigned long metadata_id;
  unsigned int element_id;
  unsigned int qualifier_id;
  string value;
  multimap<string, string> attribute_map;

public: ⟨ Dublin_Core_Metadata_Sub_Type function declarations 1320 ⟩
};

```

This code is used in sections 1333 and 1335.

1320. Default constructor. [LDF 2012.12.06.]

```

⟨ Dublin_Core_Metadata_Sub_Type function declarations 1320 ⟩ ≡
  Dublin_Core_Metadata_Sub_Type(void);

```

See also sections 1322, 1324, 1326, 1328, and 1330.

This code is used in section 1319.

1321.

```
< Dublin_Core_Metadata_Sub_Type function definitions 1321 > ≡
  Dublin_Core_Metadata_Sub_Type::Dublin_Core_Metadata_Sub_Type(void)
  {
    id = metadata_id = element_id = qualifier_id = 0;
    return;
  }
```

See also sections 1323, 1325, 1327, 1329, and 1331.

This code is used in section 1333.

1322. Destructor. [LDF 2012.12.06.]

Log

[LDF 2012.12.06.] Added this function.

```
< Dublin_Core_Metadata_Sub_Type function declarations 1320 > +≡
  ~Dublin_Core_Metadata_Sub_Type(void);
```

1323.

```
< Dublin_Core_Metadata_Sub_Type function definitions 1321 > +≡
  Dublin_Core_Metadata_Sub_Type::~Dublin_Core_Metadata_Sub_Type(void)
  {
    return;
  }
```

1324. Equality operator. [LDF 2012.12.21.]

Log

[LDF 2012.12.21.] Added this function.

```
< Dublin_Core_Metadata_Sub_Type function declarations 1320 > +≡
  bool operator==(const Dublin_Core_Metadata_Sub_Type &d) const;
```

1325.

```

⟨ Dublin_Core_Metadata_Sub_Type function definitions 1321 ⟩ +≡
bool Dublin_Core_Metadata_Sub_Type::operator==(const Dublin_Core_Metadata_Sub_Type
                                              &d) const
{
    bool DEBUG = false;      /* true */
    set_debug_level(DEBUG, 0, 0);
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Entering 'Dublin_Core_Metadata_Sub_Type::operator=='." << endl <<
            "element_id==" << element_id << endl << "d.element_id==" << d.element_id <<
            endl << "qualifier_id==" << qualifier_id << endl << "d.qualifier_id==" <<
            d.qualifier_id << endl << "value==" << value << endl << "d.value==" <<
            d.value << endl << "attribute_map.size() == " << attribute_map.size() << endl <<
            "d.attribute_map.size() == " << d.attribute_map.size() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    if (!(element_id == d.element_id & qualifier_id == d.qualifier_id & value == d.value & attribute_map.size() == d.attribute_map.size())) return false;
    if (attribute_map.size() == 0) return true;
    multimap<string, string>::const_iterator iter = attribute_map.begin();
    multimap<string, string>::const_iterator iter_1 = d.attribute_map.begin();
    for ( ; iter != attribute_map.end() & iter_1 != d.attribute_map.end(); ++iter, ++iter_1) {
        if (!(iter->first == iter_1->first & iter->second == iter_1->second)) return false;
    }
    return true;
} /* End of Dublin_Core_Metadata_Sub_Type::operator== definition */

```

1326. Inequality operator. [LDF 2012.12.21.]

Log

[LDF 2012.12.21.] Added this function.

```

⟨ Dublin_Core_Metadata_Sub_Type function declarations 1320 ⟩ +≡
bool operator≠(const Dublin_Core_Metadata_Sub_Type &d) const;

```

1327.

```
⟨ Dublin_Core_Metadata_Sub_Type function definitions 1321 ⟩ +≡
  bool Dublin_Core_Metadata_Sub_Type::operator≠(const Dublin_Core_Metadata_Sub_Type
                                                &d) const
  {
    bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
#ifndef DEBUG_COMPILE
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "Entering " ‘Dublin_Core_Metadata_Sub_Type::operator!=’ . “ << endl;
      unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return !operator==(d);
  }
```

1328. Clear. [LDF 2012.12.06.]

Log

[LDF 2012.12.06.] Added this function.

```
⟨ Dublin_Core_Metadata_Sub_Type function declarations 1320 ⟩ +≡
  void clear(void);
```

1329.

```
⟨ Dublin_Core_Metadata_Sub_Type function definitions 1321 ⟩ +≡
  void Dublin_Core_Metadata_Sub_Type::clear(void)
  {
    id = metadata_id = element_id = qualifier_id = 0;
    attribute_map.clear();
  } /* Dublin_Core_Metadata_Sub_Type::clear */
```

1330. Show. [LDF 2012.12.06.]

Log

[LDF 2012.12.06.] Added this function.

```
⟨ Dublin_Core_Metadata_Sub_Type function declarations 1320 ⟩ +≡
  int show(string s = "") const;
```

1331.

```

⟨ Dublin_Core_Metadata_Sub_Type function definitions 1321 ⟩ +≡
int Dublin_Core_Metadata_Sub_Type::show(string s) const
{
    if (s.size() == 0) s = "Dublin_Core_Metadata_Sub_Type:";

    cerr << s << endl << "id==uuuuuuuuuu" << id << endl << "metadata_id==u" << metadata_id << endl;
    if (element_id == 0 & qualifer_id == 0) cerr << "element_id==uqualifier_id==u0" << endl;
    else if (element_id > 0) cerr << "element_id==uuu" << element_id << "(" <<
        Dublin_Core_Metadata_Type::element_map[element_id] << ")";
    else cerr << "qualifier_id==uuuuu" << qualifer_id << "(" <<
        Dublin_Core_Metadata_Type::qualifier_map[qualifer_id] << ")" << endl;
    cerr << "value==uuuuuuuu" << value << endl;
    if (attribute_map.size() == 0) cerr << "attribute_map_is_empty." << endl;
    else {
        cerr << "attribute_map:" << endl;
        for (multimap<string,
            string>::const_iterator iter = attribute_map.begin(); iter != attribute_map.end(); ++iter) {
            cerr << iter->first << ":" << iter->second << endl;
        }
    }
    cerr << endl;
    return 0;
} /* Dublin_Core_Metadata_Sub_Type::show */

```

1332. Putting dcmttpp.web together.

1333. This is what's compiled.

```

⟨ Include files 3 ⟩
using namespace std;
using namespace gwrdifpk;
⟨ Preprocessor macro definitions 1262 ⟩
⟨ File-local variables 1263 ⟩
class Scan_Parse_Parameter_Type;
class Dublin_Core_Metadata_Type;
⟨ class Dublin_Core_Metadata_Sub_Type declaration 1319 ⟩
⟨ class Dublin_Core_Metadata_Type declaration 1264 ⟩
⟨ Initialize static Dublin_Core_Metadata_Type data members 1265 ⟩
⟨ Dublin_Core_Metadata_Sub_Type function definitions 1321 ⟩
⟨ Dublin_Core_Metadata_Type constructor definition 1267 ⟩
⟨ Dublin_Core_Metadata_Type destructor definition 1269 ⟩
⟨ Dublin_Core_Metadata_Type::clear definition 1271 ⟩
⟨ Dublin_Core_Metadata_Type::operator≡ definition 1273 ⟩
⟨ Dublin_Core_Metadata_Type::show definition 1275 ⟩
⟨ Dublin_Core_Metadata_Type::initialize_maps definition 1277 ⟩
⟨ Dublin_Core_Metadata_Type::output definition 1279 ⟩
⟨ Dublin_Core_Metadata_Type::parse definition 1283 ⟩
⟨ Dublin_Core_Metadata_Type::start definition 1287 ⟩
⟨ Dublin_Core_Metadata_Type::end definition 1292 ⟩
⟨ Dublin_Core_Metadata_Type::handle_data definition 1296 ⟩
⟨ Dublin_Core_Metadata_Type::write_to_database definition 1298 ⟩
⟨ Dublin_Core_Metadata_Type::set_handle_id definition 1314 ⟩

```

1334. This is what's written to the header file dcmtdtpp.h.

1335.

```
⟨ dcmtdtpp.h 1335 ⟩ ≡
#ifndef DCMTDTTP_H
#define DCMTDTTP_H 1
using namespace std;
using namespace gwrdifpk;
class Scan_Parse_Parameter_Type;
class Dublin_Core_Metadata_Type;
⟨ class Dublin_Core_Metadata_Sub_Type declaration 1319 ⟩
⟨ class Dublin_Core_Metadata_Type declaration 1264 ⟩
#endif
```

1336. class Distinguished_Name_Type (dstngnmt.web).

1337. Include files.

```
<Include files 3> +≡
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <gcrypt.h> /* for gcry_control */
#include <gnutls/gnutls.h>
#include <iomanip>
#include <iostream>
#include <map>
#include <fstream>
#include <sstream>
#include <string>
#include <vector>
#include <deque>
#include <stack>
#include <set>
#include <pthread.h>
#include <expat.h>
#if HAVE_CONFIG_H
#include <config.h>
#endif
#include <mysql.h>
#ifndef _XOPEN_SOURCE
#define _XOPEN_SOURCE
#endif
#include "glblcnst.h++"
#include "glblvrbl.h++"
#include "excptntp.h++"
#include "utilfncs.h++"
#include "parser.h++"
#include "scanner.h++"
#include "hdlvltp.h++"
#include "rspnstp.h++"
#include "irdsavtp.h++"
#include "irdsobtp.h++"
#include "hdltype.h++"
#include "dcmtddtp.h++"
#include "x509cert.h++"
```

1338. class Distinguished_Name_Type declaration. [LDF 2010.04.21.]

Log

[LDF 2010.04.21.] Added this **struct** declaration.

[LDF 2012.02.10.] Changed **Distinguished_Name_Type** from a **struct** to a **class**. Added **friend** declarations. All data members are **private**. The constructors are **public** and all other functions are **private**. !! TODO: Add functions for accessing data members and see if I can get rid of some of the **friend** declarations.

[LDF 2012.03.28.] Added **friend** declaration for **int client_connect_auth(void)**.

```
< class Distinguished_Name_Type declaration 1338 > ≡
class Distinguished_Name_Type {
    string organization;
    string organizationalUnitName;
    string commonName;
    string countryName;
    string stateOrProvinceName;
    string localityName;
    string user_name;
    int user_id;
    friend class Scan_Parse_Parameter_Type;
public: < Distinguished_Name_Type constructor declarations 1341 >
    < Distinguished_Name_Type function declarations 1345 >
};
```

This code is used in sections 1366 and 1367.

1339. Distinguished_Name_Type functions. [LDF 2010.04.21.]

Log

[LDF 2010.04.21.] Added this section.

1340. Constructors and Setting Functions. [LDF 2010.04.21.]

Log

[LDF 2010.04.21.] Added this section.

1341. Default constructor. [LDF 2010.04.21.]

Log

[LDF 2010.04.21.] Added this function.

```
< Distinguished_Name_Type constructor declarations 1341 > ≡
Distinguished_Name_Type(void);
```

See also section 1343.

This code is used in section 1338.

1342.

```
< Distinguished_Name_Type function definitions 1342 > ≡
Distinguished_Name_Type::Distinguished_Name_Type(void)
{
    user_id = -1;
    user_name = "";
    return;
} /* End of Distinguished_Name_Type default constructor definition */
```

See also sections 1344, 1346, 1347, 1348, 1349, 1361, 1363, and 1365.

This code is used in section 1366.

1343. Constructor with arguments. [LDF 2010.04.21.]

Log

[LDF 2010.04.21.] Added this function.

```
< Distinguished_Name_Type constructor declarations 1341 > +≡
Distinguished_Name_Type(string oorganization, string organizationalUnitName = "", string
    ccommonName = "", string ccountryName = "", string llocalityName = "", string
    sstateOrProvinceName = "", unsigned int uuser_id = 0, string uuser_name = "");
```

1344.

```
< Distinguished_Name_Type function definitions 1342 > +≡
Distinguished_Name_Type::Distinguished_Name_Type(string oorganization, string
    organizationalUnitName, string ccommonName, string ccountryName, string
    llocalityName, string sstateOrProvinceName, unsigned int uuser_id, string uuser_name)
: organization(oorganization), organizationalUnitName(organizationalUnitName),
    commonName(ccommonName), countryName(ccountryName), localityName(llocalityName),
    stateOrProvinceName(sstateOrProvinceName), user_id(uuser_id), user_name(uuser_name) {
return;
} /* End of Distinguished_Name_Type non-default constructor definition */
```

1345. Setting function with arguments. [LDF 2010.04.21.]

Log

[LDF 2010.04.21.] Added this function.

[LDF 2010.05.25.] BUG FIX: Added code for handling angle braces used as delimiters.

```
< Distinguished_Name_Type function declarations 1345 > ≡
int set(string distinguished_name_str, int uuser_id = -1, string *uuser_name = 0);
```

See also sections 1351, 1354, 1356, 1358, 1360, 1362, and 1364.

This code is used in section 1338.

1346.

```
< Distinguished_Name_Type function definitions 1342 > +≡
int Distinguished_Name_Type::set(string s, int uuser_id, string *uuser_name){ bool
    DEBUG = false; /* true */
    set_debug_level(DEBUG);
    string::size_type;
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Entering " Distinguished_Name_Type::set'." << endl;
        cerr << "s== " << s << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
    if (s[0] == '\'' || s[0] == '<') s.erase(0, 1);
    if (uuser_id > 0) user_id = uuser_id;
    if (uuser_name) {
        user_name = *uuser_name;
    }
}
```

1347. Erase the closing quotation mark or angle brace. [LDF 2010.05.25.]

```
< Distinguished_Name_Type function definitions 1342 > +≡
t = s.find_last_of('\'');
if (t != string::npos) s.erase(t);
t = s.find_last_of('>');
if (t != string::npos) s.erase(t);
```

1348.

```
< Distinguished_Name_Type function definitions 1342 > +≡
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "s== " << s << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
```

1349.

```

⟨ Distinguished_Name_Type function definitions 1342 ⟩ +≡
string temp_str = s;
string keyword;
string value;
while (temp_str.size() > 0) {
    if (temp_str[0] == '/') temp_str.erase(0, 1);
    t = temp_str.find('=');
    keyword = temp_str.substr(0, t);
    temp_str.erase(0, t + 1);
    t = temp_str.find('/');
    if (t == string::npos) {
        value = temp_str;
        temp_str.clear();
    }
    else {
        value = temp_str.substr(0, t);
        temp_str.erase(0, t);
    }
}
#endif 0
cerr << "keyword== " << keyword << endl;
cerr << "value== " << value << endl;
#endif
if (keyword == "C") countryName = value;
else if (keyword == "ST") stateOrProvinceName = value;
else if (keyword == "L") localityName = value;
else if (keyword == "O") organization = value;
else if (keyword == "OU") organizationalUnitName = value;
else if (keyword == "CN") commonName = value;
} /* while */
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "Exiting 'Distinguished_Name_Type::set' successfully with return value 0. " <<
        endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
return 0; } /* End of Distinguished_Name_Type::set definition */

```

1350. Operators. [LDF 2013.05.10.]

1351. Assignment from an X509_Cert_Type. [LDF 2013.05.10.]

Log

[LDF 2013.05.10.] Added this function.

```

⟨ Distinguished_Name_Type function declarations 1345 ⟩ +≡
void operator=(const X509_Cert_Type &);

```

1352.

```
⟨ Distinguished_Name_Type::operator=(const X509_Cert_Type &) definition 1352 ⟩ ≡
void Distinguished_Name_Type::operator=(const X509_Cert_Type &cert)
{
    organization = cert.organization;
    organizationalUnitName = cert.organizationalUnitName;
    commonName = cert.commonName;
    countryName = cert.countryName;
    localityName = cert.localityName;
    stateOrProvinceName = cert.stateOrProvinceName;
    user_id = cert.user_id;
    user_name = cert.user_name;
    return;
} /* End of Distinguished_Name_Type::operator=(const x509_Cert_Type&) definition */
```

This code is used in section 1366.

1353. Equality. [LDF 2010.04.23.]

1354. const Distinguished_Name_Type & argument. [LDF 2010.04.23.]

Log

[LDF 2010.04.23.] Added this function.

```
⟨ Distinguished_Name_Type function declarations 1345 ⟩ +≡
bool operator≡(const Distinguished_Name_Type &d) const;
```

1355.

```
⟨ Distinguished_Name_Type::operator≡ definition 1355 ⟩ ≡
bool Distinguished_Name_Type::operator≡(const Distinguished_Name_Type &d) const
{
    if (this ≡ &d) return true;
    /* user_name and user_id are ignored for the sake of comparing for equality. [LDF 2010.04.23.] */
    return (organization ≡ d.organization ∧ organizationalUnitName ≡
            d.organizationalUnitName ∧ commonName ≡ d.commonName ∧ countryName ≡
            d.countryName ∧ localityName ≡ d.localityName ∧ stateOrProvinceName ≡ d.stateOrProvinceName);
} /* End of Distinguished_Name_Type::operator≡(const Distinguished_Name_Type &) definition */
```

See also section 1357.

This code is used in section 1366.

1356. const Distinguished_Name_Type & argument. [LDF 2013.05.19.]

Log

[LDF 2013.05.19.] Added this function.

```
⟨ Distinguished_Name_Type function declarations 1345 ⟩ +≡
bool operator≡(const string &s) const;
```

1357.

```
⟨ Distinguished_Name_Type::operator≡ definition 1355 ⟩ +≡
bool Distinguished_Name_Type::operator≡(const string &s) const
{
    Distinguished_Name_Type d;
    d.set(s);
    return operator≡(d);
} /* End of Distinguished_Name_Type::operator≡(const string &) definition */
```

1358. Inequality. [LDF 2011.05.11.]

Log

[LDF 2011.05.11.] Added this function.

```
⟨ Distinguished_Name_Type function declarations 1345 ⟩ +≡
bool operator≠(const Distinguished_Name_Type &);
```

1359.

```
⟨ Distinguished_Name_Type::operator≠ definition 1359 ⟩ ≡
bool Distinguished_Name_Type::operator≠(const Distinguished_Name_Type &d) const
{
    return (!operator≡(d));
} /* End of Distinguished_Name_Type::operator≠(const Distinguished_Name_Type &) definition */
```

This code is used in section 1366.

1360. Clear. [LDF 2010.04.22.]

Log

[LDF 2010.04.22.] Added this function.

```
⟨ Distinguished_Name_Type function declarations 1345 ⟩ +≡
void clear(void);
```

1361.

```
⟨ Distinguished_Name_Type function definitions 1342 ⟩ +≡
void Distinguished_Name_Type::clear(void)
{
    organization = "";
    organizationalUnitName = "";
    commonName = "";
    countryName = "";
    localityName = "";
    stateOrProvinceName = "";
    user_name = "";
    user_id = -1;
    return;
} /* End of Distinguished_Name_Type::clear definition */
```

1362. Output. [LDF 2013.07.03.]

Log

[LDF 2013.07.03.] Added this function.

⟨ Distinguished_Name_Type function declarations 1345 ⟩ +≡
 string *output*(**void**);

1363.

〈Distinguished_Name_Type function definitions 1342〉 +≡

```
string Distinguished_Name_Type::output(void)
```

{

```
stringstream temp_strm;
```

```

temp_strm << "/C=" << countryName << "/O=" << organization << "/OU=" << organizationalUnitName <<
    "/L=" << localityName << "/ST=" << stateOrProvinceName << "/CN=" << commonName;
return temp_strm.str();
/* End of Distinguished_Name_Type::output definition */

```

364. Show. [LDF 2010.04.21.]

Log

[LDF 2010.04.21.] Added this function.

[LDF 2010.05.05.] Made this function **const**.

```
< Distinguished_Name_Type function declarations 1345 > +≡  
void show(string s = "", stringstream *strm = 0) const;
```

1365.

〈Distinguished_Name_Type function definitions 1342〉 +≡

```
void Distinguished_Name_Type :: show(string s, stringstream *strm) const
```

{

```
if (s ≡ "") s = "Distinguished_Name_Type:";
```

```
stringstream temp_strm;
```

```

temp_strm << setfill(' ') << s << endl << setw(34) << std::left << "organization:" <<
organization << endl << setw(34) << std::left << "organizationalUnitName:" <<
organizationalUnitName << endl << setw(34) << std::left << "commonName:" <<
commonName << endl << setw(34) << std::left << "countryName:" << countryName <<
endl << setw(34) << std::left << "localityName:" << localityName << endl <<
setw(34) << std::left << "stateOrProvinceName:" << stateOrProvinceName << endl <<
setw(34) << std::left << "user_id:" << user_id << endl << setw(34) << std::left <<
"user_name:" << user_name << dec << endl;

```

```
if (strm) *strm <= temp strm.str( );
```

```
else cerr << temp.strm.str();
```

return;

} /* End of Distinguished_Name_Type::show definition */

1366. Putting **class Distinguished_Name_Type** together. [LDF 2010.04.21.]

```
⟨ Include files 3 ⟩
using namespace std;
using namespace gwrdifpk;
class X509_Cert_Type;
⟨ class Distinguished_Name_Type declaration 1338 ⟩
⟨ Distinguished_Name_Type function definitions 1342 ⟩
⟨ Distinguished_Name_Type::operator≡ definition 1355 ⟩
⟨ Distinguished_Name_Type::operator≠ definition 1359 ⟩
⟨ Distinguished_Name_Type::operator=(const X509_Cert_Type &) definition 1352 ⟩
```

1367.

```
⟨ dstngnmt.h 1367 ⟩ ≡
#ifndef DSTNGNMT_H
#define DSTNGNMT_H 1
class X509_Cert_Type;
⟨ class Distinguished_Name_Type declaration 1338 ⟩
#endif
```

1368. Functions for GNUTLS (gntlsfnc.web).

1369. Include files. [LDF 2009.11.25.]

```
<Include files 3> +≡
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <string.h>
#include <unistd.h>
#include <gnutls/gnutls.h>
#include <gnutls/x509.h>
#include <gcrypt.h> /* for gcry-control */
#include <iomanip>
#include <iostream>
#include <map>
#include <fstream>
#include <sstream>
#include <string>
#include <vector>
#include <deque>
#include <stack>
#include <set>
#include <expat.h>
#include <pthread.h>
#ifndef HAVE_CONFIG_H
#include <config.h>
#endif
#include <mysql.h>
#include "glblcnst.h++"
#include "glblvrbl.h++"
#include "excptntp.h++"
#include "utilfncts.h++"
#include "parser.h++"
#include "scanner.h++"
#include "grouptp.h++"
#include "hndlvltp.h++"
#include "rspnntp.h++"
#include "irdsavtp.h++"
#include "irdsobtp.h++"
#include "hndltype.h++"
#include "dcmtddtp.h++"
#include "x509cert.h++"
#include "dstngnmt.h++"
#include "scprpmtp.h++"
#ifndef _XOPEN_SOURCE
#define _XOPEN_SOURCE
#endif
```

1370. Preprocessor macro definitions. [LDF 2009.11.25.]

```
⟨ Preprocessor macro definitions 1262 ⟩ +≡
#define TLS_SESSION_CACHE 50
#define MAX_SESSION_ID_SIZE 32
#define MAX_SESSION_DATA_SIZE 512
#define DH_BITS 1024
```

1371. struct CACHE. [LDF 2009.11.25.]

```
⟨ Type definitions 1371 ⟩ ≡
typedef struct {
    char session_id[MAX_SESSION_ID_SIZE];
    int session_id_size;
    char session_data[MAX_SESSION_DATA_SIZE];
    int session_data_size;
} CACHE;
```

This code is used in section 1409.

1372. Global variables. [LDF 2009.11.25.]

```
⟨ Global variables 310 ⟩ +≡
static CACHE *cache_db;
static int cache_db_ptr = 0;
```

1373.

```
⟨ Declare functions 1373 ⟩ ≡
void wrap_db_init(void);
```

See also sections 1375, 1377, 1379, 1381, 1383, 1386, 1397, 1399, 1401, 1415, and 1419.

This code is used in sections 1409, 1410, 1423, and 1424.

1374.

```
⟨ Define functions 1374 ⟩ ≡
void wrap_db_init(void)
{
    /* allocate cache_db */
    cache_db = static_cast<CACHE *>(calloc(1, TLS_SESSION_CACHE * sizeof(CACHE)));
}
```

See also sections 1376, 1378, 1380, 1382, 1384, 1387, 1398, 1400, 1402, 1416, 1417, 1418, and 1420.

This code is used in sections 1409 and 1423.

1375.

```
⟨ Declare functions 1373 ⟩ +≡
void wrap_db_deinit(void);
```

1376.

```
⟨ Define functions 1374 ⟩ +≡
void wrap_db_deinit(void)
{
    if (cache_db) free(cache_db);
    cache_db = Λ;
    return;
}
```

1377.

⟨ Declare functions 1373 ⟩ +≡
int *wrap_db_store*(**void** **dbf*, *gnutls_datum_t* *key*, *gnutls_datum_t* *data*);

1378.

⟨ Define functions 1374 ⟩ +≡
int *wrap_db_store*(**void** **dbf*, *gnutls_datum_t* *key*, *gnutls_datum_t* *data*)
{
 if (*cache_db* ≡ Λ) **return** -1;
 if (*key.size* > MAX_SESSION_ID_SIZE) **return** -1;
 if (*data.size* > MAX_SESSION_DATA_SIZE) **return** -1;
 memcpy(*cache_db*[*cache_db_ptr*].*session_id*, *key.data*, *key.size*);
 cache_db[*cache_db_ptr*].*session_id_size* = *key.size*;
 memcpy(*cache_db*[*cache_db_ptr*].*session_data*, *data.data*, *data.size*);
 cache_db[*cache_db_ptr*].*session_data_size* = *data.size*;
 cache_db_ptr++;
 cache_db_ptr %= TLS_SESSION_CACHE;
 return 0;
}

1379.

⟨ Declare functions 1373 ⟩ +≡
gnutls_datum_t *wrap_db_fetch*(**void** **dbf*, *gnutls_datum_t* *key*);

1380.

⟨ Define functions 1374 ⟩ +≡
gnutls_datum_t *wrap_db_fetch*(**void** **dbf*, *gnutls_datum_t* *key*)
{
 gnutls_datum_t *res* = {Λ, 0};
 int *i*;
 if (*cache_db* ≡ Λ) **return** *res*;
 for (*i* = 0; *i* < TLS_SESSION_CACHE; *i*++) {
 if (*key.size* ≡ *cache_db*[*i*].*session_id_size* ∧ *memcmp*(*key.data*, *cache_db*[*i*].*session_id*, *key.size*) ≡ 0) {
 res.size = *cache_db*[*i*].*session_data_size*;
 res.data = **static_cast**<unsigned char *>(*gnutls_malloc*(*res.size*));
 if (*res.data* ≡ Λ) **return** *res*;
 memcpy(*res.data*, *cache_db*[*i*].*session_data*, *res.size*);
 return *res*;
 }
 }
 return *res*;
}

1381.

⟨ Declare functions 1373 ⟩ +≡
int *wrap_db_delete*(**void** **dbf*, *gnutls_datum_t* *key*);

1382.

⟨ Define functions 1374 ⟩ +≡

```
int wrap_db_delete(void *dbf, gnutls_datum_t key)
{
    int i;
    if (cache_db == NULL) return -1;
    for (i = 0; i < TLS_SESSION_CACHE; i++) {
        if (key.size == cache_db[i].session_id_size & memcmp(key.data, cache_db[i].session_id, key.size) == 0) {
            cache_db[i].session_id_size = 0;
            cache_db[i].session_data_size = 0;
            return 0;
        }
    }
    return -1;
} /* End of wrap_db_delete definition */
```

1383. print_info. [LDF 2009.12.03.]

⟨ Declare functions 1373 ⟩ +≡

```
int print_info(gnutls_session_t session);
```

1384.

```

⟨ Define functions 1374 ⟩ +≡
int print_info(gnutls_session_t session)
{
    cerr << "GnuTLS_version:" << gnutls_check_version(0) << endl;
const char *tmp;
gnutls_credentials_type_t cred;
gnutls_kx_algorithm_t kx; /* print the key exchange's algorithm name */
kx = gnutls_kx_get(session);
tmp = gnutls_kx_get_name(kx);
fprintf(stderr, "-_Key_Exchange:_%s\n", tmp);
/* Check the authentication type used and switch * to the appropriate. */
cred = gnutls_auth_get_type(session);
switch (cred) {
case GNUTLS_CRD_IA: fprintf(stderr, "-_TLS/IA_session\n");
break;
#endif ENABLE_SRP
case GNUTLS_CRD_SRP:
    fprintf(stderr, "-_SRP_session_with_username_%s\n", gnutls_srp_server_get_username(session));
    break;
#endif
case GNUTLS_CRD_PSK: /* This returns NULL in server side. */
if (gnutls_psk_client_get_hint(session) != Λ)
    fprintf(stderr, "-_PSK_authentication._PSK_hint_%s\n", gnutls_psk_client_get_hint(session));
    /* This returns NULL in client side. */
if (gnutls_psk_server_get_username(session) != Λ)
    fprintf(stderr, "-_PSK_authentication._Connected_as_%s\n",
            gnutls_psk_server_get_username(session));
    break;
case GNUTLS_CRD_ANON: /* anonymous authentication */
    fprintf(stderr, "-_Anonymous_DH_using_prime_of_%d_bits\n", gnutls_dh_get_prime_bits(session));
    break;
case GNUTLS_CRD_CERTIFICATE: /* certificate authentication */
    /* Check if we have been using ephemeral Diffie-Hellman. */
if (kx ≡ GNUTLS_KX_DHE_RSA ∨ kx ≡ GNUTLS_KX_DHE_DSS) {
    fprintf(stderr, "\n-_Ephemeral_DH_using_prime_of_%d_bits\n",
            gnutls_dh_get_prime_bits(session));
}
    /* if the certificate list is available, then print some information about it. */
    print_x509_certificate_info(session);
} /* switch */ /* print the protocol's name (ie TLS 1.0) */
tmp = gnutls_protocol_get_name(gnutls_protocol_get_version(session));
fprintf(stderr, "-_Protocol:_%s\n", tmp); /* print the certificate type of the peer, i.e., X.509 */
tmp = gnutls_certificate_type_get_name(gnutls_certificate_type_get(session));
fprintf(stderr, "-_Certificate_Type:_%s\n", tmp);
/* print the compression algorithm (if any) */
tmp = gnutls_compression_get_name(gnutls_compression_get(session));
fprintf(stderr, "-_Compression:_%s\n", tmp); /* print the name of the cipher used. * ie 3DES. */
tmp = gnutls_cipher_get_name(gnutls_cipher_get(session));
fprintf(stderr, "-_Cipher:_%s\n", tmp); /* Print the MAC algorithms name. * ie SHA1 */
tmp = gnutls_mac_get_name(gnutls_mac_get(session));
fprintf(stderr, "-_MAC:_%s\n", tmp);

```

```

    return 0;
} /* End of print_info definition */

```

1385. bin2hex.**1386.**

⟨ Declare functions 1373 ⟩ +≡
const char *bin2hex(const void *bin, size_t bin_size);

1387.

⟨ Define functions 1374 ⟩ +≡
const char *bin2hex(const void *bin, size_t bin_size)
{
 static char printable[110];
 const unsigned char *_bin = (const unsigned char *) bin;
 char *print;
 size_t i;
 if (bin_size > 50) bin_size = 50;
 print = printable;
 for (i = 0; i < bin_size; i++) {
 sprintf(print, "%.2x\u202a", _bin[i]);
 print += 2;
 }
 return printable;
}

1388. print_x509_certificate_info. [LDF 2009.12.03.]

This function will print information about this session's peer certificate.

⟨ *print_x509_certificate_info* declaration 1388 ⟩ ≡
void print_x509_certificate_info(gnutls_session_t session);

This code is used in sections 1409 and 1410.

1389.

```
<print_x509_certificate_info definition 1389> ≡
void print_x509_certificate_info(gnutls_session_t session){ bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    char serial[40];
    char dn[256];
    size_t size;
    unsigned int algo, bits;
    time_t expiration_time, activation_time;
    const gnutls_datum_t* cert_list;
    unsigned int cert_list_size = 0;
    gnutls_x509_crt_t cert;
    char message_str[256];
    int status = 0; /* This function only works for X.509 certificates. */
    cerr << "***Entering 'print_x509_certificate_info'" << endl;
    if (gnutls_certificate_type_get(session) ≠ GNUTLS_CRT_X509) {
        cerr << "WARNING! In 'print_x509_certificate_info':"
            << endl << "Certificate type isn't X.509." << endl << "Exiting function and continuing."
            << endl;
        return;
    } /* if (gnutls_certificate_type_get(session) ≠ GNUTLS_CRT_X509) */
    cert_list = gnutls_certificate_get_peers(session, &cert_list_size);
    cerr << "Peer provided" << cert_list_size << " certificates." << endl;
}
```

See also sections 1390, 1391, 1392, 1393, 1394, and 1395.

This code is used in section 1409.

1390.

```
<print_x509_certificate_info definition 1389> +≡
#if 0 /* From [...]/gnutls/include/gnutls/x509.h LDF 2009.12.21. */
    typedef void *gnutls_x509_dn_t;
    typedef struct gnutls_x509_ava_st {
        gnutls_datum_t oid;
        gnutls_datum_t value;
        unsigned long value_tag;
    } gnutls_x509_ava_st; /* From [...]/gnutls/include/gnutls/gnutls.h LDF 2009.12.21. */
    typedef struct {
        unsigned char *data;
        unsigned int size;
    } gnutls_datum_t;
#endif
#if 0
    GNUTLS_CRT_PRINT_FULLGNUTLS_CRT_PRINT_ONELINE
    /* gnutls_x509_crt_print: Defined in [...]/gnutls/lib/x509/output.c */
    /* Numerical codes for the fields in the DN ("Distinguished Name") can be found in this file:
       [...]/gnutls/doc/protocol/x509guide.txt LDF 2009.12.21. */
#endif
```

1391.

```
<print_x509_certificate_info definition 1389> +≡
    gnutls_datum_t out;
```

1392.

```

⟨ print_x509_certificate_info definition 1389 ⟩ +≡
  if (cert_list_size > 0) { gnutls_x509_crt_init(&cert); for (int i = 0; i < cert_list_size; ++i) {
    gnutls_x509_crt_import(cert, &cert_list[i], GNUTLS_X509_FMT_DER);
  if (DEBUG) cerr ≪ "Certificate_info_for_certificate_"
    ≪ i ≪ endl;

```

1393. Get subject information. [LDF 2009.12.21.]

This works. Not calling `extract_dn_fields` at present, because it's called elsewhere and I don't want the terminal output at present. [LDF 2009.12.22.]

```

⟨ print_x509_certificate_info definition 1389 ⟩ +≡
#if 0
    extract_dn_fields(cert, true);      /* Subject */
#endif

```

1394

```
< print_x509_certificate_info definition 1389 > +≡  
#if 0  
    extract_dn_fields(cert, false);      /* Issuer */  
#endif
```

1395.

```

< print_x509_certificate_info definition 1389 > +≡
    expiration_time = gnutls_x509_crt_get_expiration_time(cert);
    activation_time = gnutls_x509_crt_get_activation_time(cert);
    cerr ≪ "Certificate is valid since: " ≪ ctime(&activation_time) ≪ endl;
#endif 0
/* !! TODO: LDF 2012.01.18. BUG!! This prints the same information as ctime(&activation_time),
   i.e., the contents of the "Not Before" field. See whether it's been fixed in a newer version of
   GNUTLS. Otherwise, submit bug report. */
    ≪ "Certificate expires: " ≪ ctime(&expiration_time) ≪ endl;
#endif /* Print the serial number of the certificate. */
size = sizeof(serial);
gnutls_x509_crt_get_serial(cert, serial, &size);
cerr ≪ "Certificate serial number: " ≪ bin2hex(serial, size) ≪ endl;
/* Extract some of the public key algorithm's parameters */
algo = gnutls_x509_crt_get_pk_algorithm(cert, &bits);
cerr ≪ "Certificate public key: " ≪ gnutls_pk_algorithm_get_name((gnutls_pk_algorithm_t)algo) ≪
      endl; /* Print the version of the X.509 certificate. */
cerr ≪ "Certificate version: #" ≪ gnutls_x509_crt_get_version(cert) ≪ endl;
size = sizeof(dn);
gnutls_x509_crt_get_dn(cert, dn, &size);
cerr ≪ "sizeof(DN): " ≪ size ≪ endl ≪ "DN: " ≪ dn ≪ endl;
size = sizeof(dn);
gnutls_x509_crt_get_issuer_dn(cert, dn, &size);
cerr ≪ "Issuer's DN: " ≪ dn ≪ endl; } /* for */
gnutls_x509_crt_deinit(cert); } /* if (cert_list_size > 0) */
if (DEBUG) {
    cerr ≪ "*** Exiting 'print_x509_certificate_info' successfully." ≪ endl;
} /* if (DEBUG) */
return; } /* End of print_x509_certificate_info definition */

```

1396.

```
< Global variables 310 > +≡      /* Export-grade cipher suites require temporary RSA keys. */
static char srp_dh_group2048[] = "-----BEGIN_DH PARAMETERS-----\n"
"MIIBBwKCAQCsa9tBMkq\am/Fm314TiVgvr3K2ZRmH7gf8MZKUPbVgUKNzKcu0oJnt\n"
"gZPgdXdnoT3VIxKrSwM\xDc1/SKnaBP1Q6Ag5ae23Z7DPYJUXmhY6s2YaBfvV+qro\n"
"KRipli8Lk7hV+XmT7Jd\ne6qgNdArb9P90c1nQQdXDPqcdKB5EaxR308qXtDoj+4AW\n"
"dr0gekNsZIHx0rkHhx\GGludMuai+HdIVEUjtSSw1X1ep3onddLs+gMs+9v1L7N4\n"
"YWAnkATleuavh05zA85\TKZzMBBx7wwjYK1aY86jQw4JxrjX46dv7tpS1yAPYn3rk\n"
"Nd4jbVJfVHWbZeNy/Na\08g+nER+eSv9zAgEC\n"
-----END_DH PARAMETERS-----\n";
```

1397.

```
< Declare functions 1373 > +≡
int generate_dh_params(gnutls_dh_params_t & dh_params);
```

1398.

```
< Define functions 1374 > +≡
int generate_dh_params(gnutls_dh_params_t & dh_params)
{
    bool DEBUG = false;      /* true */
    set_debug_level(DEBUG, 0, 0);
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ "Entering_generate_dh_params." ≪ endl;
        unlock_cerr_mutex();
    }
    gnutls_datum_t dparams = {reinterpret_cast<unsigned char *>(srp_dh_group2048), sizeof
        (srp_dh_group2048)};      /* Here instead of generating Diffie Hellman parameters (for use with
        DHE kx algorithms) we import them. */
    gnutls_dh_params_init(&dh_params);
    gnutls_dh_params_import_pkcs3(dh_params, &dparams, GNUTLS_X509_FMT_PEM);
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ "Exiting_generate_dh_params." ≪ endl;
        unlock_cerr_mutex();
    }
    return 0;
}
```

1399.

```
< Declare functions 1373 > +≡
int generate_rsa_params(gnutls_rsa_params_t & rsa_params);
```

1400.

```

⟨ Define functions 1374 ⟩ +≡
int generate_rsa_params(gnutls_rsa_params_t & rsa_params)
{
    bool DEBUG = false;      /* true */
    set_debug_level(DEBUG, 0, 0);
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Entering generate_rsa_params." << endl;
        unlock_cerr_mutex();
    }      /* Generate RSA parameters - for use with RSA-export * cipher suites. This is an RSA private
           key and should be * discarded and regenerated once a day, once every 500 * transactions etc.
           Depends on the security requirements. */
#ifndef IS_OPTINUM_SRV
#ifndef 0
    int status = gnutls_rsa_params_generate2(rsa_params, 31);      /* 512 */
#endif
#endif      /* !! TODO: This fails on ‘optimum-srv’. Find out why. LDF 2010.04.16. */
    gnutls_rsa_params_init(&rsa_params);
    int status = gnutls_rsa_params_generate2(rsa_params, 512);
#endif
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "Exiting generate_rsa_params." << endl;
    unlock_cerr_mutex();
}
return 0;
}      /* End of generate_rsa_params definition */

```

1401. initialize_tls_session.

```

⟨ Declare functions 1373 ⟩ +≡
gnutls_session_t initialize_tls_session(gnutls_certificate_credentials_t & cert_cred);

```

1402.

```

⟨ Define functions 1374 ⟩ +≡
gnutls_session_t initialize_tls_session(gnutls_certificate_credentials_t & cert_cred)
{
    gnutls_session_t session;
    gnutls_init(&session, GNUTLS_SERVER);      /* Use the default priorities, plus, export cipher suites. */
    gnutls_priority_set_direct(session, "EXPORT", Λ);
    gnutls_credentials_set(session, GNUTLS_CRD_CERTIFICATE, cert_cred);
    /* request client certificate if any. */
    gnutls_certificate_server_set_request(session, GNUTLS_CERT_REQUEST);
    gnutls_dh_set_prime_bits(session, DH_BITS);
    if (TLS_SESSION_CACHE ≠ 0) {
        gnutls_db_set_retrieve_function(session, wrap_db_fetch);
        gnutls_db_set_remove_function(session, wrap_db_delete);
        gnutls_db_set_store_function(session, wrap_db_store);
        gnutls_db_set_ptr(session, Λ);
    }
    return session;
}      /* End of initialize_tls_session definition */

```

1403. Extract DN fields. [LDF 2009.12.21.]

The **bool** *subject* argument should be *false* when this function is used for the issuer's certificate. [LDF 2009.12.30.] ■

Log

[LDF 2009.12.21.] Added this function.

[LDF 2009.12.23.] Added the argument **X509_Cert_Type** **x509_cert*. The extracted information is stored in the object it references. Also added the argument **Scan_Parse_Parameter_Type** **param*.

[LDF 2013.05.08.] Added this function to **gwrdifpk**. I've copied it from the **dbsrvcli** package (from the OptiNum-Grid project).

⟨ *extract_dn_fields* declaration 1403 ⟩ ≡

```

int extract_dn_fields(gnutls_x509_crt_t & cert, X509_Cert_Type *x509_cert = 0, bool
                      subject = true, Scan_Parse_Parameter_Type *param = 0);

```

This code is used in sections 1409 and 1410.

1404.

```

⟨ extract_dn_fields definition 1404 ⟩ ≡
int extract_dn_fields(gnutls_x509_crt_t & cert, X509_Cert_Type *x509_cert, bool subject,
                      Scan_Parse_Parameter_Type *param){ bool DEBUG = false; /* true */
set_debug_level(DEBUG, 0, 0);
string thread_ctr_str = "";
if (param) {
    stringstream s;
    s << "[Thread:" << param->get_thread_ctr() << "]";
    thread_ctr_str = s.str();
}
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "***Entering‘extract_dn_fields’." << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
gnutls_x509_ava_st curr_ava;
gnutls_x509_dn_t curr_dn;
string value_str = "";
string oid_str = "";
int status = 0;
if (subject) {
    status = gnutls_x509_crt_get_subject(cert, &curr_dn);
    if (DEBUG) cerr << 'gnutls_x509_crt_get_subject' returned " << status << endl;
}
else {
    status = gnutls_x509_crt_get_issuer(cert, &curr_dn);
    if (DEBUG) cerr << 'gnutls_x509_crt_get_issuer' returned " << status << endl;
}

```

See also sections 1405 and 1406.

This code is used in section 1409.

1405.

```

⟨ extract_dn_fields definition 1404 ⟩ +≡
for (int j = 0; ; ++j) {
    if (DEBUG) cerr << "j==" << j << endl;
    status = gnutls_x509_dn_get_rdn_ava(curr_dn, j, 0, &curr_ava);
    if (DEBUG) cerr << 'gnutls_x509_dn_get_rdn_ava' returned " << status << endl;
    if (status ≠ 0) {
        if (DEBUG) cerr << "Breaking." << endl;
        break;
    }
}

```

1406. *curr_ava.value.data* and *curr_ava.oid.data* are copied to **strings**, because it's a lot easier to handle **strings** than it is to handle arrays of **unsigned char**. [LDF 2009.12.21.]

Log

[LDF 2009.12.22.] Now checking that *curr_ava.value.data*[*k*] > 0 and *curr_ava.oid.size* & *curr_ava.oid.data*[*k*] > 0 before putting the characters onto the corresponding strings. Otherwise, it's possible that an ASCII null character could get onto one (or both) of them.

```
{ extract_dn_fields definition 1404 } +≡
value_str = "";
for (int k = 0; k < curr_ava.value.size & curr_ava.value.data[k] > 0; ++k)
    value_str += static_cast<char>(curr_ava.value.data[k]);
oid_str = "";
for (int k = 0; k < curr_ava.oid.size & curr_ava.oid.data[k] > 0; ++k)
    oid_str += static_cast<char>(curr_ava.oid.data[k]);
if (DEBUG)
    cerr << "curr_ava.value.data==\" " << value_str << endl << "curr_ava.value.size==\" " <<
        curr_ava.value.size << endl << "curr_ava.value_tag==\" " << curr_ava.value_tag << endl <<
        "curr_ava.oid.data==\" " << oid_str << endl << endl << "curr_ava.oid.size==\" " <<
        curr_ava.oid.size << endl << "value_str==\" " << value_str << endl << "oid_str==\" " <<
        oid_str << endl << "dn_fields[\" " << oid_str << "\"]==\" " << dn_fields[oid_str] << endl;
if (oid_str == "2.5.4.3") x509_cert-commonName = value_str;
else if (oid_str == "2.5.4.6") x509_cert-countryName = value_str;
else if (oid_str == "2.5.4.7") x509_cert-localityName = value_str;
else if (oid_str == "2.5.4.8") x509_cert-stateOrProvinceName = value_str;
else if (oid_str == "2.5.4.10") x509_cert-organization = value_str;
else if (oid_str == "2.5.4.11") x509_cert-organizationalUnitName = value_str;
} /* for */
if (DEBUG) {
    cerr << "Exiting 'extract_dn_fields' successfully with return value 0." << endl;
} /* if (DEBUG) */
return 0; /* End of extract_dn_fields definition */
```

1407.

```
{ Garbage 303 } +≡
#ifndef 0
#endif
```

1408. Putting Functions for GNUTLS together.

1409. This is what's compiled. [LDF 2013.05.08.]

```
using namespace std;
⟨ Include files 3 ⟩
using namespace gwrdifpk;
⟨ Preprocessor macro definitions 1262 ⟩
⟨ Type definitions 1371 ⟩
⟨ Global variables 310 ⟩
void lock_cerr_mutex(void);
void unlock_cerr_mutex(void);
#ifndef 0 /* 1 */
⟨ Garbage 303 ⟩
#endif
⟨ Declare functions 1373 ⟩
⟨ print_x509_certificate_info declaration 1388 ⟩
⟨ extract_dn_fields declaration 1403 ⟩
⟨ Define functions 1374 ⟩
⟨ print_x509_certificate_info definition 1389 ⟩
⟨ extract_dn_fields definition 1404 ⟩
```

1410. This is what's written to the header file `gntlsfnc.h`. [LDF 2013.05.08.]

```
⟨ gntlsfnc.h 1410 ⟩ ≡
#ifndef GNTLSFNC_H
#define GNTLSFNC_H 1
⟨ Preprocessor macro definitions 1262 ⟩
⟨ Declare functions 1373 ⟩
⟨ print_x509_certificate_info declaration 1388 ⟩
⟨ extract_dn_fields declaration 1403 ⟩
#endif
```

1411. Helper functions for GNUTLS examples.

7.3.10 Helper Function for TCP Connections

This helper function abstracts away TCP connection handling from the other examples. It is required to build some examples.

Copyright 2007, 2008, 2009 Free Software Foundation

Copying and distribution of this file, with or without modification, are permitted in any medium without royalty provided the copyright notice and this notice are preserved.

1412.

```
<Include files 3> +≡
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <netdb.h>
#include <unistd.h>
#include <string>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <map>
#include <deque>
#include <vector>
#include <mysql.h>
#if HAVE_CONFIG_H
#include <config.h>
#endif
#include "glblcnst.h++"
#include "glblvrbl.h++"
#include "utilfncts.h++"
```

1413.

```
<Preprocessor definitions 1413> ≡
#ifndef _XOPEN_SOURCE
#define _XOPEN_SOURCE
#endif
#define SStruct sockaddr
```

This code is used in section 1423.

1414. *tcp_connect*. Connects to the peer and returns a socket descriptor.

1415.**Log**

[LDF 2009.12.15.] Removed **const char *PORT = "5556"** and **const char *SERVER = "127.0.0.1"** and replaced them with optional **string** arguments. The defaults are the previous fixed values of ***PORT** and ***SERVER**.

[LDF 2009.12.18.] Now using *getaddrinfo*. This makes it possible to pass the IP address of the server to the client using a name such as as “localhost” or “pcfinston”. Previously, an explicit address with digits and periods only was required.

[LDF 2010.01.27.] BUG FIX: Now using **string ip_address** argument. Previously, Λ was used, which caused the client to connect only to a server on the same computer.

[LDF 2013.08.29.] Removed default values for arguments.

⟨ Declare functions 1373 ⟩ +≡

```
int tcp_connect(string ip_address, string port_str);
```

1416.

```

⟨ Define functions 1374 ⟩ +≡
int tcp_connect(string ip_address, string port_str){ bool DEBUG = false;      /* true */
    set_debug_level(DEBUG, 0, 0);

    int err;
    int sd;
    struct sockaddr_in sa;
    int rv;
    struct addrinfo hints;
    struct addrinfo *servinfo;
    struct addrinfo *p;

    memset(&hints, 0, sizeof hints);
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;
    hints.ai_flags = AI_PASSIVE;      /* use my IP */
    if ((rv = getaddrinfo(ip_address.c_str(), port_str.c_str(), &hints, &servinfo)) != 0) {
        cerr << "ERROR! In 'tcp_connect': 'getaddrinfo' failed, returning" << rv << ":" <<
            endl << gai_strerror(rv);
        return 1;
    }
    else if (DEBUG) {
        cerr << "In 'tcp_connect': 'getaddrinfo' succeeded, returning" << rv << "." << endl;
    } /* else if (DEBUG) */ /* loop through all the results and connect to the first we can */
    int i = 0;
#endif 1
    if (DEBUG) {
        for (p = servinfo; p != NULL; p = p->ai_next) i++;
        cerr << i << " elements in 'servinfo'" << endl;
    } /* if (DEBUG) */
#endif
    i = 0;      /* Now only outputting error message with perror if debugging, or it's the last address.
                    LDF 2012.03.14. */
    for (p = servinfo; p != NULL; p = p->ai_next) {
#endif 0
        cerr << "i==" << i << endl;
#endif
        if ((sd = socket(p->ai_family, p->ai_socktype, p->ai_protocol)) == -1) {
            if (DEBUG || p->ai_next == 0) perror("client:socket");
            continue;
        }
        if (connect(sd, p->ai_addr, p->ai_addrlen) == -1) {
            close(sd);
            if (DEBUG || p->ai_next == 0) perror("client:connect");
            continue;
        }
        break;
    } /* for */

```

1417.**Log**

[LDF 2010.03.25.] BUG FIX: Now returning -1 instead of 2.

```
< Define functions 1374 > +≡
if (p ≡ Λ) {
    cerr ≪ "In ‘tcp_connect’: Client: failed to connect" ≪ endl;
    return -1;
}
```

1418.

```
< Define functions 1374 > +≡
else
    if (DEBUG) {
        cerr ≪ "In ‘tcp_connect’: Client: connection succeeded." ≪ endl;
    } /* else if (DEBUG) */
int status = 0; /* connects to server */
char s[INET6_ADDRSTRLEN];
void *v;
inet_ntop(p→ai_family, get_in_addr((struct sockaddr *) p→ai_addr), s, sizeof (s));
if (s ∧ DEBUG) cerr ≪ "In ‘tcp_connect’: Client: connecting to" ≪ s ≪ endl;
freeaddrinfo(servinfo); /* all done with this structure */
return sd; } /* End of tcp_connect definition. */
```

1419. Closes the given socket descriptor.

```
< Declare functions 1373 > +≡
void tcp_close(int sd);
```

1420.

```
< Define functions 1374 > +≡
void tcp_close(int sd)
{
    shutdown(sd, SHUT_RDWR); /* no more receptions */
    close(sd);
    sd = 0;
}
```

1421. Get in-address. [LDF 2009.09.29.]

This function comes from *Beej’s Guide to Network Programming*. [LDF 2009.09.29.]
Get *sockaddr*, IPv4 or IPv6.

Log

[LDF 2009.09.29.] Added this function.

```
< get_in_addr declaration 1421 > ≡
void *get_in_addr(struct sockaddr *sa);
```

This code is used in sections 1423 and 1424.

1422.

```
<get_in_addr definition 1422> ≡
void *get_in_addr(struct sockaddr *sa)
{
    if (sa→sa_family ≡ AF_INET) {
        return &(((struct sockaddr_in *) sa)→sin_addr);
    }
    return &(((struct sockaddr_in6 *) sa)→sin6_addr);
}
```

This code is used in section 1423.

1423.

```
<Include files 3>
using namespace std;
<Preprocessor definitions 1413>
<Declare functions 1373>
<get_in_addr declaration 1421>
<Define functions 1374>
<get_in_addr definition 1422>
```

1424.

```
<helper.h 1424> ≡
<Declare functions 1373>
<get_in_addr declaration 1421>
```

1425. Example RFC 2818.

1426. Include files. [LDF 2009.11.30.]

```
<Include files 3> +≡
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <limits.h>
#include <string>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <map>
#include <deque>
#include <vector>
#include <pthread.h>
#include <mysql.h>
#include <gnutls/gnutls.h>
#include <gnutls/x509.h>
#ifndef HAVE_CONFIG_H
#include <config.h>
#endif
#include "glblcnst.h++"
#include "glblvrb.h++"
#include "utilfnscs.h++"
#if 0
#include "entries.h++"
#include "dstngnmt.h++"
#include "glblfnscs.h++"
#endif
#include "x509cert.h++"
#include "gntlsfnc.h++"
```

1427. Verify certificate. [LDF 2009.11.30.]

This function will try to verify the peer's certificate, and also check if the hostname matches, and the activation, expiration dates. [LDF 2009.11.30.]

It is invoked in *client_connect_auth*, which is defined in *cnnctcli.web*. [LDF 2010.08.04.]

Log

[LDF 2010.01.15.] Made *hostname* argument optional with default 0.

[LDF 2010.04.01.] Changed return value from **void** to **int**.

[LDF 2013.05.08.] Added optional argument **X509_Cert_Type** **x509_cert_ptr* = 0.

```
<Declare verify_certificate 1427> ≡
int verify_certificate(gnutls_session_t session, X509_Cert_Type *x509_cert_ptr = 0, const char
*hostname = 0);
```

This code is used in sections 1441 and 1442.

1428.

```

⟨ Define verify_certificate 1428 ⟩ ≡
int verify_certificate(gnutls_session_t session, X509_Cert_Type *x509_cert_ptr, const char *hostname){
    bool DEBUG = false; /* true */
    set_debug_level(DEBUG);
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "***\u00a9Entering\u00a9‘verify_certificate’.\u00a9" << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    unsigned int status;
    const gnutls_datum_t *cert_list;
    unsigned int cert_list_size;
    int ret;
    gnutls_x509_crt_t cert;
    char temp_str[256];
    /* This verification function uses the trusted CAs in the credentials * structure. So you must
       have installed one or more CA certificates. LDF 2009.11.23. This would seem to mean using the
       gnutls_certificate_set_x509_trust_file and gnutls_certificate_set_x509_key_file functions. */
    ret = gnutls_certificate_verify_peers2(session, &status);
    if (ret < 0) {
        strcpy(temp_str, "ERROR\u00a9In\u00a9‘verify_certificate’:\\n");
        strcat(temp_str, "'gnutls_certificate_verify_peers2'\u00a9failed\u00a9returning\u00a9%d\\n");
        lock_cerr_mutex();
        fprintf(stderr, temp_str, ret);
        fprintf(stderr, "***\u00a9Exiting\u00a9‘verify_certificate’\u00a9unsuccessfully\u00a9return\u00a9value\u00a91.\u00a9");
        unlock_cerr_mutex();
        return 1;
    }
}

```

See also sections 1429, 1430, 1431, 1432, 1433, 1434, 1435, 1436, 1437, 1438, 1439, and 1440.

This code is used in section 1441.

1429.**Log**

[LDF 2010.04.01.] Now exiting unsuccessfully when the certificate not trusted, revoked, etc.

```

⟨ Define verify_certificate 1428 ⟩ +≡
if (status & GNUTLS_CERT_INVALID) {
    lock_cerr_mutex();
    fprintf(stderr, "The certificate is not trusted.\n");
    fprintf(stderr, "*** Exiting 'verify_certificate' unsuccessfully with return value 1.\n");
    unlock_cerr_mutex();
    return 1;
}
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        fprintf(stderr, "The certificate is trusted.\n");
        unlock_cerr_mutex();
    }
#endif /* DEBUG_COMPILE */
if (status & GNUTLS_CERT_SIGNER_NOT_FOUND) {
    lock_cerr_mutex();
    fprintf(stderr, "The certificate hasn't got a known issuer.\n");
    fprintf(stderr, "*** Exiting 'verify_certificate' unsuccessfully with return value 1.\n");
    unlock_cerr_mutex();
    return 1;
}
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        fprintf(stderr, "The certificate has got a known issuer.\n");
        unlock_cerr_mutex();
    }
#endif /* DEBUG_COMPILE */
if (status & GNUTLS_CERT_REVOKED) {
    lock_cerr_mutex();
    fprintf(stderr, "The certificate has been revoked.\n");
    fprintf(stderr, "*** Exiting 'verify_certificate' unsuccessfully with return value 1.\n");
    unlock_cerr_mutex();
    return 1;
}
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        fprintf(stderr, "The certificate hasn't been revoked.\n");
        unlock_cerr_mutex();
    }
#endif /* DEBUG_COMPILE */

```

1430.

```

⟨ Define verify_certificate 1428 ⟩ +≡
/* Up to here the process is the same for X.509 certificates and * OpenPGP keys. From now on X.509
   certificates are assumed. This can * be easily extended to work with openpgp keys as well. */
if (gnutls_certificate_type_get(session) ≠ GNUTLS_CRT_X509) {
    lock_cerr_mutex();
    fprintf(stderr, "gnutls_certificate_type_get(session)' != GNUTLS_CRT_X509\n");
    fprintf(stderr, "*** Exiting 'verify_certificate' unsuccessfully with return value 1.\n");
    unlock_cerr_mutex();
    return 1;
}
if (gnutls_x509_crt_init(&cert) < 0) {
    lock_cerr_mutex();
    fprintf(stderr, "ERROR In 'verify_certificate': Error in initialization\n");
    fprintf(stderr, "*** Exiting 'verify_certificate' unsuccessfully with return value 1.\n");
    unlock_cerr_mutex();
    return 1;
}
cert_list = gnutls_certificate_get_peers(session, &cert_list_size);
if (cert_list ≡ Λ) {
    lock_cerr_mutex();
    fprintf(stderr, "ERROR In 'verify_certificate': No certificate was found!\n");
    fprintf(stderr, "*** Exiting 'verify_certificate' unsuccessfully with return value 1.\n");
    unlock_cerr_mutex();
    gnutls_x509_crt_deinit(cert);
    return 1;
}
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        fprintf(stderr, "In 'verify_certificate': Certificate was found.\n");
        unlock_cerr_mutex();
    }
#endif /* DEBUG_COMPILE */
if (gnutls_x509_crt_import(cert, &cert_list[0], GNUTLS_X509_FMT_DER) < 0) {
    lock_cerr_mutex();
    fprintf(stderr, "ERROR In 'verify_certificate': error parsing certificate\n");
    fprintf(stderr, "*** Exiting 'verify_certificate' unsuccessfully with return value 1.\n");
    unlock_cerr_mutex();
    gnutls_x509_crt_deinit(cert);
    return 1;
}
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        fprintf(stderr, "Parsed certificate successfully\n");
        unlock_cerr_mutex();
    }
#endif /* DEBUG_COMPILE */

```

1431.

```
( Define verify_certificate 1428 ) +≡
  time_t temp_val = gnutls_x509_crt_get_expiration_time(cert);
  if (temp_val ≡ static_cast<time_t>(-1)) {
    lock_cerr_mutex();
    cerr ≪ "ERROR! In 'verify_certificate': gnutls_x509_crt_get_expiration_time" ≪
      "failed, returning (time_t)-1." ≪ endl ≪
      "Exiting 'verify_certificate' unsuccessfully with return value 1." ≪ endl;
    unlock_cerr_mutex();
    gnutls_x509_crt_deinit(cert);
    return 1;
} /* if (temp_val ≡ static_cast<time_t>(-1)) */ /* !! TODO: LDF 2013.05.15. Maybe return
  different values, depending on what error occurs. In particular, maybe return different values if
  cert. is expired or not yet activated. Possibly, I could pass an argument to store this info. */
else if (temp_val < time(0)) {
  lock_cerr_mutex();
  cerr ≪ "WARNING! In 'verify_certificate': Certificate expired." ≪ endl ≪
    "Exiting function unsuccessfully with return value 1." ≪ endl;
  unlock_cerr_mutex();
  gnutls_x509_crt_deinit(cert);
  return 1;
}
#ifndef DEBUG_COMPILE
else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "The certificate hasn't expired." ≪ endl;
    unlock_cerr_mutex();
  }
#endif /* DEBUG_COMPILE */
```

1432.

```

⟨ Define verify_certificate 1428 ⟩ +≡
temp_val = gnutls_x509_crt_get_activation_time(cert);
if (temp_val == static_cast<time_t>(-1)) {
    lock_cerr_mutex();
    cerr << "ERROR! In verify_certificate": gnutls_x509_crt_get_activation_time' <<
        "failed, returning '(time_t)-1.'." << endl <<
        "Exiting verify_certificate' unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    gnutls_x509_crt_deinit(cert);
    return 1;
} /* if (temp_val == static_cast<time_t>(-1)) */
else if (temp_val > time(0)) {
    lock_cerr_mutex();
    cerr << "WARNING! In verify_certificate': Certificate not activated yet." << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    gnutls_x509_crt_deinit(cert);
    return 1;
}
#endif /* DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "The certificate has been activated." << endl;
        unlock_cerr_mutex();
    }
#endif /* DEBUG_COMPILE */

```

1433. !! TODO: [LDF 2009.12.08.] Maybe use this to test whether the owner of the server's certificate is who we think it is. [LDF 2009.12.08.]

Log

[LDF 2009.12.08.] Commented-out. Later, this code may be useful. See “TODO” note above.

```

⟨ Define verify_certificate 1428 ⟩ +≡
#ifndef 0
    if (hostname & !gnutls_x509_crt_check_hostname(cert, hostname)) {
        lock_cerr_mutex();
        fprintf(stderr, "The certificate's owner does not match hostname '%s'\n", hostname);
        fprintf(stderr, "*** Exiting verify_certificate' unsuccessfully with return value 1.\n");
        unlock_cerr_mutex();
        gnutls_x509_crt_deinit(cert);
        return 1;
    }
    else {
        lock_cerr_mutex();
        fprintf(stderr, "The certificate's owner matches hostname '%s'\n", hostname);
        unlock_cerr_mutex();
    }
#endif

```

1434.**Log**

[LDF 2013.05.08.] Added this section.

```
< Define verify_certificate 1428 > +≡
if (x509_cert_ptr ≠ 0) { char serial[40];
size_t size;
size = sizeof (serial);
status = gnutls_x509_crt_get_serial(cert, serial, &size);
if (status ≠ GNUTLS_E_SUCCESS) {
    lock_cerr_mutex();
    cerr ≪ "ERROR! In 'verify_certificate': " ≪ "gnutls_x509_crt_ge\
t_serial' failed, returning " ≪ status ≪ ":" ≪ endl ≪ gnutls_strerror(status) ≪
endl ≪ "Failed to obtain serial number of user certificate." ≪ endl ≪
"Exiting function unsuccessfully with return value 1." ≪ endl;
unlock_cerr_mutex();
gnutls_x509_crt_deinit(cert);
return 1;
} /* if (status ≠ GNUTLS_E_SUCCESS) */
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "In 'verify_certificate': " ≪ "gnutls_x509_crt_get_serial returned " ≪
        status ≪ "." ≪ endl ≪ "Subject certificate serial number: " ≪ bin2hex(serial,
        size) ≪ endl ≪ "'size' == " ≪ size ≪ endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1435.

```

⟨ Define verify_certificate 1428 ⟩ +≡
unsigned long int temp_uint;
errno = 0;
temp_uint = strtoul(bin2hex(serial, size), 0, 16);
if (temp_uint ≡ ULONG_MAX) {
    lock_cerr_mutex();
    cerr ≪ "ERROR! In 'verify_certificate': "
        ≪ "'strtoul' failed, returning ULONG_MAX:'" ≪ endl ≪ strerror(errno) ≪ endl ≪
        "'Failed to convert serial number of user certificate to unsigned long integer.'"
        ≪ endl ≪ "Exiting function unsuccessfully with return value 1." ≪ endl;
    unlock_cerr_mutex();
    gnutls_x509_crt_deinit(cert);
    return 1;
} /* if (temp_uint ≡ ULONG_MAX) */
x509_cert_ptr->serialNumber = temp_uint;
#if DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ "In 'verify_certificate_chain': "
            ≪ 'x509_cert_ptr->serialNumber' ≡ hex" ≪ hex ≪
            x509_cert_ptr->serialNumber ≪ " "
            ≪ "decimal" ≪ dec ≪ x509_cert_ptr->serialNumber ≪ endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1436. Get validity dates for *x509_cert_ptr*. [LDF 2013.05.10.]

```

⟨ Define verify_certificate 1428 ⟩ +≡
time_t t;
char outstr[200];
struct tm tmp;
struct tm *tmp_ptr = &tmp;

```

1437. Activation. (*Validity_notBefore*) for *x509_cert_ptr*. [LDF 2013.05.10.]

```

⟨ Define verify_certificate 1428 ⟩ +≡
  t = gnutls_x509_crt_get_activation_time(cert);
  if (t ≡ static_cast<time_t>(-1)) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'verify_certificate': " << endl <<
      " 'gnutls_x509_crt_get_activation_time' failed, returning '(time_t)-1'." << endl <<
      "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    gnutls_x509_crt_deinit(cert);
    return 1;
  } /* if (t ≡ static_cast<time_t>(-1)) */
x509_cert_ptr->Validity_notBefore = t;
errno = 0;
gmtime_r(&t, tmp_ptr);
if (tmp_ptr ≡ 0) {
  lock_cerr_mutex();
  cerr << "ERROR! In 'verify_certificate': 'gmtime_r' failed, "
      "returning NULL: " << endl << strerror(errno) << endl <<
      "Exiting function unsuccessfully with return value 1." << endl;
  unlock_cerr_mutex();
  gnutls_x509_crt_deinit(cert);
  return 1;
} /* if (tmp_ptr ≡ 0) */
if (strftime(outstr, sizeof(outstr), "%Y-%m-%d %H:%M:%S UTC", tmp_ptr) ≡ 0) {
  lock_cerr_mutex();
  cerr << "ERROR! In 'verify_certificate': 'strftime' returned 0." << endl <<
      "Exiting function unsuccessfully with return value 1." << endl;
  unlock_cerr_mutex();
  gnutls_x509_crt_deinit(cert);
  return 1;
}
#endif DEBUG_COMPILE
else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'verify_certificate': Activation time is \""
        << outstr << "\" " << endl;
    unlock_cerr_mutex();
  }
#endif /* DEBUG_COMPILE */

```

1438. Expiration. (*Validity_notAfter*) for *user_cert*. [LDF 2013.05.10.]

```

⟨ Define verify_certificate 1428 ⟩ +≡
tmp_ptr = &tmp;
t = gnutls_x509_crt_get_expiration_time(cert);
if (t ≡ static_cast<time_t>(-1)) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'verify_certificate_chain':"
        << endl <<
        "'gnutls_x509_crt_get_expiration_time' failed, returning '(time_t)-1'."
        << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    gnutls_x509_crt_deinit(cert);
    return 1;
}
x509_cert_ptr->Validity_notAfter = t;
errno = 0;
gmtime_r(&t, tmp_ptr);
if (tmp_ptr ≡ 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'verify_certificate': 'gmtime_r' failed,"
        << "returning NULL:"
        << endl << strerror(errno) << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    gnutls_x509_crt_deinit(cert);
    return 1;
} /* if (tmp_ptr ≡ 0) */
if (strftime(outstr, sizeof(outstr), "%Y-%m-%d %H:%M:%S UTC", tmp_ptr) ≡ 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'verify_certificate': 'strftime' returned 0."
        << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    gnutls_x509_crt_deinit(cert);
    return 1;
}
#endif /* DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'verify_certificate':Expiration_time is \""
        << outstr << "\""
        << endl;
    unlock_cerr_mutex();
}
#endif /* DEBUG_COMPILE */

```

1439.

```

⟨ Define verify_certificate 1428 ⟩ +≡
  status = extract_dn_fields(cert, x509_cert_ptr);
  if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ "ERROR! In 'verify_certificate': 'extract_dn_fields' failed, returning" ≪
      status ≪ "." ≪ endl ≪ "Exiting function unsuccessfully with return value 1." ≪ endl;
    unlock_cerr_mutex();
    gnutls_x509_crt_deinit(cert);
    return 1;
  } /* if (status ≠ 0) */
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr ≪ "In 'verify_certificate': 'extract_dn_fields' succeeded, returning 0." ≪ endl;
      x509_cert_ptr_show();
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (x509_cert_ptr ≠ 0) */

```

1440.

```

⟨ Define verify_certificate 1428 ⟩ +≡
  gnutls_x509_crt_deinit(cert);
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    fprintf(stderr, "Verification succeeded\n");
    fprintf(stderr, "*** Exiting 'verify_certificate' successfully with return value 0.\n");
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  return 0; } /* End of verify_certificate definition */

```

1441. This is what's compiled. [LDF 2009.11.30.]

```

⟨ Include files 3 ⟩
using namespace std;
using namespace gwrdifpk;
⟨ Declare verify_certificate 1427 ⟩
⟨ Define verify_certificate 1428 ⟩

```

1442. This is what's written to the header file `ex_rfc2818.h`.

```

⟨ ex_rfc2818.h 1442 ⟩ ≡
#ifndef EX_RFC2818_H
#define EX_RFC2818_H 1
⟨ Declare verify_certificate 1427 ⟩
#endif

```

1443. Connect function for server. [LDF 2012.11.21.]

1444. Include files.

```
<Include files 3> +≡
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <pwd.h>
#include <errno.h>
#include <grp.h>
#include <sys/wait.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>
#include <time.h>
#include <sys/stat.h>
#include <string>
#include <iomanip>
#include <iostream>
#include <stdarg.h>
#include <limits.h>
#include <algorithm>
#include <fstream>
#include <iterator>
#include <time.h>
#include <math.h>
#include <sstream>
#include <map>
#include <vector>
#include <deque>
#include <stack>
#include <set>
#include <pthread.h>
#include <gcrypt.h> /* for gcry-control */
#include <gnutls/gnutls.h>
#include <gnutls/x509.h>
#if HAVE_CONFIG_H
#include "config.h"
#endif
#include <mysql.h>
#include <expat.h>
#undef NAME_LEN
#undef LOCAL_HOST
#include "glblcnst.h++"
#include "glblvrbl.h++"
#include "excptntp.h++"
#include "utilfncts.h++"
#include "grouptp.h++"
#include "hdlvltp.h++"
#include "irdsavtp.h++"
#include "parser.h++"
#include "scanner.h++"
```

```
#include "rspnstp.h++"
#include "irdsobtp.h++"
#include "hdltype.h++"
#include "dcmtddtp.h++"
#include "x509cert.h++"
#include "dstngnmt.h++"
#include "scprpmtp.h++"
#include "exchncli.h++"
```

1445. Connection function. (*connect_func*). [LDF 2012.07.02.]

[LDF 2013.03.21.] !! TODO: This function doesn't do anything significant other than call *exchange_data_with_client*. Consider whether it may be useful for some purpose in the future, or whether the latter could be modified and called as a thread function in the “listen” functions.

```
< connect_func declaration 1445 > ≡
void *connect_func(void *v);
```

This code is used in sections 1457 and 1458.

1446.

```
< connect_func definition 1446 > ≡
void *connect_func(void *v){ bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    string error_warning_str;
    stringstream temp_strm;
    char buffer[BUFFER_SIZE];
    memset(buffer, 0, BUFFER_SIZE);
    int status;
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Entering 'connect_func'." << endl << flush;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
}
```

See also sections 1447, 1448, 1449, 1450, 1451, 1452, and 1453.

This code is used in section 1457.

1447.

```
< connect_func definition 1446 > +≡
Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type *>(v);
param->PARSER_DEBUG = false; /* true */
#ifndef 0
if (DEBUG) {
    lock_cerr_mutex();
    param->show("param");
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif
pthread_cleanup_push(cleanup_handler, 0);
```

1448.

```
< connect_func definition 1446 > +≡
status = exchange_data_with_client(*param);
{
    int temp_width = 0;
    int i = (param→errors_occurred > param→warnings_occurred) ? param→errors_occurred :
        param→warnings_occurred;
    do {
        ++temp_width;
        i /= 10;
    } while (i > 0);
    temp_strm.str("");
    temp_strm << " " << setw(temp_width) << param→errors_occurred << "error" <<
        ((param→errors_occurred == 1) ? "" : "s") << " ." << endl << " " << setw(temp_width) <<
        param→warnings_occurred << "warning" << ((param→warnings_occurred == 1) ? "" : "s") <<
        " ." << endl;
    error_warning_str = temp_strm.str();
    temp_strm.str("");
}
```

1449.**Log**

[LDF 2013.04.18.] Added this section.

```
< connect_func definition 1446 > +≡
if (status == 2) { pthread_cleanup_push(cleanup_handler, 0);
lock_cerr_mutex();
cerr << "In 'connect_func': 'exchange_data_with_client' succeeded, returning 2:" <<
endl << "'END_SERVER' command succeeded." << endl <<
"Will exit gwiridsif with exit status 0." << endl << error_warning_str;
unlock_cerr_mutex();
pthread_mutex_lock(&thread_ctr_id_map_mutex);
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "Before erase: thread_ctr_id_map.size() == " << thread_ctr_id_map.size() << endl;
    for (map<unsigned int, pthread_t>::iterator iter = thread_ctr_id_map.begin();
         iter != thread_ctr_id_map.end(); ++iter)
        cerr << "iter->first == " << iter->first << endl << "iter->second == " << iter->second << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1450. The current thread may exit before *finish* has a chance to call *pthread_cancel* on it. In this case, the latter will fail, because the thread will no longer exist. This would not be a problem, however. On the other hand, if *finish* exits and the process ends before the current thread reaches the call *pthread_exit* below, there's no harm done either, because this code doesn't do anything significant between the calls to *pthread_kill* and *pthread_exit*. Therefore, we might as well remove the thread counter and ID (i.e., *param-thread_ctr* and *pthread_self()*, respectively) from *thread_ctr_id_map* and *thread_id_ctr_map* so that we don't get the error message if *pthread_cancel* should fail in *finish*. [LDF 2013.04.23.]

```

⟨ connect_func definition 1446 ⟩ +≡
    pthread_t save_thread_id = thread_ctr_id_map[param→thread_ctr];
    status = thread_ctr_id_map.erase(param→thread_ctr);
    if (status == 1) status = thread_id_ctr_map.erase(pthread_self());
    if (status != 1) {
        lock_cerr_mutex();
        cerr << "ERROR! In 'connect_func': map<unsigned_int, pthread_t>::erase' failed" <<
            "map<pthread_t, unsigned_int>::erase' failed, returning" << status << endl <<
            "Failed to erase entries for thread counter and thread ID for current thread" <<
            "from 'thread_ctr_id_map' and 'thread_id_ctr_map': " << endl <<
            "Current thread number: " << param→thread_ctr << endl << "Current thread ID: " <<
            thread_id_ctr_map[param→thread_ctr] << endl << "Will try to continue." << endl;
        unlock_cerr_mutex();
    } /* if (status != 1) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Removed entries for thread counter and ID for current thread" <<
            "(" << param→thread_ctr << ")" << "from 'thread_ctr_id_map' and 'thread_id\"
            "_ctr_map' successfully." << endl << "Current thread ID: " << save_thread_id << endl;
        for (map<unsigned int, pthread_t>::iterator iter = thread_ctr_id_map.begin();
             iter != thread_ctr_id_map.end(); ++iter)
            cerr << "iter->first==" << iter→first << endl << "iter->second==" << iter→second << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'connect_func': 'thread_ctr_id_map.size()' == " << thread_ctr_id_map.size() <<
            endl;
        if (thread_ctr_id_map.size() > 0) cerr << "'thread_ctr_id_map':" << endl;
        else cerr << "'thread_ctr_id_map' is empty. Not showing." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    pthread_mutex_unlock(&thread_ctr_id_map_mutex);

```

1451. Call *exit*. This is the only way to exit the program, since the threads that *connect_func* run in are detached. It would be possible to use a condition variable to announce that the program should exit, but there's no advantage to doing it that way. [LDF 2013.04.18.]

Call *pthread_cleanup_pop* with an argument > 1 to call *cleanup_handler*. Cleanup handlers are not automatically popped and executed when *exit* is called, so *pthread_cleanup_pop* must be called explicitly here, if we want a cleanup handler to be executed. However, as of this date, *cleanup_handler* doesn't do anything significant. [LDF 2013.04.18.]

```

⟨ connect_func definition 1446 ⟩ +≡
  delete param;
  param = 0;
  pthread_cleanup_pop(0); /* 1 */
  status = pthread_kill(thread_ctr_id_map[0], SIGINT);
  if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ "ERROR! In 'connect_func': 'pthread_kill' failed, returning " ≪ status ≪ ":" ≪
      endl ≪ strerror(status) ≪ endl ≪ "Failed to send SIGINT to main thread." ≪ endl ≪
      endl ≪ "Exiting gwirdsif unsuccessfully with exit status 1." ≪ endl;
    unlock_cerr_mutex();
    exit(1);
  } /* if (status ≠ 0) */
#endif /* DEBUG_COMPILE
else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "In 'connect_func': 'pthread_kill' succeeded, returning 0." ≪ endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
pthread_exit(0); } /* if (status ≡ 2) */

```

1452.

Log

[LDF 2013.04.23.] Added code for removing the entry for the current thread from `map<unsigned int, pthread_t> thread_ctr_id_map` and `map<pthread_t, unsigned int> thread_id_ctr_map`.

```

⟨ connect_func definition 1446 ⟩ +≡
else
  if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'connect_func': 'exchange_data_with_client' failed, returning" <<
           status << endl << error_warning_str <<
           "Exiting thread function unsuccessfully with exit status 0." << endl;
    unlock_cerr_mutex();
    pthread_mutex_lock(&thread_ctr_id_map_mutex);
    thread_ctr_id_map.erase(param - thread_ctr);
    thread_id_ctr_map.erase(pthread_self());
    pthread_mutex_unlock(&thread_ctr_id_map_mutex);
    delete param;
    param = 0;
    pthread_exit(0);
  } /* else if (status ≠ 0) */
#endif /* DEBUG_COMPILE
else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'connect_func': 'exchange_data_with_client' succeeded, returning 0." <<
          endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1453.**Log**

[LDF 2013.04.23.] Added code for removing the entry for the current thread from **map<unsigned int, pthread_t>** *thread_id_map* and **map<pthread_t, unsigned int>** *thread_id_ctr_map*.

```

⟨ connect_func definition 1446 ⟩ +≡
#if DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "Exiting thread function 'connect_func' successfully with return value 0." <<
        endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  lock_cerr_mutex();
  lock_cout_mutex();
  cout << "[Thread" << param->thread_ctr << "] Exiting thread:" << endl << error_warning_str << endl;
  unlock_cout_mutex();
  unlock_cerr_mutex(); /* !! TODO: LDF 2012.07.27. Don't delete param, if we keep it around in case
                        a session is resumed. */
  pthread_mutex_lock(&thread_id_map_mutex);
  thread_id_map.erase(param->thread_ctr);
  thread_id_ctr_map.erase(pthread_self());
  pthread_mutex_unlock(&thread_id_map_mutex);
  delete param;
  param = 0;
  pthread_cleanup_pop(0);
  pthread_exit(0); } /* End of connect_func definition */

```

1454. Cleanup handler. [LDF 2013.04.18.]**Log**

[LDF 2013.04.18.] Added this function.

```

⟨ cleanup_handler declaration 1454 ⟩ ≡
void cleanup_handler(void *arg);

```

This code is used in sections 1457 and 1458.

1455.

```

⟨ cleanup_handler definition 1455 ⟩ ≡
void cleanup_handler(void *arg)
{
    bool DEBUG = false;      /* true */
    set_debug_level(DEBUG, 0, 0);
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Entering 'cleanup_handler'." << endl;
        unlock_cerr_mutex();
    }      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Exiting 'cleanup_handler'." << endl;
        unlock_cerr_mutex();
    }      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */
    return;
}      /* End of cleanup_handler definition */

```

This code is used in section 1457.

1456. Putting connect together. [LDF 2012.06.26.]**1457.** This is what's compiled. [LDF 2012.06.26.]

```

⟨ Include files 3 ⟩
using namespace std;
using namespace gwrdifpk;
⟨ connect_func declaration 1445 ⟩
⟨ cleanup_handler declaration 1454 ⟩
⟨ connect_func definition 1446 ⟩
⟨ cleanup_handler definition 1455 ⟩

```

1458. This is what's written to the header file. [LDF 2012.07.02.]

```

⟨ connect.h 1458 ⟩ ≡
#ifndef CONNECT_H
#define CONNECT_H 1
#ifndef __GNUC_SOURCE
#define __GNUC_SOURCE
#endif
⟨ connect_func declaration 1445 ⟩
⟨ cleanup_handler declaration 1454 ⟩
#endif

```

1459. Listen function for local connections. [LDF 2012.09.19.]

1460. Include files.

```
<Include files 3> +≡
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <pwd.h>
#include <errno.h>
#include <grp.h>
#include <sys/wait.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>
#include <time.h>
#include <sys/stat.h>
#include <dirent.h>
#include <string>
#include <iomanip>
#include <iostream>
#include <iostream>
#include <stdarg.h>
#include <limits.h>
#include <algorithm>
#include <fstream>
#include <iterator>
#include <time.h>
#include <math.h>
#include <sstream>
#include <map>
#include <vector>
#include <deque>
#include <stack>
#include <set>
#include <pthread.h>
#if HAVE_CONFIG_H
#include "config.h"
#endif
#include <gcrypt.h> /* for gcry_control */
#include <gnutls/gnutls.h>
#include <gnutls/x509.h>
#include <mysql.h>
#include <expat.h>
#undef NAME_LEN
#undef LOCAL_HOST
#include "glblcnst.h++"
#include "glblvrbl.h++"
#include "excptntp.h++"
#include "utilfncts.h++"
#include "grouptp.h++"
#include "hndlvltp.h++"
#include "irdsavtp.h++"
#include "parser.h++"
```

```
#include "scanner.h++"
#include "rspnstp.h++"
#include "irdsobtp.h++"
#include "hndltype.h++"
#include "dcmtddtp.h++"
#include "x509cert.h++"
#include "dstngnmt.h++"
#include "scprpmtp.h++"
#include "connect.h++"
```

1461. *listen_local*. [LDF 2012.09.19.]**Log**

[LDF 2012.09.19.] Added this function. It contains code formerly in *main* of the server program **gwirldsif**.

```
{ listen_local declaration 1461 } ≡
void *listen_local(void *v);
```

This code is used in sections 1476 and 1477.

1462.

```
{ listen_local definition 1462 } ≡
void *listen_local(void *v){ bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status = 0;
    int s;
    int listen_local_thread_ctr = *static_cast<int *>(v);
```

See also sections 1463, 1464, 1465, 1466, 1467, 1468, 1469, 1470, 1471, 1472, and 1473.

This code is used in section 1476.

1463. Get name for socket and delete, if present. [LDF 2012.07.02.]

We just try deleting it instead of testing whether it exists. If it doesn't exist, we ignore the error. [LDF 2012.07.03.]

```
<listen_local definition 1462> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[gwiridsif] [Thread" << listen_local_thread_ctr <<
            "] In 'listen_local': socket_path == " << socket_path << endl;
        unlock_cerr_mutex();
    }
#endif /* DEBUG_COMPILE */
    errno = 0;
    status = unlink(socket_path.c_str());
    if (status == -1) {
        if (errno == ENOENT) {
#ifndef DEBUG_COMPILE
            if (DEBUG) {
                lock_cerr_mutex();
                cerr << "[gwiridsif] [Thread" << listen_local_thread_ctr <<
                    "] In 'listen_local': File " << socket_path << "' doesn't exist." <<
                    "Continuing." << endl;
                unlock_cerr_mutex();
            }
#endif /* DEBUG_COMPILE */
        }
    }
    else {
        lock_cerr_mutex();
        cerr << "[gwiridsif] [Thread" << listen_local_thread_ctr <<
            "] ERROR! In 'listen_local': 'unlink' failed to delete " <<
            socket_path << ", returning -1." << endl << "unlink_error: " << strerror(errno) <<
            endl << "Exiting thread function 'listen_local' unsuccessfully" <<
            "with exit status 0." << endl;
        unlock_cerr_mutex();
        pthread_exit(0);
    } /* else */
} /* if (status == -1) */
#endif /* DEBUG_COMPILE */
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[gwiridsif] [Thread" << listen_local_thread_ctr <<
            "] In 'listen_local': 'unlink' succeeded." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1464. Get socket. [LDF 2011.07.14.]

```

⟨ listen_local definition 1462 ⟩ +≡
int t;
int len;
pthread_attr_t attr;
pthread_attr_init(&attr);
pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);
struct sockaddr_un local;
struct sockaddr_un remote;
errno = 0;
s = socket(AF_UNIX, SOCK_STREAM, 0);
if (s ≡ -1) {
    lock_cerr_mutex();
    cerr << "[gwirdsif] [Thread" << listen_local_thread_ctr <<
        "] [ERROR! In 'listen_local': socket failed, returning -1]" <<
        endl << "socket_error:" << strerror(errno) << endl <<
        "Exiting thread function 'listen_local' unsuccessfully" << "with exit status 0." <<
        endl;
    unlock_cerr_mutex();
    pthread_exit(0);
} /* if (s ≡ -1) */
#endif DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[gwirdsif] [Thread" << listen_local_thread_ctr <<
        "] In 'listen_local': socket succeeded." << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
local.sun_family = AF_UNIX;
strcpy(local.sun_path, socket_path.c_str());
len = strlen(local.sun_path) + sizeof (local.sun_family);
errno = 0;

```

1465. Bind socket. [LDF 2012.07.02.]

```

⟨ listen_local definition 1462 ⟩ +≡
errno = 0; status = bind (s, reinterpret_cast <sockaddr *> (&local), len ) ;
if (s ≡ -1) {
    lock_cerr_mutex();
    cerr << "[gwirdsif] [Thread]" << listen_local_thread_ctr <<
        "] ERROR! In 'listen_local': 'bind' failed, returning -1:" <<
        endl << "bind_error:" << strerror(errno) << endl <<
        "Exiting thread function 'listen_local' unsuccessfully" << "with exit status 0." <<
        endl;
    unlock_cerr_mutex();
    close(s);
    pthread_exit(0);
} /* if (s ≡ -1) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[gwirdsif] [Thread]" << listen_local_thread_ctr <<
            "] In 'listen_local': 'bind' succeeded." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1466.

Log

[LDF 2012.01.31.] Added this section.

```

⟨ listen_local definition 1462 ⟩ +≡
status = chmod(socket_path.c_str(),
    S_IRUSR | S_IWUSR | S_IXUSR | S_IRGRP | S_IWGRP | S_IXGRP | S_IROTH | S_IWOTH | S_IXOTH);
if (status ≡ -1) {
    lock_cerr_mutex();
    cerr << "[gwirdsif] [Thread]" << listen_local_thread_ctr <<
        "] ERROR! In 'listen_local': 'chmod' failed, returning -1:" <<
        endl << "chmod_error:" << strerror(errno) << endl <<
        "Exiting thread function 'listen_local' unsuccessfully" << "with exit status 0." <<
        endl;
    unlock_cerr_mutex();
    close(s);
    pthread_exit(0);
}

```

1467. Listen. [LDF 2012.07.02.]

```

⟨ listen_local definition 1462 ⟩ +≡
status = listen(s, 128);
if (status ≡ -1) {
    lock_cerr_mutex();
    cerr << "[gwirdsif] [Thread" << listen_local_thread_ctr <<
        "] In 'listen_local': listen failed, returning -1:" << endl << "listen_error:" <<
        strerror(errno) << endl << "Exiting thread function 'listen_local' unsuccessfully" <<
        "with exit status 0." << endl;
    unlock_cerr_mutex();
    close(s);
    pthread_exit(0);
}
#endif DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[gwirdsif] [Thread" << listen_local_thread_ctr <<
        "] In 'listen_local': listen succeeded." << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1468. Main accept loop. [LDF 2012.07.02.]

```

⟨ listen_local definition 1462 ⟩ +≡
stringstream temp_strm;
temp_strm << "[Thread" << listen_local_thread_ctr << "]"
<< "In 'listen_local': Server ready. Listening to Unix domain socket"
<< " " <<
socket_path << endl;
lock_cout_mutex();
lock_cerr_mutex();
cout << temp_strm.str();
unlock_cerr_mutex();
unlock_cout_mutex();
temp_strm.str("");
Scan_Parse_Parameter_Type *param = 0;
pthread_t id = 0UL; for ( ; ; ) {

```

1469.

Log

[LDF 2012.07.13.] Added **try ... catch** block.

```

⟨ listen_local definition 1462 ⟩ +≡
param = 0;
id = 0UL;
try {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[gwiridsif][Thread]" << listen_local_thread_ctr <<
            "] [Thread]" << listen_local_thread_ctr << "] " <<
            "In 'listen_local': Creating new 'Scan_Par
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    param = new Scan_Par
    param->connection_type = Scan_Par
    } /* try */
catch(Initialize_Exception_Type) /* Currently, the Scan_Par
    doesn't throw any other exceptions. [LDF 2012.07.13.] */
{
    lock_cerr_mutex();
    cerr << "[gwiridsif][Thread]" << listen_local_thread_ctr <<
        "] ERROR! In 'listen_local': " << "The 'Scan_Par
        "failed, throwing exception 'Initialize_Exception_Type'." << endl <<
        "Failed to create 'Scan_Par
        "Exiting thread function 'listen_local' unsuccessfully" << "with exit status 0." <<
        endl;
    unlock_cerr_mutex();
    close(s);
    pthread_exit(0);
} /* catch */
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[gwiridsif][Thread]" << listen_local_thread_ctr << "] In 'listen_local': "
            "Finished creating new 'Scan_Par
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1470.

```

⟨ listen_local definition 1462 ⟩ +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[gwirdsif][Thread]" << listen_local_thread_ctr <<
            "]In['listen_local']:Waiting_for_a_connection...\n";
        unlock_cerr_mutex();
    }
#endif /* DEBUG_COMPILE */
t = sizeof (remote);
int sock; sock = accept (s,(struct sockaddr *) &remote, (socklen_t *) &t );
lock_cout_mutex();
lock_cerr_mutex();
cout << "[gwirdsif][Thread]" << listen_local_thread_ctr <<
    "]In['listen_local']:New_local_connection_through_Unix_Domain_socket" << " "
    << socket_path << endl;
unlock_cerr_mutex();
unlock_cout_mutex();
if (sock == -1) {
    lock_cerr_mutex();
    cerr << "[gwirdsif][Thread]" << listen_local_thread_ctr <<
        "]ERROR!In['listen_local']:'accept' failed, returning -1:" << endl <<
        "accept_error:" << strerror(errno) << endl << "Can't connect. Will try to continue." <<
        endl;
    unlock_cerr_mutex();
    continue;
} /* if (sock == -1) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[gwirdsif][Thread]" << listen_local_thread_ctr <<
            "]In['listen_local']:'accept' succeeded." << "'sock'" << sock << "."
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[gwirdsif][Thread]" << listen_local_thread_ctr <<
            "]In['listen_local']:'Connected.' << endl;
        unlock_cerr_mutex();
    }
#endif /* DEBUG_COMPILE */
param→sock = sock;

```

1471.

Log

[LDF 2013.05.23.] Added code for setting *global_thread_ctr* after it has wrapped around. This should never happen, unless **gwirdsif** runs for a *long* time without being stopped.

```
⟨ listen_local definition 1462 ⟩ +≡
  pthread_mutex_lock(&thread_ctr_mutex);
  param→thread_ctr = 0;
  if (global_thread_ctr_wrapped_around ≡ false) {
    param→thread_ctr = ++global_thread_ctr;
    if (global_thread_ctr ≡ UINT_MAX) {
      global_thread_ctr_wrapped_around ≡ true;
      global_thread_ctr = save_global_thread_ctr;
    }
  }
```

1472.

```

⟨ listen_local definition 1462 ⟩ +≡
else {
    map<unsigned int, pthread_t>::iterator iter;
    for (int i = global_thread_ctr; i ≤ UINT_MAX; ++i) {
        iter = thread_ctr_id_map.find(i);
        if (iter ≡ thread_ctr_id_map.end()) {
            param->thread_ctr = i;
            if (i ≡ UINT_MAX) global_thread_ctr = save_global_thread_ctr;
            else global_thread_ctr = i + i;
            break;
        }
    } /* for */
if (param->thread_ctr ≡ 0) { /* Try again once more from the beginning. [LDF 2013.05.23.] */
    for (unsigned int i = save_global_thread_ctr; i ≤ UINT_MAX; ++i) {
        iter = thread_ctr_id_map.find(i);
        if (iter ≡ thread_ctr_id_map.end()) {
            param->thread_ctr = i;
            if (i ≡ UINT_MAX) global_thread_ctr = save_global_thread_ctr;
            else global_thread_ctr = i + i;
            break;
        }
    }
} /* if */
if (param->thread_ctr ≡ 0) {
    /* If we still couldn't set param->thread_ctr, we quit. [LDF 2013.05.23.] */
    lock_cerr_mutex();
    cerr << "[gwirdsif] [Thread" << listen_local_thread_ctr <<
        "] [ERROR!] In 'listen_local': Failed to set 'param->thread_ctr'." <<
        endl << "Exiting thread unsuccessfully with return value 0." << endl;
    unlock_cerr_mutex();
    pthread_mutex_unlock(&thread_ctr_mutex);
    delete param;
    param = 0;
    close(s);
    pthread_exit(0);
}
/* else */
pthread_mutex_unlock(&thread_ctr_mutex);

```

1473.

```

⟨ listen_local definition 1462 ⟩ +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[gwirdsif] [Thread" << listen_local_thread_ctr << "]"
            << "In 'listen_local':"
            << "Calling 'connect_func' via 'pthread_create'."
            << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    status = pthread_create(&id, &attr, /* Attribute */
                           connect_func, static_cast<void *>(param));
    thread_ctr_id_map[param->thread_ctr] = id;
    thread_id_ctr_map[id] = param->thread_ctr;
#endif /* DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[gwirdsif] [Thread" << listen_local_thread_ctr << "]"
            << "In 'listen_local':"
            << "'pthread_create' returned"
            << status << "."
            << endl << "'thread_ctr_id_map.size()' =="
            << thread_ctr_id_map.size() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    } /* for */
    close(s);
    pthread_exit(0); /* Will normally never be reached. [LDF 2012.09.19.] */
} /* End of listen_local definition */

```

1474.

⟨ Garbage 303 ⟩ +≡

1475. Putting `listnlcl.web` together.

1476. This is what's compiled.

```

⟨ Include files 3 ⟩
using namespace std;
using namespace gwrdifpk;
⟨ listen_local declaration 1461 ⟩
⟨ listen_local definition 1462 ⟩
#ifndef 0
    ⟨ Garbage 303 ⟩
#endif

```

1477. This is what's written to the header file `listnlcl.h`. [LDF 2012.06.27.]

```
<listnlcl.h 1477>≡
#ifndef LISTNLCL_H
#define LISTNLCL_H 1
    using namespace std;
    using namespace gwrdifpk;
    <listen_local declaration 1461>
#endif
```

1478. Listen function for remote connections. [LDF 2012.09.19.]

1479. Include files.

```
<Include files 3> +≡
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <fcntl.h>
#include <math.h>
#include <pwd.h>
#include <grp.h>
#include <dirent.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/un.h>
#include <sys/wait.h>
#include <time.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <netinet/in.h>
#include <gcrypt.h> /* for gcry_control */
#include <gnutls/gnutls.h>
#include <gnutls/x509.h>
#include <expat.h>
#include <string>
#include <iomanip>
#include <ios>
#include <iostream>
#include <stdarg.h>
#include <limits.h>
#include <algorithm>
#include <fstream>
#include <iterator>
#include <time.h>
#include <math.h>
#include <sstream>
#include <map>
#include <vector>
#include <deque>
#include <stack>
#include <set>
#include <pthread.h>
#if HAVE_CONFIG_H
#include "config.h"
#endif
#include <mysql.h>
#undef NAME_LEN
#undef LOCAL_HOST
#include "glblcnst.h++"
#include "glblvrbl.h++"
#include "excptntp.h++"
```

```
#include "utilfncts.h++"
#include "grouppt.h++"
#include "hdlvltp.h++"
#include "irdsavtp.h++"
#include "parser.h++"
#include "scanner.h++"
#include "rspnstp.h++"
#include "irdsobtp.h++"
#include "hdltype.h++"
#include "dcmtdttp.h++"
#include "x509cert.h++"
#include "dstngnmt.h++"
#include "scprpmtp.h++"
#include "connect.h++"
```

1480. *listen_remote_anon*. [LDF 2012.09.19.]**Log**

[LDF 2012.09.19.] Added this function.

(listen_remote_anon declaration 1480) ≡
void **listen_remote_anon*(**void** **v*);

This code is used in sections 1494 and 1495.

1481.

(listen_remote_anon definition 1481) ≡
void **listen_remote_anon*(**void** **v*){ **bool** DEBUG = *false*; /* true */
 set_debug_level(DEBUG, 0, 0);
int status = 0;
int listen_remote_anon_thread_ctr = *static_cast<**int** *>(*v*);
#if DEBUG_COMPILE
 if (DEBUG) {
 lock_cerr_mutex();
 cerr << "[gwirdsif][Thread" << listen_remote_anon_thread_ctr << "] " <<
 "Entering 'listen_remote_anon'." << endl;
 unlock_cerr_mutex();
 } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

See also sections 1482, 1483, 1484, 1485, 1486, and 1487.

This code is used in section 1494.

1482.

```

⟨ listen_remote_anon definition 1481 ⟩ +≡
int err, listen_sd, i;
int ret;
struct sockaddr_in sa_serv;
struct sockaddr_in sa_cli;
int client_len;
char topbuf[512];
int optval = 1;
gnutls_anon_allocate_server_credentials(&anoncred);
generate_dh_params();
gnutls_anon_set_server_dh_params(anoncred, dh_params); /* Socket operations */
errno = 0;
listen_sd = socket(AF_INET, SOCK_STREAM, 0);
if (listen_sd == -1) {
    lock_cerr_mutex();
    cerr << "[gwiridsif][Thread" << listen_remote_anon_thread_ctr << "]"
        << endl << "ERROR! In 'listen_remote_anon': 'socket' failed, returning -1:" <<
        endl << strerror(errno) << endl << "Exiting thread function unsuccessfully\"
        with return value 0." << endl;
    unlock_cerr_mutex();
    pthread_exit(0);
}
memset(&sa_serv, '\0', sizeof(sa_serv));
sa_serv.sin_family = AF_INET;
sa_serv.sin_addr.s_addr = INADDR_ANY;
sa_serv.sin_port = htons(port_num_anon); /* Server Port number */
setsockopt(listen_sd, SOL_SOCKET, SO_REUSEADDR, &optval, sizeof(int));
errno = 0; status = bind(listen_sd, (SA *) &sa_serv, sizeof(sa_serv));
if (status == -1) {
    lock_cerr_mutex();
    cerr << "[gwiridsif][Thread" << listen_remote_anon_thread_ctr << "]"
        << endl << "ERROR! In 'listen_remote_anon': 'bind' failed, returning -1:" <<
        endl << strerror(errno) << endl << "Exiting thread function unsuccessfully\"
        with return value 0." << endl;
    unlock_cerr_mutex();
    close(listen_sd);
    pthread_exit(0);
}
errno = 0;
status = listen(listen_sd, 1024);
if (status == -1) {
    lock_cerr_mutex();
    cerr << "[gwiridsif][Thread" << listen_remote_anon_thread_ctr << "]"
        << endl << "ERROR! In 'listen_remote_anon': 'listen' failed, returning -1:" <<
        endl << strerror(errno) << endl << "Exiting thread function unsuccessfully\"
        with return value 0." << endl;
    unlock_cerr_mutex();
    close(listen_sd);
    pthread_exit(0);
}
#endif DEBUG_COMPILE

```

```
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[gwirdsif][Thread" << listen_remote_anon_thread_ctr << "]"
        << endl <<
        "In 'listen_remote_anon': Server ready. Listening to port"
        << port_num_anon <<
        "." << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
client_len = sizeof (sa_cli);
```

1483. Main accept loop. [LDF 2012.09.20.]

```

⟨ listen_remote_anon definition 1481 ⟩ +≡
  stringstream temp_strm;
  temp_strm << "[Thread]" << listen_remote_anon_thread_ctr << "]"
  << "In 'listen_remote_anon': Server ready. Listening to port" << port_num_anon <<
  "." << endl;
  lock_cout_mutex();
  lock_cerr_mutex();
  cout << temp_strm.str();
  unlock_cerr_mutex();
  unlock_cout_mutex();
  temp_strm.str("");
Scan_Parse_Parameter_Type *param = 0;
pthread_attr_t attr;
pthread_attr_init(&attr);
pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);
pthread_t id = 0UL; for ( ; ; ) { param = 0;
id = 0UL;
try {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[gwirdsif][Thread]" << listen_remote_anon_thread_ctr <<
    "] [Thread]" << listen_remote_anon_thread_ctr << "]"
    << "In 'listen_remote_anon': Creating new 'Scan_Parse_Parameter_Type' object."
    << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  param = new Scan_Parse_Parameter_Type;
  param->connection_type = Scan_Parse_Parameter_Type::ANON_AUTH_TYPE;
} /* try */
catch(Initialize_Exception_Type) /* Currently, the Scan_Parse_Parameter_Type constructor
  doesn't throw any other exceptions. [LDF 2012.07.13.] */
{
  lock_cerr_mutex();
  cerr << "[gwirdsif][Thread]" << listen_remote_anon_thread_ctr << "] ERROR! In 'listen"
  << "remote_anon': The 'Scan_Parse_Parameter_Type' constructor"
  << " failed, throwing exception 'Initialize_Exception_Type'." << endl <<
  "Failed to create 'Scan_Parse_Parameter_Type' object." << endl <<
  "Exiting thread function 'listen_remote_anon' unsuccessfully"
  << "with exit status 0." << endl;
  unlock_cerr_mutex();
  close(listen_sd);
  pthread_exit(0);
} /* catch */
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[gwirdsif][Thread]" << listen_remote_anon_thread_ctr << "] In 'listen_remote"
    << "anon':"
    << "Finished creating new 'Scan_Parse_Parameter_Type' object." << endl;
}

```

```

    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
param->session = initialize_tls_session();
/* BUG FIX: Added cast of &client_len argument to socklen_t *. The way it was before is only a bug
   when compiling with a C++ compiler. LDF 2010.02.03. */
#if DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[gwiridsif][Thread" << listen_remote_anon_thread_ctr <<
        "]In['listen_remote_anon']:Waiting for a connection...\n";
    unlock_cerr_mutex();
}
#endif /* DEBUG_COMPILE */
param->sock = accept (listen_sd, ( SA * ) &sa_cli , ( socklen_t * ) &client_len );
lock_cout_mutex();
lock_cerr_mutex();
cout << "[gwiridsif][Thread" << listen_remote_anon_thread_ctr << "]" << endl <<
    "In['listen_remote_anon']:Connection from " << inet_ntop(AF_INET,&sa_cli.sin_addr,
        topbuf,sizeof (topbuf)) << ",port" << ntohs(sa_cli.sin_port) << "." << endl;
unlock_cerr_mutex();
unlock_cout_mutex();
gnutls_transport_set_ptr(param->session,reinterpret_cast<gnutls_transport_ptr_t>(param->sock));
ret = gnutls_handshake(param->session);
if (ret < 0) {
    lock_cerr_mutex();
    cerr << "[gwiridsif][Thread" << listen_remote_anon_thread_ctr <<
        "]ERROR!In['listen_remote_anon']:" << endl <<
        "'gnutls_handshake' failed, returning" << ret << ":" << endl <<
        gnutls_strerror(ret) << endl << "Can't connect. Continuing." << endl;
    unlock_cerr_mutex();
    delete param;
    param = 0;
    continue;
}
#endif /* DEBUG_COMPILE */
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[gwiridsif][Thread" << listen_remote_anon_thread_ctr <<
        "]In['listen_remote_anon']:" << endl << "Handshake was completed." << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1484.

!! TODO: Test this! This will not be easy to test. Some maximum value much less than `UINT_MAX` will have to be used and multiple threads will have to be started and kept running. !! PLEASE NOTE: The other “listen” functions `listen_local` and `listen_remote_X_509` contain (nearly) identical code. It will have to be tested with all three functions! [LDF 2013.05.23.]

Log

[LDF 2013.05.23.] Added code for setting `global_thread_ctr` after it has wrapped around. This should never happen, unless `gwirdsif` runs for a *long* time without being stopped.

```
⟨ listen_remote_anon definition 1481 ⟩ +≡  
  pthread_mutex_lock(&thread_ctr_mutex);  
  param→thread_ctr = 0;  
  if (global_thread_ctr_wrapped_around ≡ false) {  
    param→thread_ctr = ++global_thread_ctr;  
    if (global_thread_ctr ≡ UINT_MAX) {  
      global_thread_ctr_wrapped_around ≡ true;  
      global_thread_ctr = save_global_thread_ctr;  
    }  
  }
```

1485.

```

⟨ listen_remote_anon definition 1481 ⟩ +≡
else {
    map<unsigned int, pthread_t>::iterator iter;
    for (int i = global_thread_ctr; i ≤ UINT_MAX; ++i) {
        iter = thread_ctr_id_map.find(i);
        if (iter ≡ thread_ctr_id_map.end()) {
            param→thread_ctr = i;
            if (i ≡ UINT_MAX) global_thread_ctr = save_global_thread_ctr;
            else global_thread_ctr = i + i;
            break;
        }
    } /* for */
if (param→thread_ctr ≡ 0) { /* Try again once more from the beginning. [LDF 2013.05.23.] */
    for (unsigned int i = save_global_thread_ctr; i ≤ UINT_MAX; ++i) {
        iter = thread_ctr_id_map.find(i);
        if (iter ≡ thread_ctr_id_map.end()) {
            param→thread_ctr = i;
            if (i ≡ UINT_MAX) global_thread_ctr = save_global_thread_ctr;
            else global_thread_ctr = i + i;
            break;
        }
    }
} /* if */
if (param→thread_ctr ≡ 0) {
    /* If we still couldn't set param→thread_ctr, we quit. [LDF 2013.05.23.] */
    lock_cerr_mutex();
    cerr << "[gwirdsif] [Thread" << listen_remote_anon_thread_ctr <<
        "] ERROR! In 'listen_remote_anon': Failed to set 'param->thread_ctr'." <<
        endl << "Exiting thread unsuccessfully with return value 0." << endl;
    unlock_cerr_mutex();
    pthread_mutex_unlock(&thread_ctr_mutex);
    delete param;
    param = 0;
    close(listen_sd);
    pthread_exit(0);
}
/* else */
pthread_mutex_unlock(&thread_ctr_mutex);

```

1486.

```
< listen_remote_anon definition 1481 > +≡
lock_cerr_mutex();
cerr << "[gwiridsif] [Thread" << listen_remote_anon_thread_ctr << "]"
     << "In 'listen_remote_anon': "
     << "Calling 'connect_func' via 'pthread_create'. "
     << endl;
unlock_cerr_mutex();
param→remote_connection = true;
status = pthread_create(&id, &attr,      /* Attribute */
                      connect_func, static_cast<void *>(param));
thread_ctr_id_map[param→thread_ctr] = id;
thread_id_ctr_map[id] = param→thread_ctr;
lock_cerr_mutex();
cerr << "[gwiridsif] [Thread" << listen_remote_anon_thread_ctr << "]"
     << "In 'listen_remote_anon': "
     << "'pthread_create' returned "
     << status << ". "
     << endl;
unlock_cerr_mutex(); } /* for */
```

1487. Exit. This code will never be reached in normal operation. [LDF 2012.09.20.]

```
< listen_remote_anon definition 1481 > +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[gwiridsif] [Thread"
             << listen_remote_anon_thread_ctr << "]"
             << "Exiting thread function 'listen_remote_anon' successfully with return value 0."
             << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    close(listen_sd); /* Will normally never be reached. [LDF 2012.09.20.] */
    pthread_exit(0); } /* End of listen_remote_anon definition */
```

1488.

```
< initialize_tls_session declaration 1488 > ≡
gnutls_session_t initialize_tls_session(void);
```

This code is used in sections 1494 and 1495.

1489.

```
< initialize_tls_session definition 1489 > ≡
gnutls_session_t initialize_tls_session(void)
{
    gnutls_session_t session;
    gnutls_init(&session, GNUTLS_SERVER);
    gnutls_priority_set_direct(session, "NORMAL:+ANON-DH", Λ);
    gnutls_credentials_set(session, GNUTLS_CRD_ANON, anoncred);
    gnutls_dh_set_prime_bits(session, DH_BITS);
    return session;
}
```

This code is used in section 1494.

1490.

```
<generate_dh_params declaration 1490> ≡
static int generate_dh_params(void);
```

This code is used in sections 1494 and 1495.

1491.

```
<generate_dh_params definition 1491> ≡
int generate_dh_params(void)
{ /* Generate Diffie Hellman parameters - for use with DHE * kx algorithms. These should be
   discarded and regenerated * once a day, once a week or once a month. Depending on the * security
   requirements. */
  gnutls_dh_params_init(&dh_params);
  gnutls_dh_params_generate2(dh_params, DH_BITS);
  return 0;
}
```

This code is used in section 1494.

1492.

```
<Garbage 303> +≡
```

1493. Putting `lstnrmta.web` together.

1494. This is what's compiled.

```
<Include files 3>
#define SAsstruct sockaddr
#define DH_BITS 1024
#define SOCKET_ERR(err,s)
  if (err ≡ -1) {
    perror(s);
    return (1);
}
using namespace std;
using namespace gwrdifpk;
extern gnutls_anon_server_credentials_t anoncred;
static gnutls_dh_params_t dh_params;
<generate_dh_params declaration 1490>
<initialize_tls_session declaration 1488>
<listen_remote_anon declaration 1480>
<generate_dh_params definition 1491>
<initialize_tls_session definition 1489>
<listen_remote_anon definition 1481>
#if 0
<Garbage 303>
#endif
```

1495. This is what's written to the header file `lstnrmta.h`. [LDF 2012.06.27.]

```
<lstnrmta.h 1495>≡
#ifndef LSTNRMTA_H
#define LSTNRMTA_H 1
    using namespace std;
    using namespace gwrdifpk;
    <generate_dh_params declaration 1490>
    <initialize_tls_session declaration 1488>
    <listen_remote_anon declaration 1480>
#endif
```

1496. Listen function for remote connections with X.509 authentication. [LDF 2012.09.21.]

1497. Include files.

```
<Include files 3> +≡
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <fcntl.h>
#include <math.h>
#include <pwd.h>
#include <grp.h>
#include <dirent.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/un.h>
#include <sys/wait.h>
#include <time.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <netinet/in.h>
#include <gcrypt.h> /* for gcry_control */
#include <gnutls/gnutls.h>
#include <gnutls/x509.h>
#include <string>
#include <iomanip>
#include <iostream>
#include <iostream>
#include <stdarg.h>
#include <limits.h>
#include <algorithm>
#include <fstream>
#include <iterator>
#include <time.h>
#include <math.h>
#include <sstream>
#include <map>
#include <vector>
#include <deque>
#include <stack>
#include <set>
#include <pthread.h>
#if HAVE_CONFIG_H
#include "config.h"
#endif
#include <mysql.h>
#include <expat.h>
#undef NAME_LEN
#undef LOCAL_HOST
#include "glblcnst.h++"
#include "glblvrbl.h++"
#include "excptntp.h++"
```

```
#include "utilfncts.h++"
#include "grouppt.h++"
#include "hdlvltp.h++"
#include "irdsavtp.h++"
#include "parser.h++"
#include "scanner.h++"
#include "rspnstp.h++"
#include "irdsobtp.h++"
#include "hdltype.h++"
#include "dcmtdttp.h++"
#include "x509cert.h++"
#include "dstngnmt.h++"
#include "scprpmtp.h++"
#include "connect.h++"
#include "gntlsfnc.h++"
#include "ex_rfc2818.h++"
#include "usrinftp.h++"
```

1498. *listen_remote_X_509*. [LDF 2012.09.21.]

Log

[LDF 2012.09.21.] Added this function.

⟨ listen_remote_X_509 declaration 1498 ⟩ ≡
void **listen_remote_X_509*(**void** **v*);

This code is used in sections 1512 and 1513.

1499.

⟨ listen_remote_X_509 definition 1499 ⟩ ≡
void **listen_remote_X_509*(**void** **v*) { **bool** DEBUG = *false*; /* true */
set_debug_level(DEBUG, 0, 0);
int status = 0;
char buffer[BUFFER_SIZE];
memset(buffer, 0, BUFFER_SIZE);
int *listen_remote_X_509_thread_ctr* = ***static_cast**<**int** *>(*v*);
#if DEBUG_COMPILE
if (DEBUG) {
lock_cerr_mutex();
cerr ≪ "[gwirldsif][Thread]" ≪ *listen_remote_X_509_thread_ctr* ≪ "]" ≪
 "Entering 'listen_remote_X_509'." ≪ *endl*;
unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

See also sections 1500, 1501, 1502, 1503, 1504, 1505, 1506, 1507, 1508, and 1509.

This code is used in section 1512.

1500.

```

⟨ listen_remote_X_509 definition 1499 ⟩ +≡
  DEFAULT_CERT_FILENAME = config_dir;
  DEFAULT_CERT_FILENAME += "/gwiridsif_cert.pem";
  DEFAULT_KEY_FILENAME = config_dir;
  DEFAULT_KEY_FILENAME += "/gwiridsif_key.pem";
  DEFAULT_CA_FILENAME = config_dir;
  DEFAULT_CA_FILENAME += "/ca_cert.pem";
  DEFAULT_CRL_FILENAME = "";
  string curr_key_filename = DEFAULT_KEY_FILENAME;
  string curr_cert_filename = DEFAULT_CERT_FILENAME;
  string curr_ca_filename = DEFAULT_CA_FILENAME;
  string curr_crl_filename = DEFAULT_CRL_FILENAME;
  pthread_attr_t attr;
  gnutls_certificate_credentials_t cert_cred;
  gnutls_x509_crt_t * ca_list;
  unsigned int ca_list_size;
  int err, listen_sd, i;
  int ret;
  struct sockaddr_in sa_serv;
  struct sockaddr_in sa_cli;
  int client_len;
  char topbuf[512];
  int optval = 1;
  Scan_Parse_Parameter_Type *param = 0; /* to disallow usage of the blocking /dev/random */
  gnutls_certificate_allocate_credentials(&cert_cred);
  status = gnutls_certificate_set_x509_trust_file(cert_cred, curr_ca_filename.c_str(), GNUTLS_X509_FMT_PEM);
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'listen_remote_X_509':" << endl << "'gnutls_certificate_set_x509_trust_file' "
        "returned" << status << "(the number of certificates processed)." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  if (curr_crl_filename != "") {
    status = gnutls_certificate_set_x509_crl_file(cert_cred, curr_crl_filename.c_str(), GNUTLS_X509_FMT_PEM);
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'listen_remote_X_509':" << endl <<
        "'gnutls_certificate_set_x509_crl_file' returned" << status <<
        "(the number of CRLs (certificate revocation lists) processed)." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  } /* if (curr_crl_filename != "") */
  status = gnutls_certificate_set_x509_key_file(cert_cred, curr_cert_filename.c_str(),
    curr_key_filename.c_str(), GNUTLS_X509_FMT_PEM);
#ifndef DEBUG_COMPILE
  if (DEBUG) {

```

```

lock_cerr_mutex();
cerr << "In 'listen_remote_X_509': " << endl <<
    "'gnutls_certificate_set_x509_key_file' returned" << status << endl;
unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
if (status != GNUTLS_E_SUCCESS) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'listen_remote_X_509': " << endl <<
        "'gnutls_certificate_set_x509_key_file' returned" << status << ":" << endl;
    gnutls_perror(status);
    unlock_cerr_mutex();
} /* if (status != GNUTLS_E_SUCCESS) */
#endif DEBUG_COMPILE
else
{
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'listen_remote_X_509': " << endl <<
            "'gnutls_certificate_set_x509_key_file' returned" GNUTLS_E_SUCCESS " << status <<
            ")" << endl;
        gnutls_perror(status);
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    gnutls_certificate_get_x509_cas(cert_cred, &ca_list, &ca_list_size);
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'listen_remote_X_509': " << "ca_list_size==" << ca_list_size << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    gnutls_dh_params_t dh_params;
    gnutls_rsa_params_t rsa_params;
    generate_dh_params(dh_params);
    generate_rsa_params(rsa_params);
    if (TLS_SESSION_CACHE != 0) {
        wrap_db_init();
    }
    gnutls_certificate_set_dh_params(cert_cred, dh_params);
    gnutls_certificate_set_rsa_export_params(cert_cred, rsa_params);
    errno = 0;
    listen_sd = socket(AF_INET, SOCK_STREAM, 0);
    if (listen_sd == -1) {
        lock_cerr_mutex();
        cerr << "[gwidifpk][Thread" << listen_remote_X_509_thread_ctr << "]" << endl <<
            "ERROR! In 'listen_remote_X_509': socket failed, returning -1: " <<
            endl << strerror(errno) << endl << "Exiting thread function unsuccessfully\n"
            "with return value 0." << endl;
        unlock_cerr_mutex();
        pthread_exit(0);
    }
}

```

```

memset(&sa_serv, '\0', sizeof(sa_serv));
sa_serv.sin_family = AF_INET;
sa_serv.sin_addr.s_addr = INADDR_ANY;
sa_serv.sin_port = htons(port_num_x_509); /* Server Port number, default 5557 */
setsockopt(listen_sd, SOL_SOCKET, SO_REUSEADDR, &optval, sizeof(int));
errno = 0; status = bind(listen_sd, (SA *) &sa_serv, sizeof(sa_serv));
if (status == -1) {
    lock_cerr_mutex();
    cerr << "[gwrdsif][Thread" << listen_remote_X_509_thread_ctr << "]"
        << endl << "ERROR! In 'listen_remote_X_509': 'bind' failed, returning -1."
        << endl << strerror(errno) << endl << "Exiting thread function unsuccessfully"
        << endl << "with return value 0." << endl;
    unlock_cerr_mutex();
    close(listen_sd);
    pthread_exit(0);
}
errno = 0;
status = listen(listen_sd, 1024);
if (status == -1) {
    lock_cerr_mutex();
    cerr << "[gwrdsif][Thread" << listen_remote_X_509_thread_ctr << "]"
        << endl << "ERROR! In 'listen_remote_X_509': 'listen' failed, returning -1."
        << endl << strerror(errno) << endl << "Exiting thread function unsuccessfully"
        << endl << "with return value 0." << endl;
    unlock_cerr_mutex();
    close(listen_sd);
    pthread_exit(0);
}
stringstream temp_strm;
temp_strm << "[Thread" << listen_remote_X_509_thread_ctr << "]"
    << endl << "In 'listen_remote_X_509': Server ready. Listening to port"
        << port_num_x_509 << endl << ".";
lock_cout_mutex();
lock_cerr_mutex();
cout << temp_strm.str();
unlock_cerr_mutex();
unlock_cout_mutex();
temp_strm.str("");
client_len = sizeof(sa_cli);
pthread_attr_init(&attr);
pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);

```

1501. Main accept loop. [LDF 2012.09.21.]

Log

[LDF 2013.05.15.] BUG FIX: Now deleting *param* and continuing the main accept loop when an error occurs. Previously, *param* was not deleted (memory leak!) and *pthread_exit* was called, which caused the “listen” thread to exit, which after a certain point was not what I wanted.

```

⟨ listen_remote_X_509 definition 1499 ⟩ +≡
  pthread_t id = 0UL; for ( ; ; ) { param = 0;
  id = 0UL;
  try {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[gwirldsif][Thread]" << listen_remote_X_509_thread_ctr << "] [Thread]" <<
      listen_remote_X_509_thread_ctr << "] " << "In['listen_remote_X_509']:" <<
      "Creating new 'Scan_Parse_Parameter_Type' object." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  param = new Scan_Parse_Parameter_Type;
  param->connection_type = Scan_Parse_Parameter_Type::X_509_AUTH_TYPE;
} /* try */ /* Currently, the Scan_Parse_Parameter_Type constructor doesn't throw any
   other exceptions. [LDF 2012.07.13.] */
catch(Initialize_Exception_Type)
{
  lock_cerr_mutex();
  cerr << "[gwirldsif][Thread]" << listen_remote_X_509_thread_ctr << "] ERROR! In['liste\
n_remote_X_509']:" << "The 'Scan_Parse_Parameter_Type' constructor" <<
  "failed, throwing exception 'Initialize_Exception_Type'." << endl <<
  "Failed to create 'Scan_Parse_Parameter_Type' object." << endl <<
  "Exiting thread function 'listen_remote_X_509' unsuccessfully" <<
  "with exit status 0." << endl;
  unlock_cerr_mutex();
  close(listen_sd);
  pthread_exit(0);
} /* catch */
#ifndef DEBUG_COMPILE
if (DEBUG) {
  lock_cerr_mutex();
  cerr << "[gwirldsif][Thread]" << listen_remote_X_509_thread_ctr << "] In['listen_remote\
_X_509']:" << "Finished creating new 'Scan_Parse_Parameter_Type' object." << endl;
  unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
param->session = initialize_tls_session(cert_cred); param->sock = accept (listen_sd, ( SA * ) &sa_cli , (
  socklen_t * ) &client_len );
lock_cout_mutex();
lock_cerr_mutex();
cout << "[gwirldsif][Thread]" << listen_remote_X_509_thread_ctr << "]" << endl <<
  "In['listen_remote_X_509']:" Connection from " << inet_ntop(AF_INET, &sa_cli.sin_addr,
  topbuf, sizeof (topbuf)) << ", port" << ntohs(sa_cli.sin_port) << endl;

```

```
unlock_cerr_mutex();
unlock_cout_mutex();
gnutls_transport_set_ptr(param->session, reinterpret_cast<gnutls_transport_ptr_t>(param->sock));
status = gnutls_handshake(param->session);
if (status < 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'listen_remote_X_509': *** Handshake has failed:" << endl;
    gnutls_strerror(status);
    cerr << endl << "Deinitializing 'param->session'. Continuing main accept loop." << endl;
    unlock_cerr_mutex();
    gnutls_deinit(param->session);
    delete param;
    param = 0;
    continue;
} /* if (status < 0) */
```

1502.**Log**

[LDF 2013.05.08.] Added this section.
 [LDF 2013.05.15.] Added code for sending error message to client.

```

⟨ listen_remote_X_509 definition 1499 ⟩ +≡
  status = verify_certificate(param->session, &param->user_cert);
  if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ "ERROR! In 'listen_remote_X_509': 'verify_certificate' failed, returning" ≪
      endl ≪ status ≪ "." ≪ endl ≪
      "Deinitializing 'param->session'. Continuing main accept loop." ≪ endl;
    unlock_cerr_mutex();
    memset(buffer, 0, BUFFER_SIZE);
    strcpy(buffer, "AUTHENTICATION_ERROR");
    gnutls_record_send(param->session, buffer, strlen(buffer));
    memset(buffer, 0, BUFFER_SIZE);
    close(param->sock);
    param->sock = 0;
    gnutls_deinit(param->session);
    delete param;
    param = 0;
    continue;
  } /* if (status ≠ 0) */
#endif DEBUG_COMPILE
else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "In 'listen_remote_X_509': 'verify_certificate' succeeded, returning 0." ≪
      endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
Distinguished_Name_Type distinguished_name;
distinguished_name = param->user_cert;
#endif DEBUG_COMPILE
if (DEBUG) {
  lock_cerr_mutex();
  distinguished_name.show("distinguished_name:");
  unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1503.**Log**

[LDF 2013.05.10.] Added this section.

```

⟨ listen_remote_X_509 definition 1499 ⟩ +≡
status = param->get_database_username();
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'listen_remote_X_509': " << "Scan_Parse_Parameter\"
        er_Type::get_database_username' failed, returning " << status << ". " << endl <<
        "Deinitializing 'param->session'. Continuing main accept loop." << endl;
    unlock_cerr_mutex();
    gnutls_deinit(param->session);
    delete param;
    param = 0;
    continue;
} /* if (status ≠ 0) */
#if DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'listen_remote_X_509': " << "Scan_Parameter_Type::get_databas\"
            e_username' succeeded, returning 0. " << endl << "'param->username' = "
            param->username << endl << "'param->user_id' = " << param->user_id << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
User_Info_Type user_info;
status = param->get_user(0, 0, "", &user_info, true);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'listen_remote_X_509': " <<
        "Scan_Parameter_Type::get_user' failed, returning " << status << ". " << endl <<
        "Deinitializing 'param->session'. Continuing main accept loop." << endl;
    unlock_cerr_mutex();
    gnutls_deinit(param->session);
    delete param;
    param = 0;
    continue;
} /* if (status ≠ 0) */
#if DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'listen_remote_X_509': " << "Scan_Parameter_Type::get_user' s\
            ucceeded, returning 0. " << endl << "'param->username' = " << param->username <<
            endl << "'param->user_id' = " << param->user_id << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1504. Put **User-Info-Type** *user_info* onto *global_user_info_map*. [LDF 2013.07.18.]

Log

[LDF 2013.07.18.] Added this section.

```
{ listen_remote_X_509 definition 1499 } +≡
    pthread_mutex_lock(&global_user_info_map_mutex);
    global_user_info_map[user_info.get_user_id()] = user_info;
    pthread_mutex_unlock(&global_user_info_map_mutex);
#
#
```

1505. Print session info. [LDF 2013.05.08.]

```
{ listen_remote_X_509 definition 1499 } +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[gwirdsif][Thread" << listen_remote_X_509_thread_ctr << "]"
            "In['listen_remote_X_509']:" << endl << "Printing[session]info:" << endl;
        print_info(param->session);
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1506.

Log

[LDF 2013.05.23.] Added code for setting *global_thread_ctr* after it has wrapped around. This should never happen, unless **gwirdsif** runs for a *long* time without being stopped.

```
{ listen_remote_X_509 definition 1499 } +≡
    pthread_mutex_lock(&thread_ctr_mutex);
    param->thread_ctr = 0;
    if (global_thread_ctr_wrapped_around ≡ false) {
        param->thread_ctr = ++global_thread_ctr;
        if (global_thread_ctr ≡ UINT_MAX) {
            global_thread_ctr_wrapped_around ≡ true;
            global_thread_ctr = save_global_thread_ctr;
        }
    }
```

1507.

```

⟨ listen_remote_X_509 definition 1499 ⟩ +≡
else {
    map<unsigned int, pthread_t>::iterator iter;
    for (int i = global_thread_ctr; i ≤ UINT_MAX; ++i) {
        iter = thread_ctr_id_map.find(i);
        if (iter ≡ thread_ctr_id_map.end()) {
            param→thread_ctr = i;
            if (i ≡ UINT_MAX) global_thread_ctr = save_global_thread_ctr;
            else global_thread_ctr = i + i;
            break;
        }
    } /* for */
if (param→thread_ctr ≡ 0) { /* Try again once more from the beginning. [LDF 2013.05.23.] */
    for (unsigned int i = save_global_thread_ctr; i ≤ UINT_MAX; ++i) {
        iter = thread_ctr_id_map.find(i);
        if (iter ≡ thread_ctr_id_map.end()) {
            param→thread_ctr = i;
            if (i ≡ UINT_MAX) global_thread_ctr = save_global_thread_ctr;
            else global_thread_ctr = i + i;
            break;
        }
    } /* if */
if (param→thread_ctr ≡ 0) {
    /* If we still couldn't set param→thread_ctr, we quit. [LDF 2013.05.23.] */
    lock_cerr_mutex();
    cerr << "[gwrdsif] [Thread]" << listen_remote_X_509_thread_ctr <<
        "] ERROR! In 'listen_remote_X_509': Failed to set 'param->thread_ctr'." <<
        endl << "Exiting thread unsuccessfully with return value 0." << endl;
    unlock_cerr_mutex();
    pthread_mutex_unlock(&thread_ctr_mutex);
    delete param;
    param = 0;
    close(listen_sd);
    pthread_exit(0);
}
} /* else */
pthread_mutex_unlock(&thread_ctr_mutex);

```

1508.

```

⟨ listen_remote_X_509 definition 1499 ⟩ +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[gwiridsif][Thread]" << listen_remote_X_509_thread_ctr << "]"
            << "In 'listen_remote_X_509':"
            << "Calling 'connect_func' via 'pthread_create'."
            << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    param->remote_connection = true;
    param->anonymous = false;
    status = pthread_create(&id, &attr,      /* Attribute. Thread is detached. [LDF 2013.05.15.] */
                           connect_func, static_cast<void *>(param));
    thread_ctrl_id_map[param->thread_ctrl] = id;
    thread_id_ctrl_map[id] = param->thread_ctrl;
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[gwiridsif][Thread]" << listen_remote_X_509_thread_ctr << "]"
            << "'pthread_create' returned"
            << status << "."
            << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    param = 0; } /* for */

```

1509. Exit. This code will never be reached in normal operation. [LDF 2012.09.21.]

```

⟨ listen_remote_X_509 definition 1499 ⟩ +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[gwiridsif][Thread]"
            << listen_remote_X_509_thread_ctr << "]"
            << "Exiting thread function 'listen_remote_X_509' successfully with return value 0."
            << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    pthread_attr_destroy(&attr);
    pthread_exit(0); } /* End of listen_remote_X_509 definition */

```

1510.

⟨ Garbage 303 ⟩ +≡

1511. Putting `lstnrmtx.web` together.

1512. This is what's compiled.

```
< Include files 3 >
#define SAstruct sockaddr
#define DH_BITS 1024
#define SOCKET_ERR(err,s)
    if (err == -1) {
        perror(s);
        return (1);
    }
using namespace std;
using namespace gwrdifpk;
< listen_remote_X_509 declaration 1498 >
< listen_remote_X_509 definition 1499 >
#ifndef 0
    < Garbage 303 >
#endif
```

1513. This is what's written to the header file `lstnrmtx.h`. [LDF 2012.09.21.]

```
< lstnrmtx.h 1513 > ==
#ifndef LSTNRMTX_H
#define LSTNRMTX_H 1
using namespace std;
using namespace gwrdifpk;
< listen_remote_X_509 declaration 1498 >
#endif
```

1514. Exchange data with client. (*exchange_data_with_client*). [LDF 2012.07.27.]

This file belongs to the *server*. *exchange_data_with_client* is called by the server, so `exchncli.o` is linked with `gwirdsif`. [LDF 2012.07.30.]

Log

[LDF 2012.07.27.] Added this file.

1515. Include files.

```
<Include files 3> +≡
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <pwd.h>
#include <errno.h>
#include <grp.h>
#include <sys/wait.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>
#include <time.h>
#include <sys/stat.h>
#include <dirent.h>
#include <fcntl.h>
#include <string>
#include <iomanip>
#include <iostream>
#include <iostream>
#include <stdarg.h>
#include <limits.h>
#include <algorithm>
#include <fstream>
#include <iterator>
#include <time.h>
#include <math.h>
#include <sstream>
#include <map>
#include <vector>
#include <deque>
#include <stack>
#include <set>
#include <pthread.h>
#include <expat.h>
#include <gcrypt.h> /* for gcry_control */
#include <gnutls/gnutls.h>
#include <gnutls/x509.h>
#include <mysql.h>
#undef NAME_LEN
#undef LOCAL_HOST
#if HAVE_CONFIG_H
#include "config.h"
#endif
#include "glblcnst.h++"
#include "glblvrbl.h++"
#include "excptntp.h++"
#include "utilfncts.h++"
#include "grouptp.h++"
#include "hndlvltp.h++"
#include "irdsavtp.h++"
```

```
#include "helper.h++"
#include "parser.h++"
#include "scanner.h++"
#include "rspnstp.h++"
#include "irdsobtp.h++"
#include "hndltype.h++"
#include "dcmtddtp.h++"
#include "x509cert.h++"
#include "dstngnmt.h++"
#include "scprpmtp.h++"
#include "pidfncs.h++"
```

1516. Exchange data with client (*exchange_data_with_client*). [LDF 2012.07.27.]

Log

[LDF 2012.07.27.] Added this function.

⟨ *exchange_data_with_client* declaration 1516 ⟩ ≡
int *exchange_data_with_client*(**Scan_Parse_Parameter_Type** &*param*);

This code is used in sections 1544 and 1545.

1517.

⟨ *exchange_data_with_client* definition 1517 ⟩ ≡
int *exchange_data_with_client*(**Scan_Parse_Parameter_Type** &*param*) { **bool** DEBUG = *false*;
 /* true */
set_debug_level(DEBUG, 0, 0);
int *status*;
stringstream *temp_strm*;
int *fd*;
ofstream *out_strm*;
ifstream *in_strm*;
char *buffer*[BUFFER_SIZE];
memset(*buffer*, 0, BUFFER_SIZE);
char **buffer_ptr* = 0;
yyscan_t *parameter*;
yylex_init(&*parameter*);
yyset_extra(&*param*, *parameter*);
 YY_BUFFER_STATE *yy_buffer_state*;
char *out_strm_filename*[] = "/tmp/gwirdsif.XXXXXX";
string *temp_filename*;
FILE **fp* = 0;
bool *read_data* = *false*;
bool *authentication_error* = *false*;

See also sections 1518, 1519, 1520, 1521, 1522, 1523, 1524, 1525, 1526, 1527, 1528, 1529, 1530, 1531, 1532, 1533, 1534, 1535, 1536, 1537, 1538, 1539, and 1540.

This code is used in section 1544.

1518.**Log**

[LDF 2012.12.13.] BUG FIX: Cleaned up the code for dumping input to a file. This will need to be tested. The amount of input has been small so far, so it may never have been dumped.

```

⟨ exchange_data_with_client definition 1517 ⟩ +≡
  for ( ; ; ) { status = 0;
    fp = 0;
    read_data = false;
    fd = 0; do {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[gwirdsif] [Thread" << param.thread_ctr << "] In " 'exchange_data_with_client' :" <<
      endl << "'param.remote_connection'" <=> param.remote_connection << endl;
    if (param.remote_connection) cerr << "Calling " 'gnutls_record_recv' "..." << endl;
    else cerr << "Calling " 'recv' "..." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    memset(buffer, 0, BUFFER_SIZE);
    if (param.remote_connection == true) {
      status = gnutls_record_recv(param.session, buffer, BUFFER_SIZE);
    }
    else {
      status = recv(param.sock, buffer, BUFFER_SIZE, 0);
    }
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[gwirdsif] [Thread" << param.thread_ctr << "] In " 'exchange_data_with_client' :" <<
      endl << "'gnutls_record_recv'" ;
    if (param.remote_connection) cerr << "'recv'" ;
    else cerr << "returned" << status << "." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1519.

```

⟨ exchange_data_with_client definition 1517 ⟩ +≡
  if (status > 0) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[gwirdsif] [Thread" << param.thread_ctr << "] " <<
      "Received_text_from_client: buffer==" << endl;
    fwrite(buffer, 1, status, stderr);
    cerr << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1520.**Log**

[LDF 2013.04.09.] BUG FIX: Now pushing *out_strm_filename* onto *param.temp_file_vector*, if *save_temp_files* \equiv *false*. This ensures that the temporary file is deleted when the thread exits (unless temporary files are being saved).

```

⟨ exchange_data_with_client definition 1517 ⟩ +≡
  if (status ≡ BUFFER_SIZE ∧ read_data ≡ false) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[gwirdsif][Thread" << param.thread_ctr <<
      "] " << "In‘exchange_data_with_client’: " << endl <<
      "'status' == 'BUFFER_SIZE'. Will dump input to file." << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  strcpy(out_strm_filename, "/tmp/gwirdsif.XXXXXX");
  fd = mkstemp(out_strm_filename);
  if (fd ≡ -1) {
    lock_cerr_mutex();
    cerr << "[gwirdsif][Thread" << param.thread_ctr << "] "
      << "ERROR! In‘exchange_data_with_client’: " << "'mkstemp' failed, returning -1." <<
      endl << "mkstemp error: " << strerror(errno) << endl <<
      "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    close(param.sock);
    param.sock = 0;
    return 1;
  } /* if (fd ≡ -1) */
#endif DEBUG_COMPILE
  else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[gwirdsif][Thread" << param.thread_ctr <<
      "] In‘exchange_data_with_client’: " << "'mkstemp' succeeded. "
      << "'out_strm_filename' == " << " " << out_strm_filename << ". " << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  if (!save_temp_files) param.temp_file_vector.push_back(out_strm_filename);
  close(fd); /* We just need the name. [LDF 2012.07.30.] */
  fd = 0;
  out_strm.open(out_strm_filename, ios_base::out | ios_base::binary);
  if (!(out_strm ∧ out_strm.is_open())) {
    lock_cerr_mutex();
    cerr << "[gwirdsif][Thread" << param.thread_ctr <<
      "] ERROR! In‘exchange_data_with_client’: " << "'out_strm.open()' failed." <<
      endl << "Failed to open temporary output file " << out_strm_filename << ". "
      << endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
  }
}

```

```

close(param.sock);
param.sock = 0;
return 1;
} /* if (!(out_strm & out_strm.is_open())) */
#endif DEBUG_COMPILE
else
if (DEBUG) {
lock_cerr_mutex();
cerr << "[gwirdsif][Thread]" << param.thread_ctr <<
"]In['exchange_data_with_client':out_strm.open()'succeeded." << endl;
unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
out_strm.write(buffer, status);
read_data = true;
} /* if (status == BUFFER_SIZE & read_data == false) */
else if (status == BUFFER_SIZE & read_data == true) {
#endif DEBUG_COMPILE
if (DEBUG) {
lock_cerr_mutex();
cerr << "[gwirdsif][Thread]" << param.thread_ctr << "]In['exchange_data_with_client':" <<
endl << "'status'<='BUFFER_SIZE':" << endl << "'status'=='" <<
status << endl << "'BUFFER_SIZE'==" << BUFFER_SIZE << endl <<
"'read_data'=='true'.Writing to 'out_strm'." << endl;
unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
out_strm.write(buffer, status);
memset(buffer, 0, BUFFER_SIZE);
} /* else if (status == BUFFER_SIZE & read_data == true) */
else if (status < BUFFER_SIZE & read_data == true) {
#endif DEBUG_COMPILE
if (DEBUG) {
lock_cerr_mutex();
cerr << "[gwirdsif][Thread]" << param.thread_ctr << "]In['exchange_data_with_client':" <<
endl << "'status'<'BUFFER_SIZE':" << endl << "'status'==" <<
status << endl << "'BUFFER_SIZE'==" << BUFFER_SIZE << endl <<
"'read_data'=='true'.Writing to 'out_strm' and breaking." << endl;
unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
out_strm.write(buffer, status);
memset(buffer, 0, BUFFER_SIZE);
break;
} /* else if (status < BUFFER_SIZE & read_data == true) */
else if (status < BUFFER_SIZE & read_data == false) {
#endif DEBUG_COMPILE
if (DEBUG) {
lock_cerr_mutex();
cerr << "[gwirdsif][Thread]" << param.thread_ctr << "]In['exchange_data_with_client':" <<
endl << "'status'<'BUFFER_SIZE':" << endl << "'status'==" << status << endl <<

```

```
"'BUFFER_SIZE'" << BUFFER_SIZE << endl << "'read_data'" <= 'false' . "Breaking." <<
endl;
unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
break;
} /* else if (status == BUFFER_SIZE & read_data == false) */
} /* if (status > 0) */
```

1521.

Log

[LDF 2013.07.11.] Added conditional `if (param.client_finished & param.server_finished)`. No longer outputting error message if the client breaks off the connection in this case. Due to other changes, the client may break the connection in normal use.

```

⟨ exchange_data_with_client definition 1517 ⟩ +≡
else {
    if (param.client_finished & param.server_finished) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "'param.client_finished' and 'param.server_finished'." << endl <<
            "Client ended connection." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    lock_cerr_mutex();
    cerr << "[Thread" << param.thread_ctr << "] Client ended connection" << endl;
    unlock_cerr_mutex();
    close(param.sock);
    param.sock = 0;
    return 0;
}
if (status < 0) {
    lock_cerr_mutex();
    cerr << "[Thread" << param.thread_ctr << "] ERROR! In 'exchange_data_with_client':";
    if (param.remote_connection) cerr << "'gnutls_record_recv'";
    else cerr << "'recv'";
    cerr << " failed, returning" << status << "." << endl;
    perror("recv");
    unlock_cerr_mutex();
}
else {
    lock_cerr_mutex();
    cerr << "[gwirdsif]" << "[Thread" << param.thread_ctr <<
        "] In 'exchange_data_with_client':" << "Client closed connection\n";
    unlock_cerr_mutex();
}
lock_cerr_mutex();
cerr << "[gwirdsif]" << "[Thread" << param.thread_ctr << "]"
    "Exiting function unsuccessfully with return value 1." << endl;
unlock_cerr_mutex();
close(param.sock);
param.sock = 0;
return 1;
} /* else */
} /* do */
while (status ≡ BUFFER_SIZE) ;

```

1522. Call *yyparse*. [LDF 2012.06.26.]

Log

BUG FIX: Now calling *yyrestart* instead of *yyset_in*. Previously, using *zzset_in* in *exchange_data_with_server* (i.e., *not* this function) sometimes (but not always!) caused a segmentation fault, apparently because changing the input source caused the address of the lookahead token to become invalid. No problem ever occurred in this function, but it seems like a good idea to make this change, anyway. One reason no problem has ever occurred is probably because the only time input has been stored in the file is when the client first sends the contents of the input file. Otherwise, the amount of text in the commands has probably been very small.

```
< exchange_data_with_client definition 1517 > +≡
    fp = 0;
    if (read_data) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "read_data" <= true . " << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    out_strm.close();
    fp = fopen(out_strm_filename, "r");
    if (fp == 0) {
        lock_cerr_mutex();
        cerr << "[gwirdsif][Thread" << param.thread_ctr << "] ERROR! In 'excha\
nge_data_with_client': " << "'fopen' failed, returning NULL." << endl <<
        "Failed to open input file " << out_strm_filename << ". " << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        close(param.sock);
        param.sock = 0;
        return 1;
    } /* if (fp == 0) */
    else if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[gwirdsif][Thread" << param.thread_ctr << "] In 'exchange_data\
_with_client': " << "Calling yyrestart(reading from input file " <<
        out_strm_filename << "). " << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
    yyrestart(fp, parameter);
#endif 0
    yyset_in(fp, parameter); /* See log entry above. [LDF 2013.05.29.] */
#endif
} /* if */
else yy_buffer_state = yy_scan_string(buffer, parameter);
```

1523.**Log**

[LDF 2012.09.19.] Added this section. Now not calling *yyparse* if *buffer* is empty.
[LDF 2012.12.13.] Now testing whether *read_data* ≡ *false* in the conditional below.

```
< exchange_data_with_client definition 1517 > +≡
  if (read_data ≡ false ∧ buffer[0] ≡ '\0') {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[gwirdsif]" [Thread_] << param.thread_ctr << "] In 'exchange_data_with_client':"
      endl << "'buffer' is empty and 'read_data' is 'false'. "
      "Not calling 'yyparse' to parse input." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (No data to process) */
```

1524.

```

⟨ exchange_data_with_client definition 1517 ⟩ +≡
else /* Data to be processed is present */
{
    param.client_finished = false;
    status = yyparse(parameter);
    if (status ≡ 2) {
        lock_cerr_mutex();
        cerr << "[gwiridsif][Thread" << param.thread_ctr << "] ERROR! In 'excha\
nge_data_with_client': " << "'yyparse' failed, returning 2." << endl <<
        "Authentication_error.Will_send_message_to_client_and_break." << endl;
        unlock_cerr_mutex();
        authentication_error = true;
    } /* if (status ≡ 2) */
    else if (status ≠ 0) {
        lock_cerr_mutex();
        cerr << "[gwiridsif][Thread" << param.thread_ctr <<
            "] ERROR! In 'exchange_data_with_client': " << "'yyparse' failed, returning" <<
            status << "." << endl << "Exiting function unsuccessfully with return value 1." <<
            endl;
        unlock_cerr_mutex();
        if (fp ≡ 0) yy_delete_buffer(yy_buffer_state, parameter);
        else {
            fclose(fp);
            fp = 0;
        }
        close(param.sock);
        param.sock = 0;
        return 1;
    } /* else if (status ≠ 0) */
    else if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[gwiridsif][Thread" << param.thread_ctr <<
            "] In 'exchange_data_with_client': " << "'yyparse' succeeded, returning 0." <<
            endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
    if (fp ≡ 0) yy_delete_buffer(yy_buffer_state, parameter);
    else {
        fclose(fp);
        fp = 0;
    }
} /* else (Data to be processed is present) */

```

1525.

```

⟨ exchange_data_with_client definition 1517 ⟩ +≡
memset(buffer, 0, BUFFER_SIZE);

```

1526.**Log**

[LDF 2013.05.10.] Added this section.

```
< exchange_data_with_client definition 1517 > +≡
if (authentication_error ∨ (param.connection_type ≠ Scan_Parse_Parameter_Type::X_509_AUTH_TYPE ∧
    param.user_id ≤ 0)) {
    lock_cerr_mutex();
    cerr << "[gwiridsif] [Thread" << param.thread_ctr <<
        "] ERROR! In 'exchange_data_with_client': " << endl << "'authentication_error' == " <<
        authentication_error << endl << "'param.connection_type' == " <<
        param.connection_type << endl << "'param.user_id' == " << param.user_id <<
        endl << "Authentication error: For non-authenticated connections, " <<
        "the \"DISTINGUISHED_NAME\" command must be called with a correct argument" <<
        "during the first invocation of 'yparse'." << endl <<
        "Please note that non-authenticated connections are only for " <<
        "testing purposes and should not be used for production!" << endl <<
        "Sending error message to client and breaking." << endl;
    unlock_cerr_mutex();
    memset(buffer, 0, BUFFER_SIZE);
    strcpy(buffer, "AUTHENTICATION_ERROR\1");
    if (param.remote_connection ≡ true) {
        gnutls_record_send(param.session, buffer, strlen(buffer));
    }
    else {
        send(param.sock, buffer, strlen(buffer), 0);
    }
    break;
} /* if (Authentication error) */
```

1527.

Log

[LDF 2013.05.24.] Added this section.

```
< exchange_data_with_client definition 1517 > +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[gwirldsif] [Thread" << param.thread_ctr << "] In " 'exchange_data_with_client' : " <<
            endl << "'param.response_deque.size()' == " << param.response_deque.size() << endl <<
            "'param.delayed_response_deque.size()' == " << param.delayed_response_deque.size() <<
            endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    if (param.response_deque.size() == 0 &amp; param.delayed_response_deque.size() > 0) {
        param.response_deque = param.delayed_response_deque;
        param.delayed_response_deque.clear();
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[gwirldsif] [Thread" << param.thread_ctr << "] In " 'exchange_data_with_client' : " <<
            endl << "'param.response_deque' was empty, but 'param.delayed_response_deque' " <<
            " contained responses. " << endl << "Copied them to 'param.response_deque' and cleared " <<
            "'param.delayed_response_deque'. " << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if */
```

1528.

```
< exchange_data_with_client definition 1517 > +≡
    if (param.response_deque.size() > 0) { param.server_finished = false;
#ifndef DEBUG_COMPILE
    if (DEBUG) { lock_cerr_mutex();
        cerr << "[gwirldsif] [Thread" << param.thread_ctr << "] In " 'exchange_data_with_client' : " <<
            endl << "'param.response_deque.size()' == " << param.response_deque.size() << ". " << endl;
        int i = 0; for ( deque < Response_Type > ::iterator iter = param.response_deque.begin();
            iter != param.response_deque.end(); ++iter )
        {
            cerr << "Response" << i++ << " type == " << Response_Type::typename_map[iter->type] << endl;
            if (!iter->command.empty()) cerr << "Command: " << iter->command << endl;
        }
        cerr << endl;
        unlock_cerr_mutex(); } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (param.response_deque.size() > 0) */
```

1529.

```

⟨ exchange_data_with_client definition 1517 ⟩ +≡
  else /* param.response_deque.size() == 0 */
  {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[gwiridsif]" [Thread_] << param.thread_ctr << "]_In_`exchange_data_with_client':"
    endl << `param.response_deque.size()' == 0." << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  param.server_finished = true;
} /* else (param.response_deque.size() == 0) */

```

1530.

Log

[LDF 2013.07.15.] BUG FIX: Add space character after "SERVER_FINISHED-END". Previously, this sometimes caused a client-side parse error (i.e., in `zzparse`), when these commands were sent multiple times, because "END" and "SERVER" were combined to "ENDSERVER".

```

⟨ exchange_data_with_client definition 1517 ⟩ +≡
  if (param.client_finished ∧ param.server_finished) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[gwiridsif]" [Thread_] << param.thread_ctr << "]_In_`exchange_data_with_client':"
    endl << `param.client_finished' == 'true' and "
    `param.server_finished' == 'true'." << endl <<
    "Sending_" SERVER_FINISHED " command_to_client." << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  memset(buffer, 0, BUFFER_SIZE);
  strcpy(buffer, "SERVER_FINISHED-END");
  if (param.remote_connection == true) {
    gnutls_record_send(param.session, buffer, strlen(buffer));
  }
  else {
    send(param.sock, buffer, strlen(buffer), 0);
  }
  memset(buffer, 0, BUFFER_SIZE);
} /* if (param.client_finished ∧ param.server_finished) */

```

1531.**Log**

[LDF 2012.09.07.] Added this section. BUG FIX: Now testing whether *param.response_deque* is empty.

[LDF 2012.09.19.] Now sending SERVER FINISHED END to client. Added error-handling code for the case that *send* fails.

[LDF 2013.07.15.] BUG FIX: Add space character after "SERVER_UFINISHED_UEND". Previously, this sometimes caused a client-side parse error (i.e., in *zzparse*), when these commands were sent multiple times, because "END" and "SERVER" were combined to "ENDSERVER".

```

⟨ exchange_data_with_client definition 1517 ⟩ +≡
  if (param.response_deque.size() ≡ 0) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[gwirdsif]" << param.thread_ctr << "]In‘exchange_data_with_client’:"
    endl << "'param.response_deque.size()'==0." << endl <<
    "Setting‘param.server_finished’to‘true’and"
    "sending‘SERVER_UFINISHED_UEND’toclient." << endl << "param.client_finished=="
    param.client_finished << endl << "param.server_finished=="
    & param.server_finished << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  param.server_finished = true;
  memset(buffer, 0, BUFFER_SIZE);
  strcpy(buffer, "SERVER_UFINISHED_UEND_U");
  errno = 0;
  if (param.remote_connection ≡ true) {
    status = gnutls_record_send(param.session, buffer, strlen(buffer));
  }
  else {
    status = send(param.sock, buffer, strlen(buffer), 0);
  }
  memset(buffer, 0, BUFFER_SIZE);
}

```

1532.**Log**

[LDF 2013.07.15.] Added test of `param.client_finished` to conditional. If `param.client_finished` \equiv `true`, then the client may have exited, in which case `send` or `gnutls_record_send` will have failed. However, this doesn't matter at this point, so no error message should be output in this case.

```

⟨ exchange_data_with_client definition 1517 ⟩ +≡
if (status < 0 ∧ ¬param.client_finished) {
    lock_cerr_mutex();
    cerr ≪ "[gwirdsif] [Thread" ≪ param.thread_ctr ≪
        "] [ERROR!] In 'exchange_data_with_client':";
    if (param.remote_connection) cerr ≪ "'gnutls_record_send' failed, returning" ≪ status ≪
        "." ≪ endl ≪ "error:" ≪ gnutls_strerror(status) ≪ endl;
    else cerr ≪ "'send' failed, returning" ≪ status ≪ "." ≪ endl ≪ "error:" ≪
        strerror(errno) ≪ endl;
    cerr ≪ "Exiting function unsuccessfully with return value 1." ≪ endl;
    unlock_cerr_mutex();
    close(param.sock);
    param.sock = 0;
    return 1;
} /* if (status < 0 ∧ ¬param.client_finished) */

```

1533.

```

⟨ exchange_data_with_client definition 1517 ⟩ +≡
#ifndef DEBUG_COMPILE
    else
        if (DEBUG ∧ status ≥ 0) {
            lock_cerr_mutex();
            cerr ≪ "[gwirdsif] [Thread" ≪ param.thread_ctr ≪
                "] In 'exchange_data_with_client':";
            if (param.remote_connection) cerr ≪ "'gnutls_record_send'";
            else cerr ≪ "'send'";
            cerr ≪ " succeeded, returning" ≪ status ≪ "." ≪ endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    continue; /* if (param.response_deque.size() ≡ 0) */

```

1534. Server not finished. Handling responses. [LDF 2012.09.06.]

```

⟨ exchange_data_with_client definition 1517 ⟩ +≡
Response_Type response = param.response_deque.front();
param.response_deque.pop_front();
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        response.show("response:");
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1535. Find the "server action" function to call, based on *response.type*. [LDF 2013.05.27.]

Log

[LDF 2013.05.27.] Added this section.

[LDF 2013.05.30.] Finished moving code from this function to "server action" member functions of **Scan_Parse_Parameter_Type** defined in **srvractn.web**.

< exchange_data_with_client definition 1517 > +≡

1536.

< exchange_data_with_client definition 1517 > +≡

```
typedef int(Scan_Parse_Parameter_Type::*func_ptr)(Response_Type &);
string server_action_name;
map<unsigned int,func_ptr>::iterator iter =
    param.Scan_Parse_Parameter_Type::server_action_map.find(response.type);
```

1537.

< exchange_data_with_client definition 1517 > +≡

```
#if DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Response_type:" << Response_Type::typename_map[response.type] << " " << "(" <<
            response.type << ")" << endl;
        if (!response.command.empty())
            cerr << "'response.command'" << response.command << endl;
        if (iter != Scan_Parse_Parameter_Type::server_action_map.end()) {
            cerr << "Server_action:" <<
                Scan_Parse_Parameter_Type::server_action_name_map[response.type] << endl;
        }
        else {
            cerr << "Server_action_not_found:" << Response_Type::typename_map[response.type] <<
                endl;
        }
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1538. Under normal circumstances, this code should never be reached. It can be reached during development, if a new response type has been added, but a corresponding "server action" function has not yet been added to **Scan_Parse_Parameter_Type::server_action_map**. [LDF 2013.05.30.]

< exchange_data_with_client definition 1517 > +≡

```
if (iter == Scan_Parse_Parameter_Type::server_action_map.end()) {
    server_action_name = "Scan_Parse_Parameter_Type::server_action_unknown";
    status = param.Scan_Parse_Parameter_Type::server_action_unknown(response);
}
```

1539. Found the “server action” function. Now call it. [LDF 2013.05.30.]

```
( exchange_data_with_client definition 1517 ) +≡
else
  if (iter ≠ Scan_Parser_Parameter_Type::server_action_map.end()) {
    server_action_name = Scan_Parser_Parameter_Type::server_action_name_map[response.type];
    /* Don't bother checking if server_action_name is set. It should be, and it's only used in an error
       message or debugging output, below. [LDF 2013.05.27.] */
    status = (param.* iter->second)(response);
  if (response.type ≡ Response_Type::END_SERVER_TYPE ∧ status ≡ 2) {
    lock_cerr_mutex();
    cerr ≪ "[gwirdsif] [Thread]" ≪ param.thread_ctr ≪ "In 'exchange_data_with_client':"
    endl ≪ "'END_SERVER' command succeeded."
    "Exiting function successfully with return value 2." ≪ endl;
    unlock_cerr_mutex();
    close(param.sock);
    param.sock = 0;
    return 2;
  } /* if */
  else if (status ≡ 2) {
    lock_cerr_mutex();
    cerr ≪ "[gwirdsif] [Thread]" ≪ param.thread_ctr ≪
      "] WARNING! In 'exchange_data_with_client':"
    endl ≪ "' failed, returning'" ≪ status ≪ "."
    endl ≪ "Will try to continue." ≪ endl;
    unlock_cerr_mutex();
    ++param.warnings_occurred;
    memset(buffer, 0, BUFFER_SIZE);
    buffer_ptr = buffer;
    continue;
  } /* if (status ≡ 2) */
  else if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ "[gwirdsif] [Thread]" ≪ param.thread_ctr ≪
      "] ERROR! In 'exchange_data_with_client':"
    endl ≪ "' failed, returning'" ≪ status ≪ "."
    endl ≪ "Exiting function unsuccessfully with return value 1."
    unlock_cerr_mutex();
    param.send_to_peer(0, 1);
    /* Send a single NULL byte to prevent client from blocking, just in case. Don't bother checking
       the return value of Scan_Parser_Parameter_Type::send_to_peer. [LDF 2013.05.29.] */
    close(param.sock);
    param.sock = 0;
    return 1;
  } /* if (status ≠ 0) */
#endif DEBUG_COMPILE
else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "[gwirdsif] [Thread]" ≪ param.thread_ctr ≪
      "] In 'exchange_data_with_client':"
    endl ≪ "' succeeded, returning 0.'"
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
```

```
#endif /* DEBUG_COMPILE */
    } /* else if (iter != Scan_Parse_Parameter_Type::server_action_map.end()) */
} /* for */
```

1540.

```
< exchange_data_with_client definition 1517 > +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[gwirdsif][Thread_"
        << param.thread_ctr <<
        "]Exiting['exchange_data_with_client']"
        << "successfully with return value 0."
        << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; } /* End of exchange_data_with_client definition */
```

1541.

```
< External function declarations 32 > +≡
int yyparse(yyscan_t parameter);
```

1542.

```
< Garbage 303 > +≡
```

1543. Putting exchncli.web together.

1544. This is what's compiled.

```
< Include files 3 >
using namespace std;
using namespace gwrdifpk;
< External function declarations 32 >
< exchange_data_with_client declaration 1516 >
< exchange_data_with_client definition 1517 >
#ifndef 0
< Garbage 303 >
#endif
```

1545. This is what's written to the header file `exchncli.h`. [LDF 2012.06.27.]

```
<exchncli.h 1545>≡  
#ifndef EXCHNCLI_H  
#define EXCHNCLI_H 1  
    using namespace std;  
    using namespace gwrdifpk;  
    <exchange_data_with_client declaration 1516>  
#endif
```

1546. Exchange data with server. (*exchange_data_with_server*). [LDF 2012.07.27.]

This file belongs to the *client*. *exchange_data_with_server* is called by the client, so `exchnsrv.o` is linked with `gwrdbap` and `gwirdcli`. [LDF 2012.07.30.]

Log

[LDF 2012.07.27.] Added this file.

1547. Include files.

```
<Include files 3> +≡
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <pwd.h>
#include <errno.h>
#include <grp.h>
#include <sys/wait.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>
#include <time.h>
#include <sys/stat.h>
#include <dirent.h>
#include <fcntl.h>
#include <string>
#include <iomanip>
#include <iostream>
#include <stdarg.h>
#include <limits.h>
#include <algorithm>
#include <fstream>
#include <iterator>
#include <time.h>
#include <math.h>
#include <sstream>
#include <map>
#include <vector>
#include <deque>
#include <stack>
#include <set>
#include <pthread.h>
#if HAVE_CONFIG_H
#include "config.h"
#endif
#include <gcrypt.h> /* for gcry_control */
#include <gnutls/gnutls.h>
#include <gnutls/x509.h>
#include <mysql.h>
#include <expat.h>
#undef NAME_LEN
#undef LOCAL_HOST
#include "glblcnst.h++"
#include "glblvrbl.h++"
#include "excptntp.h++"
#include "helper.h++"
#include "utilfncts.h++"
#include "grouptp.h++"
#include "hdlvltp.h++"
```

```
#include "irdsavtp.h++"
#include "cmdlnopt.h++"
#include "prsrcnt.h++"
#include "scnrcnt.h++"
#include "rspnstp.h++"
#include "irdsobtp.h++"
#include "hndltype.h++"
#include "dcmtddtp.h++"
#include "x509cert.h++"
#include "dstngnmt.h++"
#include "scprpmtp.h++"
```

1548. Exchange data with server (*exchange_data_with_server*). [LDF 2012.07.27.]

Log

[LDF 2012.07.27.] Added this function.

```
⟨ exchange_data_with_server declaration 1548 ⟩ ≡
int exchange_data_with_server(Scan_Parse_Parameter_Type &param);
```

This code is used in sections 1576 and 1577.

1549.

```

⟨ exchange_data_with_server definition 1549 ⟩ ≡
int exchange_data_with_server(Scan_Parse_Parameter_Type &param){ bool DEBUG = false;
    /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    stringstream temp_strm;
    bool strm_val = is_gwirdcli;      /* output_func will write to cerr, if this function is called by
        gwirdcli, and stdout, if called by gwrdwbap. [LDF 2012.07.30.] */
    char buffer[BUFFER_SIZE];
    memset(buffer, 0, BUFFER_SIZE);
    size_t char_ctr;
    ifstream in_strm;
    ofstream out_strm;
    char out_strm_filename[21];
    int fd = 0;
    bool read_data = false;
    bool wrote_data = false;
    string recv_func_name = (remote_connection ≡ true) ? "gnutls_record_recv" : "recv";
    bool first_time = true;
    yy_scan_t parameter;
    zzlex_init(&parameter);
    zzset_extra(&param, parameter);
    YY_BUFFER_STATEzz_buffer_state;
FILE *fp = 0; for ( ; ; ) { status = 0;
    memset(buffer, 0, BUFFER_SIZE);
    char_ctr = 0;
    read_data = false;
    wrote_data = false;
    fd = 0;
    memset(out_strm_filename, 0, 21);
    if (first_time ≡ true ∧ param.data_filename.empty() ∧ strlen(param.data_buffer) ≡
        0 ∧ param.response_deque.size() ≡ 0) {
        temp_strm.str("");
        temp_strm ≪ "[Thread" ≪ param.thread_ctr ≪ "] ERROR! In 'exchange_data_with_server'
            ': "first_time'" ≡ 'true' ∧&& 'param.data_filename' and "
            endl ≪ "'param.data_buffer'" ≈ both ≈ empty ≈ and "
            endl ≪ "'param.response_deque.size()' ≈ 0'." ≪ endl ≪
            "Nothing to send to server 'gwirdsif'." ≪ endl ≪
            "Exiting function unsuccessfully with return value 1." ≪ endl;
        lock_cerr_mutex();
        output_func(temp_strm.str().c_str(), strm_val);
        unlock_cerr_mutex();
        return 1;
    } /* if (first_time ≡ true ∧ param.data_filename.empty() ∧ strlen(param.data_buffer) ≡
        0 ∧ param.response_deque.size() ≡ 0) */
}

```

See also sections 1550, 1551, 1552, 1553, 1554, 1555, 1556, 1557, 1558, 1559, 1560, 1561, 1562, 1563, 1564, 1565, 1566, 1567, 1568, 1569, 1570, 1571, and 1572.

This code is used in section 1576.

1550.**Log**

[LDF 2012.07.30.] Added this section.

```
< exchange_data_with_server definition 1549 > +≡
else if (first_time ≡ false ∧ param.data_filename.empty() ∧ strlen(param.data_buffer) ≡
0 ∧ param.response_deque.size() ≡ 0) {
#if DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ "[Thread" ≪ param.thread_ctr ≪ "] In 'exchange_data_with_server':"
        ≪ "'first_time' ≡ 'false' && 'param.data_filename' and "
        ≪ endl ≪ "'param.data_buffer' are both empty and "
        ≪ "'param.response_deq'"
        ue.size() == 0 . " ≪ endl ≪ "Nothing more to send to server 'gwirdsif' . "
        ≪ endl ≪ "'param.client_finished' == "
        ≪ param.client_finished ≪ endl ≪
        "'param.server_finished' == "
        ≪ param.server_finished ≪ endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1551.**Log**

[LDF 2013.07.11.] Added this section with code for reading input from standard input, if *terminate_on_end_input* \equiv *false*.

```
< exchange_data_with_server definition 1549 > +≡
if (terminate_on_end_input) {
    param.client_finished = true;
    memset(buffer, 0, BUFFER_SIZE);
    strcpy(buffer, "CLIENT\FINISHED\END\");
}
else if (param.server_finished) {
    memset(param.data_buffer, 0, BUFFER_SIZE);
    param.data_filename = "";
    status = param.get_input();
    if (status  $\neq$  0  $\vee$  (strlen(param.data_buffer)  $\equiv$  0  $\wedge$  param.data_filename.empty())) {
        param.client_finished = true;
        lock_cerr_mutex();
        lock_cout_mutex();
        cout  $\ll$  "Exiting."  $\ll$  endl;
        unlock_cout_mutex();
        unlock_cerr_mutex();
        memset(buffer, 0, BUFFER_SIZE);
        strcpy(buffer, "CLIENT\FINISHED\END\");
    }
    else if (strlen(param.data_buffer)  $>$  0) {
        param.client_finished = param.server_finished = false;
        memset(buffer, 0, BUFFER_SIZE);
        strcpy(buffer, param.data_buffer);
        memset(param.data_buffer, 0, BUFFER_SIZE);
    }
}
/* else if (Get input from standard input) */
else {
    memset(buffer, 0, BUFFER_SIZE);
}
```

1552.**Log**

[LDF 2013.03.01.] BUG FIX: Added space after "END" in the text copied to *buffer*: Previously, if this code was reached multiple times, the server would receive the text "ENDCLIENT", which caused a parse error.

```

⟨ exchange_data_with_server definition 1549 ⟩ +≡
char_ctr = strlen(buffer);
if (char_ctr ≡ 0) char_ctr += 1;
if (remote_connection) {
    gnutls_record_send(param.session, buffer, char_ctr);
}
else {
    send(param.sock, buffer, char_ctr, 0);
}
char_ctr = 0;
if (param.server_finished ∧ param.client_finished) {
#endif DEBUG_COMPILE
    if (DEBUG) {
        temp_strm.str("");
        temp_strm << "[Thread" << param.thread_ctr << "] In 'exchange_data_with_server':"
        endl << "'param.server_finished' == 'true' && "
        "'param.client_finished' == 'true'." << endl <<
        "Exiting function successfully with return value 0." << endl;
        lock_cerr_mutex();
        output_func(temp_strm.str().c_str(), strm_val);
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0;
} /* if (param.server_finished ∧ param.client_finished) */
} /* if (first_time ≡ false ∧ param.data_filename.empty() ∧ strlen(param.data_buffer) ≡
   0 ∧ param.response_deque.size() ≡ 0) */

```

1553. *param.response_deque* is non-empty.**Log**

[LDF 2012.11.19.] Added code for this case.

```

⟨ exchange_data_with_server definition 1549 ⟩ +≡
else if (param.response_deque.size() > 0) { first_time = false;
#endif DEBUG_COMPILE
    if (DEBUG) {
        temp_strm.str("");
        temp_strm << "[Thread" << param.thread_ctr << "] In 'exchange_data_with_server':"
        "'param.response_deque.size()' >= 0." << endl << "Will process top response." << endl;
        lock_cerr_mutex();
        output_func(temp_strm.str().c_str(), strm_val);
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1554. BUG FIX: If *response.command* is empty, now sending 1 byte to the server. Previously, *strlen(response.command)* 0 was “sent”, i.e., no bytes were sent, which caused the server to block. Of course, *response.command* should normally never be empty, but it can happen by mistake. [LDF 2012.11.21.]

```
< exchange_data_with_server definition 1549 > +≡
Response_Type response = param.response_deque.front();
param.response_deque.pop_front();
```

1555.

```
< exchange_data_with_server definition 1549 > +≡
typedef int(Scan_Parse_Parameter_Type::*func_ptr)(Response_Type &);
string client_action_name;
map<unsigned int,func_ptr>::iterator iter =
    param.Scan_Parse_Parameter_Type::client_action_map.find(response.type);
```

1556.

```
< exchange_data_with_server definition 1549 > +≡
#if DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Response_type:" << Response_Type::typename_map[response.type] << " " << "(" <<
            response.type << ")" << endl;
        if (response.command.empty())
            cerr << "'response.command'" << response.command << endl;
        if (iter != Scan_Parse_Parameter_Type::client_action_map.end()) {
            cerr << "Client_action:" <<
                Scan_Parse_Parameter_Type::client_action_name_map[response.type] << endl;
        }
        else {
            cerr << "Client_action_not_found:" << Response_Type::typename_map[response.type] <<
                endl;
        }
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1557. Under normal circumstances, this code should never be reached. It can be reached during development, if a new response type has been added, but a corresponding “client action” function has not yet been added to **Scan_Parse_Parameter_Type::client_action_map**. [LDF 2013.05.30.]

```
< exchange_data_with_server definition 1549 > +≡
    if (iter == Scan_Parse_Parameter_Type::client_action_map.end()) {
        client_action_name = "Scan_Parse_Parameter_Type::client_action_unknown";
        status = param.Scan_Parse_Parameter_Type::client_action_unknown(response);
    }
```

1558. Found the “client action” function. Now call it. [LDF 2013.05.30.]

```

⟨ exchange_data_with_server definition 1549 ⟩ +≡
else
if (iter ≠ Scan_Parse_Parameter_Type::client_action_map.end()) {
    client_action_name = Scan_Parse_Parameter_Type::client_action_name_map[response.type];
    /* Don't bother checking if client_action_name is set. It should be, and it's only used in an error
       message or debugging output, below. [LDF 2013.05.27.] */
    status = (param.* iter->second)(response);
    if (status ≡ 2) {
        lock_cerr_mutex();
        cerr << "[gwirdsif] [Thread" << param.thread_ctr <<
            "] In 'exchange_data_with_server': " << endl << "'"
            << client_action_name <<
            "' failed, returning" << status << "."
            << endl << "Will try to continue." << endl;
        unlock_cerr_mutex();
        ++param.warnings_occurred;
        memset(buffer, 0, BUFFER_SIZE);
        continue;
    } /* if (status ≡ 2) */
    else if (status ≠ 0) {
        lock_cerr_mutex();
        cerr << "[gwirdsif] [Thread" << param.thread_ctr <<
            "] ERROR! In 'exchange_data_with_server': " << endl << "'"
            << client_action_name << "' failed, returning"
            << status << "."
            << endl << "Exiting function unsuccessfully with return value 1."
            << endl;
        unlock_cerr_mutex();
        param.send_to_peer(0, 1);
        /* Send a single NULL byte to prevent server from blocking, just in case. Don't bother checking
           the return value of Scan_Parse_Parameter_Type::send_to_peer. [LDF 2013.05.29.] */
        close(param.sock);
        param.sock = 0;
        return 1;
    } /* if (status ≠ 0) */
#endif DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[gwirdsif] [Thread" << param.thread_ctr <<
        "] In 'exchange_data_with_server': " << endl << "'"
        << "' succeeded, returning 0."
        << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* else if (iter ≠ Scan_Parse_Parameter_Type::client_action_map.end()) */
} /* else if (param.response_deque.size() > 0) */

```

1559.

```
< exchange_data_with_server definition 1549 > +≡
  else if (¬param.data_filename.empty()) { first_time = false;
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    temp_strm.str("");
    temp_strm << "[Thread" << param.thread_ctr << "] In "exchange_data_with_server': par\
      am.data_filename' is non-empty." << endl << "'param.data_filename' == " <<
      param.data_filename << endl << "Calling send_in_a_loop..." << endl;
    lock_cerr_mutex();
    output_func(temp_strm.str().c_str(), strm_val);
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  in_strm.open(param.data_filename.c_str());
  if (¬(in_strm ∨ in_strm.is_open())) {
    temp_strm.str("");
    temp_strm << "[Thread" << param.thread_ctr << "] ERROR! In "exchange\
      _data_with_server': Failed to open " << "'param.data_filename' == " <<
      param.data_filename << ". " << endl << "Can't read input file." << endl <<
      "Exiting function unsuccessfully with return value 1." << endl;
    lock_cerr_mutex();
    output_func(temp_strm.str().c_str(), strm_val);
    unlock_cerr_mutex();
    return 1;
  } /* if (¬(in_strm ∨ in_strm.is_open())) */
}
```

1560.

```

⟨ exchange_data_with_server definition 1549 ⟩ +≡
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            temp_strm.str("");
            temp_strm << "[Thread" << param.thread_ctr <<
                "] In 'exchange_data_with_server': opened" << "param.data_filename'" == " <<
                param.data_filename << "' " << "successfully." << endl;
            lock_cerr_mutex();
            output_func(temp_strm.str().c_str(), strm_val);
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    read_data = false;
    while (in_strm & ~in_strm.eof()) {
        memset(buffer, 0, BUFFER_SIZE);
        in_strm.read(buffer, BUFFER_SIZE);
        char_ctr = in_strm.gcount();
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            temp_strm.str("");
            temp_strm << "[Thread" << param.thread_ctr << "] In 'exchange_data_with_server': Read\"
                " << char_ctr << "' " << "characters from 'in_strm' ." << endl;
            lock_cerr_mutex();
            output_func(temp_strm.str().c_str(), strm_val);
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        if (char_ctr == 0 & read_data == true) {
#ifndef DEBUG_COMPILE
            if (DEBUG) {
                temp_strm.str("");
                temp_strm << "[Thread" << param.thread_ctr <<
                    "] In 'exchange_data_with_server': 'char_ctr'" == "0" <<
                    "and 'read_data'" == "true' :" << endl << "Breaking." << endl;
                lock_cerr_mutex();
                output_func(temp_strm.str().c_str(), strm_val);
                unlock_cerr_mutex();
            } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
            break;
        } /* if (char_ctr == 0 & read_data == true) */
    else if (char_ctr == 0 & read_data == false) {
        temp_strm.str("");
        temp_strm << "[Thread" << param.thread_ctr <<
            "] ERROR! In 'exchange_data_with_server': 'char_ctr'" == "0" <<
            "and 'read_data'" == "false' :" << endl << "Failed to read from 'in_strm' ." << endl <<
            "Exiting function unsuccessfully with return value 1." << endl;
        lock_cerr_mutex();
        output_func(temp_strm.str().c_str(), strm_val);
        unlock_cerr_mutex();
    }
}

```

```

    in_strm.close();
    return 1;
} /* else if (char_ctr == 0) */
#endif DEBUG_COMPILE
else
{
    if (DEBUG) {
        temp_strm.str("");
        temp_strm << "[Thread" << param.thread_ctr <<
            "] In "exchange_data_with_server": Read" << char_ctr <<
            " characters from "in_strm". " << endl;
        lock_cerr_mutex();
        output_func(temp_strm.str().c_str(), strm_val);
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
if (char_ctr > 0) {
    read_data = true;
    errno = 0;
    if (remote_connection == true) {
        status = gnutls_record_send(param.session, buffer, char_ctr);
    }
    else {
        status = send(param.sock, buffer, char_ctr, 0);
    }
} /* if (char_ctr > 0) */
#endif DEBUG_COMPILE
if (DEBUG) {
    temp_strm.str("");
    temp_strm << "[Thread" << param.thread_ctr <<
        "] In "exchange_data_with_server": send" << returned << status << ". " << endl;
    lock_cerr_mutex();
    output_func(temp_strm.str().c_str(), strm_val);
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
if (status == -1) {
    temp_strm.str("");
    temp_strm << "[Thread" << param.thread_ctr << "] ERROR! In "exchange_data_with_server\
        ": send failed, " << "returning -1. " << endl << "send error: " << strerror(errno) <<
        endl << "Exiting function unsuccessfully with return value 1. " << endl;
    lock_cerr_mutex();
    output_func(temp_strm.str().c_str(), strm_val);
    unlock_cerr_mutex();
    in_strm.close();
    return 1;
}
/* while (in_strm != in_strm.eof) */
in_strm.close();
param.data_filename = ""; /* if (!param.data_filename.empty()) */

```

1561.

```

⟨ exchange_data_with_server definition 1549 ⟩ +≡
else
  if (strlen(param.data_buffer) > 0) {
    first_time = false;
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    temp_strm.str("");
    temp_strm << "[Thread" << param.thread_ctr <<
      "] In 'exchange_data_with_server': param.data_buffer'" <<
      endl << param.data_buffer << endl << "Calling send..." << endl;
    lock_cerr_mutex();
    output_func(temp_strm.str().c_str(), strm_val);
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
errno = 0;
if (remote_connection ≡ true) {
  status = gnutls_record_send(param.session, param.data_buffer, strlen(param.data_buffer));
}
else {
  status = send(param.sock, param.data_buffer, strlen(param.data_buffer), 0);
}
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    temp_strm.str("");
    temp_strm << "[Thread" << param.thread_ctr <<
      "] In 'exchange_data_with_server': send returned" << status << "." << endl;
    lock_cerr_mutex();
    output_func(temp_strm.str().c_str(), strm_val);
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  if (status ≡ -1) {
    temp_strm.str("");
    temp_strm << "[Thread" << param.thread_ctr <<
      "] ERROR! In 'exchange_data_with_server': "
      "'send' failed, returning -1." << endl << "send error:" << strerror(errno) <<
      endl << "Exiting function unsuccessfully with return value 1." << endl;
    lock_cerr_mutex();
    output_func(temp_strm.str().c_str(), strm_val);
    unlock_cerr_mutex();
    return 1;
} /* if (status ≡ -1) */
#endif /* DEBUG_COMPILE */
else
  if (DEBUG) {
    temp_strm.str("");
    temp_strm << "[Thread" << param.thread_ctr << "] In 'exchange_data_with_server': "
      "'send' succeeded, returning" << status << "." << endl;
    lock_cerr_mutex();
    output_func(temp_strm.str().c_str(), strm_val);
  }

```

```

        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    memset(param.data_buffer, 0, BUFFER_SIZE);
} /* else if (strlen(param.data_buffer) > 0) */

```

1562. If *send* sent exactly BUFFER_SIZE – 1 bytes, then we send a single NULL byte, so the server won't block. [LDF 2012.07.27.]

```

⟨ exchange_data_with_server definition 1549 ⟩ +≡
    memset(buffer, 0, BUFFER_SIZE);
    if (status ≡ BUFFER_SIZE) {
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            temp_strm.str("");
            temp_strm ≪ "[Thread" ≪ param.thread_ctr ≪ "] In 'exchange_data_with_server':"
            ≪ "'send' sent exactly " ≪ BUFFER_SIZE " bytes." ≪ endl ≪
            "Sending a single NULL byte, so the server won't block, waiting"
            ≪ "for more bytes." ≪ endl;
            lock_cerr_mutex();
            output_func(temp_strm.str().c_str(), strm_val);
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        if (remote_connection ≡ true) {
            status = gnutls_record_send(param.session, buffer, 1);
        }
        else {
            status = send(param.sock, buffer, 1, 0);
        }
        if (status ≡ -1) {
            temp_strm.str("");
            temp_strm ≪ "[Thread" ≪ param.thread_ctr ≪ "] ERROR! In 'exchange_data_with_server'
            ': 'send' failed, returning -1." ≪ endl ≪ "send error: " ≪ strerror(errno) ≪
            endl ≪ "Exiting function unsuccessfully with return value 1." ≪ endl;
            lock_cerr_mutex();
            output_func(temp_strm.str().c_str(), strm_val);
            unlock_cerr_mutex();
            return 1;
        } /* if (status ≡ -1) */
    } /* if (status ≡ BUFFER_SIZE) */

```

1563.

```

⟨ exchange_data_with_server definition 1549 ⟩ +≡
wrote_data = false; do { errno = 0;
memset(buffer, 0, BUFFER_SIZE);
if (remote_connection ≡ true) {
    status = gnutls_record_recv(param.session, buffer, BUFFER_SIZE);
}
else {
    status = recv(param.sock, buffer, BUFFER_SIZE, 0);
}
if (status > 0) {
#endif DEBUG_COMPILE
    if (DEBUG) {
        temp_strm.str("");
        temp_strm ≪ "[Thread" ≪ param.thread_ctr ≪ "] Received_text_back: " ≪ buffer ≪ endl;
        temp_strm.write(buffer, status);
        temp_strm ≪ endl;
        lock_cerr_mutex();
        output_func(temp_strm.str().c_str(), strm_val);
        unlock_cerr_mutex();
    }
#endif /* DEBUG_COMPILE */
}

```

1564.

```

⟨ exchange_data_with_server definition 1549 ⟩ +≡
else
if (status < 0) {
    temp_strm.str("");
    temp_strm ≪ "[Thread" ≪ param.thread_ctr ≪ "] ERROR! In 'exchange_data_with_server'
        ' : " ≪ "(" ≪ recv_func_name ≪ ')' ≪ failed, returning" ≪ status ≪ "."
        "recv_error:" ≪ strerror(errno) ≪ endl;
    lock_cerr_mutex();
    output_func(temp_strm.str().c_str(), strm_val);
    unlock_cerr_mutex();
    temp_strm.str("");
    temp_strm ≪ "[Thread" ≪ param.thread_ctr ≪ "
        "] Exiting function unsuccessfully with return value 1." ≪ endl;
    lock_cerr_mutex();
    output_func(temp_strm.str().c_str(), strm_val);
    unlock_cerr_mutex();
    return 1;
} /* else if (status < 0) */

```

1565.**Log**

[LDF 2012.09.07.] BUG FIX: Now returning 0 if the server closes the connection. It will do this if *param.server_finished* \equiv true and *param.client_finished* \equiv true in *exchange_data_with_client*.

```
< exchange_data_with_server definition 1549 > +≡
    else /* status ≡ 0 */
    {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        temp_strm.str("");
        temp_strm << "[Thread" << param.thread_ctr << "] In 'exchange_data_with_server':"
            << endl << "Server closed connection." << endl <<
            "Exiting function successfully with return value 0." << endl;
        output_func(temp_strm.str().c_str(), strm_val);
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0;
} /* else (status ≡ 0) */
```

1566.

```

⟨ exchange_data_with_server definition 1549 ⟩ +≡
  if (status == BUFFER_SIZE ∧ wrote_data == false) {
    wrote_data = true;
    strcpy(out_strm_filename, "/tmp/gwirdcli.XXXXXX");
    errno = 0;
    fd = mkstemp(out_strm_filename);
    if (fd == -1) {
      temp_strm.str("");
      temp_strm << "[Thread" << param.thread_ctr << "] ERROR! In 'exchange_data_with_server'
        ': failed, returning -1." << endl << "mkstemp error: "
      strerror(errno) << endl << "Exiting function unsuccessfully with return value 1." <<
      endl;
      lock_cerr_mutex();
      output_func(temp_strm.str().c_str(), strm_val);
      unlock_cerr_mutex();
      return 1;
    } /* if (fd == -1) */
  #if DEBUG_COMPILE
  else
    if (DEBUG) {
      temp_strm.str("");
      temp_strm << "[Thread" << param.thread_ctr <<
        "] In 'exchange_data_with_server': mkstemp succeeded."
      << "'out_strm_filename' == " << " " << out_strm_filename << "."
      << endl;
      lock_cerr_mutex();
      output_func(temp_strm.str().c_str(), strm_val);
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
  #endif /* DEBUG_COMPILE */
  close(fd); /* We just need the name. [LDF 2012.07.30.] */
  fd = 0;
  out_strm.open(out_strm_filename);
  if (¬(out_strm ∧ out_strm.is_open())) {
    temp_strm.str("");
    temp_strm << "[Thread" << param.thread_ctr <<
      "] ERROR! In 'exchange_data_with_server': 'out_strm.open()' failed." << endl <<
      "Failed to open temporary output file " << out_strm_filename << "."
      << endl <<
      "Exiting function unsuccessfully with return value 1." << endl;
    lock_cerr_mutex();
    output_func(temp_strm.str().c_str(), strm_val);
    unlock_cerr_mutex();
    return 1;
  } /* if (¬(out_strm ∧ out_strm.is_open())) */
  #if DEBUG_COMPILE
  else
    if (DEBUG) {
      temp_strm.str("");
      temp_strm << "[Thread" << param.thread_ctr <<
        "] In 'exchange_data_with_server': 'out_strm.open()' succeeded."
      << endl;
      lock_cerr_mutex();
      output_func(temp_strm.str().c_str(), strm_val);
    }
  #endif /* DEBUG_COMPILE */

```

```
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
out_strm.write(buffer, status);
} /* if (status == BUFFER_SIZE & wrote_data == false) */
else if (status > 0 & wrote_data == true) {
    out_strm.write(buffer, status);
} /* else if (status > 0 & wrote_data == true) */
else if (status > 0 & wrote_data == false) {
    ; /* Do nothing. Leave data in buffer. [LDF 2012.07.30.] */
} /* else if (status > 0 & wrote_data == true) */
} /* do */
while (status == BUFFER_SIZE) ;
out_strm.close();
#endif DEBUG_COMPILE
if (DEBUG) {
    temp_strm.str("");
temp_strm << "[Thread" << param.thread_ctr << "] In 'exchange_data_with_server': After \
receive\\loop. status == " << status << "." << endl << "'wrote_data' == " <<
wrote_data << endl << endl << "'buffer' == " << endl;
temp_strm.write(buffer, status);
temp_strm << endl;
lock_cerr_mutex();
output_func(temp_strm.str().c_str(), strm_val);
unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1567. Prepare to call *zzparse*. [LDF 2012.07.30.]

Log

[LDF 2013.05.29.] BUG FIX: Now calling *zzrestart* instead of *zzset_in*. Previously, using *zzset_in* sometimes (but not always!) caused a segmentation fault, apparently because changing the input source caused the address of the lookahead token to become invalid.

```

⟨ exchange_data_with_server definition 1549 ⟩ +≡
    fp = 0;
#ifndef 1 /* 0 */
    bool save_DEBUG = DEBUG;
    DEBUG = false; /* true */
#endif
    if (wrote_data ≡ true) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ "wrote_data==true." ≪ endl ≪ "out_strm_filename==" ≪ out_strm_filename ≪
            endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    fp = fopen(out_strm_filename, "r");
    if (fp ≡ 0) {
        lock_cerr_mutex();
        cerr ≪ "[Thread" ≪ param.thread_ctr ≪ "] ERROR! In 'excha\
nge_data_with_server': 'fopen' failed, returning NULL." ≪ endl ≪
            "Failed to open input file " ≪ out_strm_filename ≪ "." ≪ endl ≪
            "Exiting function unsuccessfully with return value 1." ≪ endl;
        unlock_cerr_mutex();
        return 1;
    } /* if (fp ≡ 0) */
    else if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ "[Thread" ≪ param.thread_ctr ≪ "] In 'exchange_data_with_server': " ≪
            "Calling 'zzrestart' (reading from input file " ≪ out_strm_filename ≪ ")" . " ≪ endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
    zzrestart(fp, parameter);
#endif 0
    zzset_in(fp, parameter); /* This caused a segmentation fault, but not always. Now calling zzrestart
        instead. [LDF 2013.05.29.] */
#endif
    } /* if (wrote_data ≡ true) */
    else {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ "wrote_data==false." ≪ endl;
        cerr ≪ "buffer==" ≪ endl ≪ buffer ≪ endl;
        unlock_cerr_mutex();
    }
}

```

```

    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    zz_buffer_state = zz_scan_string(buffer, parameter);
}

```

1568. Call *zzparse*. [LDF 2012.07.30.]
 ⟨ *exchange_data_with_server* definition 1549 ⟩ +≡
 status = zzparse(parameter);

1569.

Log

[LDF 2013.04.18.] Added this section.

```

⟨ exchange_data_with_server definition 1549 ⟩ +≡
if (status ≡ 2) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param.thread_ctr << "] In 'exchange_data_with_server': 'zzp\
            arse' succeeded, returning 2:" << endl << "'END_SERVER' command succeeded." <<
            endl << "Exiting function successfully with return value 2." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    if (fp ≡ 0) zz_delete_buffer(zz_buffer_state, parameter);
    else {
        fclose(fp);
        fp = 0;
    }
    return 2;
} /* if (status ≡ 2) */

```

1570.

```

⟨ exchange_data_with_server definition 1549 ⟩ +≡
else
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << "[Thread" << param.thread_ctr << "] ERROR! In 'exchange_data_with_server'\
        ': 'zzparse' failed, returning" << status << "." << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    if (fp ≡ 0) zz_delete_buffer(zz_buffer_state, parameter);
    else {
        fclose(fp);
        fp = 0;
    }
    return 1;
} /* else if (status ≠ 0) */

```

1571.

```

⟨ exchange_data_with_server definition 1549 ⟩ +≡
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "[Thread" << param.thread_ctr << "] In 'exchange_data_with_server': zzb\
arise' succeeded, returning 0." << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    if (fp == 0) zz_delete_buffer(zz_buffer_state, parameter);
    else {
        fclose(fp);
        fp = 0;
    }
#endif /* 0 */
    DEBUG = save_DEBUG;
#endif
} /* for */

```

1572.

```

⟨ exchange_data_with_server definition 1549 ⟩ +≡
    return 0; } /* End of exchange_data_with_server definition */

```

1573. External function declarations.

```

⟨ External function declarations 32 ⟩ +≡
    int zzparse(yyscan_t parameter);

```

1574.

```

⟨ Garbage 303 ⟩ +≡

```

1575. Putting exchnsrv.web together.**1576.** This is what's compiled.

```

⟨ Include files 3 ⟩
using namespace std;
using namespace gwrdifpk;
⟨ External function declarations 32 ⟩
extern int zzdebug;
int output_func(string, bool b = false);
⟨ exchange_data_with_server declaration 1548 ⟩
⟨ exchange_data_with_server definition 1549 ⟩
#endif 0
⟨ Garbage 303 ⟩
#endif

```

1577. This is what's written to the header file `exchnsrv.h`. [LDF 2012.06.27.]

```
{ exchnsrv.h 1577 } ≡
#ifndef EXCHNSRV_H
#define EXCHNSRV_H 1
    using namespace std;
    using namespace gwrdifpk;
    { exchange_data_with_server declaration 1548 }
#endif
```

1578. PID functions (pidfncts.web).

Log

[LDF 2012.07.19.] Added this file.

1579. Include files.

```
<Include files 3> +≡
#ifndef _XOPEN_SOURCE
#define _XOPEN_SOURCE
#endif
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <dirent.h>
#include <errno.h>
#include <fcntl.h>
#include <limits.h>
#include <pthread.h>
#include <pwd.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <string>
#include <sstream>
#include <deque>
#include <map>
#include <utility>
#include <vector>
#include <set>
#if HAVE_CONFIG_H
#include <config.h>
#endif
#include <mysql.h>
#include "glblcnst.h++"
#include "glblvrb1.h++"
#include "excptntp.h++"
#include "utilfncs.h++"
#include "hdlvltp.h++"
#include "rspnntp.h++"
#include "irdsavtp.h++"
#include "irdsobtp.h++"
#include "hdltype.h++"
```

1580. Generate PIDs. [LDF 2012.09.28.]**Log**

[LDF 2012.09.28.] Added this function.
 [LDF 2012.10.11.] Added optional argument `vector<string> *pid_vector_ptr` with default 0.
 [LDF 2012.10.16.] Added code for setting `handles.handle_id` and `handles.handle_value_id`.
 [LDF 2013.01.08.] Added optional arguments `vector<unsigned long int> *handle_id_vector_ptr = 0` and `vector<unsigned long int> *handle_value_id_vector_ptr = 0`. If they are non-null, the generated values of `handle_id` and `handle_value_id` are pushed onto them.
 [LDF 2013.02.13.] Modified to account for changing the name of the `handlesystem_standalone.pid_counter` database table to `pid_counters` and adding the `prefix` field to it.
 [LDF 2013.02.15.] Added optional argument `string fifo_pathname = ""`.
 [LDF 2013.02.22.] Added optional arguments `long int user_id = 0` and `string username = ""`.

(generate_pids declaration 1580) ≡

```
int generate_pids(MYSQL *mysql_ptr, string prefix_str, string &pid_str, vector<string>
  *pid_vector_ptr = 0, unsigned int number_of_pids = 1, vector<unsigned long int>
  *handle_id_vector_ptr = 0, vector<unsigned long int> *handle_value_id_vector_ptr = 0, bool
  standalone_hs = true, string institute_str = "", string suffix_str = "", vector<Handle_Type>
  *handle_vector = 0, string fifo_pathname = "", long int user_id = 0, string username = "");
```

This code is used in sections 1624 and 1625.

1581.

(generate_pids definition 1581) ≡

```
int generate_pids(MYSQL *mysql_ptr, string prefix_str, string &pid_str, vector<string>
  *pid_vector_ptr, unsigned int number_of_pids, vector<unsigned long int>
  *handle_id_vector_ptr, vector<unsigned long int> *handle_value_id_vector_ptr, bool
  standalone_hs, string institute_str, string suffix_str, vector<Handle_Type>
  *handle_vector, string fifo_pathname, long int user_id, string username){ bool DEBUG = false;
  /* true */
  set_debug_level(DEBUG, 0, 0);
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "Entering 'generate_pids'." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

See also sections 1582, 1583, 1584, 1585, 1586, 1587, 1588, 1589, 1590, 1591, 1592, 1593, 1594, 1595, 1596, 1597, 1598, 1599, 1600, 1601, 1602, 1603, 1604, 1605, 1606, 1607, 1608, 1609, 1610, 1611, 1612, 1613, 1614, 1615, 1616, 1617, and 1618.

This code is used in section 1624.

1582. $\text{ULONG_MAX} \equiv 2^{64} - 1 \equiv 18446744073709551615$ on both a 32-bit and a 64-bit machine I've tested. This value is also the maximum value for the `unsigned bigint` data type in MySQL, so I think we're on the safe side using `unsigned long` for `pid_ctr`. [LDF 2012.09.28.]

Log

[LDF 2013.01.11.] Now declaring `Handle_Value_Type curr_handle`.

[LDF 2013.02.27.] Changed `Handle_Type curr_handle` to `Handle_Value_Type curr_handle_value`. Added ~~Handle_Type curr_handle~~.

```
<generate_pids definition 1581> +≡
int status;
stringstream sql_strm;
stringstream sql_strm_1;
stringstream sql_strm_2;
MYSQL_RES *result = 0;
MYSQL_ROW curr_row;
unsigned int row_ctr = 0;
unsigned int field_ctr = 0;
long affected_rows = 0;
bool failed = false;
unsigned long pid_ctr;
string handles_table_name = (standalone_hs) ? "handlesystem_standalone.handles" :
    "handlesystem.handles";
string comma_str = "";
string or_str = "";
size_t pos;
stringstream temp_strm;
unsigned long int handle_id = 0;
unsigned long int handle_value_id = 0;
string temp_str;
Handle_Type curr_handle;
Handle_Value_Type curr_handle_value;
int fd;
int ret_val = 0;
```

1583. Select `handlesystem` or `handlesystem_standalone` database and lock tables. [LDF 2012.09.28.] [LDF 2012.10.16.]

The `admin_data` table is needed to avoid problems with subqueries when using the MySQL C-API. [LDF 2012.10.05.]

Log

[LDF 2012.10.16.] Now locking the `pid_counter` table. I've moved it to the `handlesystem_standalone` database to avoid problems locking tables in different databases simultaneously.

[LDF 2013.01.31.] Now locking the `handle_value_counter` table.

```

⟨ generate_pids definition 1581 ⟩ +≡
  if (standalone_hs) status = mysql_select_db(mysql_ptr, "handlesystem_standalone");
  else status = mysql_select_db(mysql_ptr, "handlesystem");
  if (status ≡ 0) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "In 'generate_pids': " << "'mysql_select_db' succeeded'." << endl;
      unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  } /* if (status ≡ 0) */
  else /* status ≠ 0 */
  {
    lock_cerr_mutex();
    cerr << "ERROR! In 'generate_pids': " << "'mysql_select_db' failed, returning" <<
          status << endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    return 1;
  } /* else (status ≠ 0) */
sql strm << "lock_tables_handles_write, admin_data_write, "
      "pid_counters_write, gwidrsif.Prefixes_read, gwidrsif.Users_read";
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "sql strm.str() == " << sql strm.str() << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  status = submit_mysql_query(sql strm.str(), result, mysql_ptr);
  if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'generate_pids':"
        "submit_mysql_query' failed, returning" << status << ":" << endl <<
        mysql_error(mysql_ptr) << endl << "Failed to lock";
    if (standalone_hs) cerr << "'handlesystem_standalone.handles', "
        "'handlesystem_standalone.admin_data' and "
        "'handlesystem_standalone.pid_counters' database tables." << endl;
  else cerr << "'handlesystem.handles'" << "'handlesystem.admin_data', "
        "'handlesystem.admin_data' database tables and "
        "'handlesystem.pid_counters' database tables." << endl;
    cerr << "Exiting function unsuccessfully with return value 1." << endl;

```

```
unlock_cerr_mutex();
if (result) mysql_free_result(result);
return 1;
} /* if (status != 0) */
#ifndef DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'generate_pids': Locked";
    if (standalone_hs) cerr << "'handlesystem_standalone.handles'";
    else cerr << "'handlesystem.handles'";
    cerr << "and 'admin_data' database tables successfully." << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
sqlstrm.str("");
mysql_free_result(result);
result = 0;
```

1584. If *user_id* ≤ 0 and *username* is non-empty, get *user_id*. [LDF 2013.02.22.]

Log

[LDF 2013.02.22.] Added this section.

```

⟨ generate_pids definition 1581 ⟩ +≡
  if (user_id ≤ 0 ∧ ¬username.empty()) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'generate_pids': user_id == " << user_id <<
      " (<= 0)" << endl << "'username' == " << username << endl <<
      "Will fetch 'user_id' from 'gwirdsif.Users' database table." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  sql_strm.str("");
  sql_strm << "select user_id from gwirdsif.Users where username = " << username << " ";
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'generate_pids': sql_strm.str() == " << sql_strm.str() << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  status = submit_mysql_query(sql_strm.str(), result, mysql_ptr, &row_ctr, &field_ctr);
  if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'generate_pids':"
      << endl <<
      "'submit_mysql_query' failed, returning " << status << ":" << endl <<
      mysql_error(mysql_ptr) << endl;
    unlock_cerr_mutex();
    failed = true;
    ret_val = 1;
    goto UNLOCK_TABLES;
  } /* if (status ≠ 0) */
#endif DEBUG_COMPILE
  else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'generate_pids':"
      << endl << "'submit_mysql_query' succeeded, returning "
      "0." << endl << "'row_ctr' == " << row_ctr << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  if (row_ctr ≡ 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'generate_pids':"
      << endl <<
      "'row_ctr' == 0. Failed to retrieve 'gwirdsif.Users.user_id' for "
      "'username' == " << username << "." << endl;
    unlock_cerr_mutex();
    failed = true;
  }
}

```

```

ret_val = 1;
goto UNLOCK_TABLES;
} /* if (row_ctr == 0) */
else if (row_ctr > 1) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'generate_pids': " << endl <<
        "'row_ctr' > 1. Retrieved multiple values of 'gwirdsif.Users.user_id' for "
        "'username'" << username << "." << endl << "This shouldn't happen." << endl;
    unlock_cerr_mutex();
    failed = true;
    ret_val = 1;
    goto UNLOCK_TABLES;
} /* if (row_ctr > 1) */
else if (row_ctr == 1) {
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'generate_pids': " << endl << "'row_ctr' == 1. Retrieved 'gwirdsif.Us"
            "ers.user_id' for " << "'username'" << username << "." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
}

```

1585.

```

⟨ generate_pids definition 1581 ⟩ +≡
curr_row = mysql_fetch_row(result);
if (curr_row == 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'generate_pids': " << endl <<
        "'mysql_fetch_row' failed, returning NULL: " << endl << "MySQL error: " <<
        mysql_error(mysql_ptr) << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    result = 0;
    failed = true;
    ret_val = 1;
    goto UNLOCK_TABLES;
} /* if (curr_row == 0) */

```

1586.

```

⟨ generate_pids definition 1581 ⟩ +≡
else {
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'generate_pids': " << endl << "'mysql_fetch_row' succeeded. " << endl <<
            "'curr_row[0]' == " << curr_row[0] << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
}

```

1587.

```
<generate_pids definition 1581> +≡
  user_id = strtol(curr_row[0], 0, 10);
  if (user_id ≡ LONG_MAX ∨ user_id ≡ LONG_MIN) {
    lock_cerr_mutex();
    cerr ≪ "ERROR! In 'generate_pids': " ≪ endl ≪ "'strtol' failed, returning";
    if (user_id ≡ LONG_MAX) cerr ≪ "'LONG_MAX': " ≪ endl;
    else cerr ≪ "'LONG_MIN': " ≪ endl;
    cerr ≪ strerror(errno) ≪ endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    result = 0;
    failed = true;
    ret_val = 1;
    goto UNLOCK_TABLES;
} /* if (user_id ≡ LONG_MAX ∨ user_id ≡ LONG_MIN) */
```

1588.

```
<generate_pids definition 1581> +≡
  if (user_id > INT_MAX) {
    lock_cerr_mutex();
    cerr ≪ "ERROR! In 'generate_pids': " ≪ endl ≪ "'user_id' == " ≪ user_id ≪
      " (> 'INT_MAX') " ≪ endl ≪ " This is not permitted. " ≪ endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    result = 0;
    failed = true;
    ret_val = 1;
    goto UNLOCK_TABLES;
} /* if (user_id > INT_MAX) */
```

1589.

```
<generate_pids definition 1581> +≡
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "In 'generate_pids': " ≪ endl ≪ "'user_id' == " ≪ user_id ≪ endl ≪
      "'username' == " ≪ username ≪ endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* else */
} /* if (row_ctr ≡ 1) */
mysql_free_result(result);
result = 0;
sql_strm.str(""); /* if (user_id ≤ 0 ∧ !username.empty()) */
```

1590.**Log**

[LDF 2013.02.22.] Added this section.

```
<generate_pids definition 1581> +≡
else
    if (user_id > 0) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'generate_pids': user_id == " << user_id << endl;
        if (!username.empty()) cerr << "'username' == " << username << endl;
        else cerr << "'username' is empty." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* else if (user_id > 0) */
```

1591.

Log

[LDF 2012.10.16.] Added this section.

[LDF 2013.01.31.] Added error-handling code for the case that *strtoul* returns `ULONG_MAX` when setting *handle_id* and *handle_value_id*.

```

⟨ generate_pids definition 1581 ⟩ +≡
sql_strm.str("") ;
temp_str = "select_handle_id from_handles_order_by_handle_id_desc_limit_1";
status = submit_mysql_query(temp_str, result, mysql_ptr, &row_ctr, &field_ctr);
if (status ≡ 0 ∧ (curr_row = mysql_fetch_row(result)) ≠ 0 ∧ row_ctr ≡ 1 ∧ curr_row[0] ∧ strlen(curr_row[0]))
{
    handle_id = strtoul(curr_row[0], 0, 10);
    if (handle_id ≡ ULONG_MAX) {
        lock_cerr_mutex();
        cerr << "ERROR! In 'generate_pids': 'strtoul' failed, returning 'ULONG_MAX'." <<
            endl << "Failed to retrieve 'handle_id'." << endl <<
            "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        failed = true;
        ret_val = 1;
        goto UNLOCK_TABLES;
    } /* if (handle_id ≡ ULONG_MAX) */
    ++handle_id;
#endif /* DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "handle_id==" << handle_id << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
mysql_free_result(result);
result = 0;
}
else {
    if (result) mysql_free_result(result);
    result = 0;
    lock_cerr_mutex();
    cerr << "ERROR! In 'generate_pids': Failed to retrieve 'handle_id'." <<
        endl << "MySQL_error:" << mysql_error(mysql_ptr) << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    failed = true;
    ret_val = 1;
    goto UNLOCK_TABLES;
} /* else */
mysql_free_result(result);
result = 0;

```

1592.

```

⟨ generate_pids definition 1581 ⟩ +≡
sql_strm.str("");
temp_str = "select_handle_value_id_from_handles_order_by_handle_value_id_desc_limit_1";
status = submit_mysql_query(temp_str, result, mysql_ptr, &row_ctr, &field_ctr);
if (status ≡ 0 ∧ (curr_row = mysql_fetch_row(result)) ≠ 0 ∧ row_ctr ≡ 1 ∧ curr_row[0] ∧ strlen(curr_row[0]))
{
    handle_value_id = strtoul(curr_row[0], 0, 10);
    if (handle_value_id ≡ ULONG_MAX) {
        lock_cerr_mutex();
        cerr ≪ "ERROR! In 'generate_pids': strtoul failed, returning ULONG_MAX." ≪
            endl ≪ "Failed to retrieve 'handle_value_id'." ≪ endl ≪
            "Exiting function unsuccessfully with return value 1." ≪ endl;
        unlock_cerr_mutex();
        failed = true;
        ret_val = 1;
        goto UNLOCK_TABLES;
    } /* if (handle_value_id ≡ ULONG_MAX) */
    ++handle_value_id;
#endif /* DEBUG_COMPILE */
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "handle_value_id==" ≪ handle_value_id ≪ endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
mysql_free_result(result);
result = 0;
}
else {
    if (result) mysql_free_result(result);
    result = 0;
    cerr ≪ "ERROR! In 'generate_pids': Failed to retrieve 'handle_value_id'." ≪
        endl ≪ "MySQL error:" ≪ mysql_error(mysql_ptr) ≪ endl ≪
        "Exiting function unsuccessfully with return value 1." ≪ endl;
    unlock_cerr_mutex();
    failed = true;
    ret_val = 1;
    goto UNLOCK_TABLES;
} /* else */
mysql_free_result(result);
result = 0;

```

1593.

```
<generate_pids definition 1581> +≡
  if (¬pid_str.empty()) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "In 'generate_pids': pid_str is not empty:" ≪ pid_str ≪ endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  string temp_prefix_str = pid_str;
  pos = temp_prefix_str.find("/");
}
```

1594.

```

⟨ generate_pids definition 1581 ⟩ +≡
  if (pos ≠ string::npos) {
    temp_prefix_str.erase(pos);
    pid_str.erase(0, pos + 1);
#if DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'generate_pids': 'pid_str' contained a prefix." << endl <<
      "'temp_prefix_str' == " << temp_prefix_str << endl << "'pid_str' == " << pid_str << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  if (prefix_str.empty()) {
#if DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'generate_pids': 'prefix_str' is empty" <<
      "and 'temp_prefix_str' is non-empty:" << endl << "'temp_prefix_str' == " <<
      temp_prefix_str << endl << "Setting 'prefix_str' to 'temp_prefix_str'." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  prefix_str = temp_prefix_str;
} /* if (prefix_str.empty()) */
else if (!prefix_str.empty() ∧ prefix_str ≠ temp_prefix_str) {
  lock_cerr_mutex();
  cerr << "ERROR! In 'generate_pids': 'prefix_str' is non-empty" <<
    "and doesn't match 'temp_prefix_str':" << endl << "'prefix_str' == " << prefix_str <<
    endl << "Exiting function unsuccessfully with return value 1." << endl;
  unlock_cerr_mutex();
  failed = true;
  ret_val = 1;
  goto UNLOCK_TABLES;
} /* else if (!prefix_str.empty() ∧ prefix_str ≠ temp_prefix_str) */
#endif DEBUG_COMPILE
else
  if (DEBUG ∧ !prefix_str.empty() ∧ prefix_str ≡ temp_prefix_str) {
    lock_cerr_mutex();
    cerr << "'generate_pids': 'prefix_str' is non-empty" <<
      "and matches 'temp_prefix_str':" << endl <<
      "'prefix_str' == 'temp_prefix_str'" << endl <<
      "No need to do anything." << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG ∧ !prefix_str.empty() ∧ prefix_str ≡ temp_prefix_str) */
#endif /* DEBUG_COMPILE */
} /* if (pos ≠ string::npos) */

```

1595.

```
< generate_pids definition 1581 > +≡
else
  if (prefix_str.empty()) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'generate_pids': 'pid_str' does not contain a prefix" <<
      "and 'prefix_str' is empty." << endl << "Exiting function unsuccessfully with re\"
      turn_value_1." << endl;
    unlock_cerr_mutex();
    failed = true;
    ret_val = 1;
    goto UNLOCK_TABLES;
} /* else if (prefix_str.empty()) */
```

1596.

```
< generate_pids definition 1581 > +≡
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "pid_str==" << pid_str << endl << "prefix_str==" << prefix_str << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1597.

```

⟨ generate_pids definition 1581 ⟩ +≡
unsigned int prefix_id;
status = check_prefix(mysql_ptr, prefix_str, prefix_id);
if (status ≡ 1) {
    lock_cerr_mutex();
    cerr ≪ "ERROR! In 'generate_pids': 'check_prefix' failed, " ≪ "returning 1:" ≪
        endl ≪ "MySQL error: query or fetch row failed for " ≪ "Prefix " ≪ prefix_str ≪
            ". " ≪ endl;
    unlock_cerr_mutex();
    failed = true;
    ret_val = 1;
    goto UNLOCK_TABLES;
} /* if (status ≡ 1) */
else if (status ≡ 2) {
    lock_cerr_mutex();
    cerr ≪ "ERROR! In 'generate_pids': 'check_prefix' failed, " ≪ "returning 2:" ≪
        endl ≪ "Prefix " ≪ prefix_str ≪ "' not found." ≪ endl;
    unlock_cerr_mutex();
    failed = true;
    ret_val = 2;
    goto UNLOCK_TABLES;
} /* if (status ≡ 2) */
else if (status ≡ 3) {
    lock_cerr_mutex();
    cerr ≪ "ERROR! In 'generate_pids': 'check_prefix' failed, " ≪ "returning 3:" ≪
        endl ≪ "Prefix " ≪ prefix_str ≪ "' ID " ≪ prefix_id ≪ ")" ≪ " is disabled." ≪ endl;
    unlock_cerr_mutex();
    failed = true;
    ret_val = 3;
    goto UNLOCK_TABLES;
} /* if (status ≡ 3) */
else if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ "ERROR! In 'generate_pids': 'check_prefix' failed, " ≪ "returning " ≪ status ≪
        ". " ≪ endl;
    unlock_cerr_mutex();
    failed = true;
    ret_val = status;
    goto UNLOCK_TABLES;
} /* if (status ≠ 0) */
#if DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ "In 'generate_pids': 'check_prefix' succeeded, returning 0." ≪ endl ≪
            "'prefix_id' == " ≪ prefix_id ≪ endl ≪ "'prefix_str' == " ≪ prefix_str ≪ endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1598.**Log**

[LDF 2013.02.13.] BUG FIX: Now removing slash and anything following it from *prefix_str* here. Previously, this was done much further below.

```
< generate_pids definition 1581 > +≡
pos = prefix_str.find("/");
/* If prefix_str contains a slash, remove it and anything that follows it. [LDF 2012.10.08.] */
if (pos != string::npos) prefix_str.erase(pos);
pid_str.insert(0, "/");
pid_str.insert(0, prefix_str);
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'generate_pids': pid_str' after prepending prefix: " << pid_str << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1599. *pid_str* is not empty. Set *sql_strm_1* and *sql_strm_2*. Only a single handle is created (possibly with multiple handle values). [LDF 2013.02.22.]

Log

[LDF 2013.01.14.] BUG FIX: Now using *time(0)* to set *timestamp* value instead of *now()*.

```
< generate_pids definition 1581 > +≡
sql_strm_1 << "insert into " << handles_table_name << "(handle, idx, type, "
<< "ttl_type, ttl, timestamp, refs, admin_read, admin_write, "
<< "pub_read, pub_write, handle_id, handle_value_id, created, "
<< "created_by_user_id)" << "values" << "(" << pid_str << ',', 300, 'HS_ADMIN', 
<< "0, 86400, " << time(0) << ",", 1, 1, 1, 0, " << handle_id << ", " << handle_value_id <<
", now(), " << user_id << ")";
sql_strm_2 << "update handles set data = (select data from admin_data where handle =
'= " << prefix_str << "/ADMIN_SAVE") where handle = " << pid_str <<
" and type = 'HS_ADMIN'";
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'generate_pids': pid_str is not empty. Skipping to "
        "'END_CREATE_PID_STR'." << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
if (pid_vector_ptr) pid_vector_ptr->push_back(pid_str);
if (handle_id_vector_ptr) handle_id_vector_ptr->push_back(handle_id);
if (handle_value_id_vector_ptr) handle_value_id_vector_ptr->push_back(handle_value_id);
++handle_id;
++handle_value_id;
goto END_CREATE_PID_STR; /* if (!pid_str.empty()) */
```

1600.

```

⟨ generate_pids definition 1581 ⟩ +≡
sql_strm.str("");
sql_strm << "select pid_counter from pid_counters" << " where prefix ='" << prefix_str <<
"' " << "order by pid_counter desc limit 1";
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'generate_pids': sql_strm.str() == " << endl << sql_strm.str() << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
status = submit_mysql_query(sql_strm.str(), result, mysql_ptr, &row_ctr, &field_ctr);
if (status != 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'generate_pids': submit_mysql_query failed, returning " <<
        status << ". " << endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    failed = true;
    ret_val = 1;
    goto UNLOCK_TABLES;
} /* if (status != 0) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'generate_pids': submit_mysql_query succeeded, returning 0. " << endl <<
            "'row_ctr' == " << row_ctr << endl << "'field_ctr' == " << field_ctr << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1601.

```
< generate_pids definition 1581 > +≡
  if ((curr_row = mysql_fetch_row(result)) == 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'generate_pids': mysql_fetch_row failed:" << endl <<
      mysql_error(mysql_ptr) << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    result = 0;
    failed = true;
    ret_val = 1;
    goto UNLOCK_TABLES;
} /* if (curr_row = mysql_fetch_row(result) == 0) */
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "In 'generate_pids': mysql_fetch_row succeeded." << endl <<
        "curr_row[0]==" << curr_row[0] << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1602.

```

⟨ generate_pids definition 1581 ⟩ +≡
errno = 0;
pid_ctr = strtoul(curr_row[0], 0, 10);
if (errno ≠ 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'generate_pids': strtoul' failed:" << endl << "'errno'" <<
        errno << endl << "stroulerror:" << strerror(errno) << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    result = 0;
    ret_val = 1;
    failed = true;
    goto UNLOCK_TABLES;
} /* if (errno ≠ 0) */
else if (pid_ctr ≡ ULONG_MAX) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'generate_pids': strtoul' returned 'ULONG_MAX'." << endl <<
        "Either the maximum value for PID counter has been reached, or, more" <<
        "probably, something has gone wrong." << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    result = 0;
    failed = true;
    ret_val = 1;
    goto UNLOCK_TABLES;
} /* else if (pid_ctr ≡ ULONG_MAX) */
#if DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'generate_pids': strtoul' returned" << pid_ctr << "." << endl <<
            "'pid_ctr'" << pid_ctr << endl << "Will increment and create PID string." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1603.

```

⟨ generate_pids definition 1581 ⟩ +≡
mysql_free_result(result);
result = 0;
sql_strm.str("");
sql_strm << "update pid_counters set pid_counter = " << pid_ctr + number_of_pids <<
    " where prefix = " << prefix_str << "";
#endif /* DEBUG_COMPILE */
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'generate_pids': sql_strm.str() == " << endl <<
        sql_strm.str() << endl << "'pid_ctr'(hex) == " << hex << pid_ctr << endl <<
        "'pid_ctr+number_of_pids'(hex) == " << hex << (pid_ctr + number_of_pids) << dec << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
status = submit_mysql_query(sql_strm.str(), result, mysql_ptr, 0, 0, &affected_rows);
if (status != 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'generate_pids': 'submit_mysql_query' failed, returning " <<
        status << ". " << endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    failed = true;
    ret_val = 1;
    goto UNLOCK_TABLES;
} /* if (status != 0) */
#endif /* DEBUG_COMPILE */
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'generate_pids': 'submit_mysql_query' succeeded, returning 0. " << endl <<
        "'affected_rows' == " << affected_rows << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
mysql_free_result(result);
result = 0;
sql_strm.str(""); END_GET_PID_COUNTER:

```

1604.**Log**

[LDF 2013.02.22.] BUG FIX: Now using *prefix_str* in *sql_strm_2*.

```
< generate_pids definition 1581 > +≡
temp_strm.str("");
sql_strm_1 ≪ "insertinto" ≪ handles_table_name ≪ "(handle, idx, type, " ≪
"ttl_type, ttl, timestamp, refs, admin_read, admin_write," ≪
"pub_read, pub_write, handle_id, handle_value_id, created," ≪
"created_by_user_id)" ≪ "values";
sql_strm_2 ≪ "updatehandlessetdata=(selectdatafromadmin_datawherehandle" ≪
"='";
if (!prefix_str.empty()) sql_strm_2 ≪ prefix_str;
else sql_strm_2 ≪ "12345";
sql_strm_2 ≪ "/ADMIN_SAVE'"where";
```

1605. Loop for creating PIDs. [LDF 2012.10.11.]

```
< generate_pids definition 1581 > +≡
for (int i = 0; i < number_of_pids; ++i) { ++pid_ctr;
curr_handle_value.clear();
curr_handle.clear();
```

1606. Create PID string. [LDF 2012.09.28.]**Log**

[LDF 2012.10.05.] Now using a counter string in hexadecimal notation of at least 5 characters. If the value of the counter is > *FFFFF*, then more characters are used, but the string is not broken up with hyphens. Previously, the pattern used was meant to appear like the PIDs from the GWDG Handle Service (EPIC-API-v1), e.g., "11858/00-ZZZZ-0000-0000-0228-2", or using a suffix, "11858/00-ZZZZ-0000-0000-1458-E-LDF-A".

[LDF 2013.01.14.] Now using *time(0)* to set the *timestamp* value instead of using 1348833495.

```
<generate_pids definition 1581> +≡
pid_str = "";
temp_strm.str("");
temp_strm << setw(5) << setfill('0') << uppercase << hex << pid_ctr;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "temp_strm.str() == " << temp_strm.str() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
pid_str = temp_strm.str();
temp_strm.str("");
temp_strm.clear();
temp_strm << prefix_str << "/";
if (!institute_str.empty()) temp_strm << institute_str << "-";
temp_strm << pid_str;
if (!suffix_str.empty()) temp_strm << "-" << suffix_str;
pid_str = temp_strm.str();
if (pid_vector_ptr) pid_vector_ptr->push_back(pid_str);
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "pid_str == " << pid_str << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
sql_strm_1 << comma_str << "(" << pid_str << ",_300,_HS_ADMIN,_" << "0,_86400,_" << time(0) <<
    ",,_1,_1,_0,_" << handle_id << ",,_" << handle_value_id << ",,_now(),,_" << user_id << ")";
comma_str = ",,_";
sql_strm_2 << or_str << "(handle=_" << pid_str << ",_and_type=_'HS_ADMIN')";
or_str = "_or_";
if (handle_id_vector_ptr) handle_id_vector_ptr->push_back(handle_id);
if (handle_value_id_vector_ptr) handle_value_id_vector_ptr->push_back(handle_value_id);
```

1607. If *handle_vector* is non-null, set *curr_handle_value*, insert *curr_handle_value* into *curr_handle.handle_value_map* and push *curr_handle* onto *handle_vector*. [LDF 2013.01.11.] [LDF 2013.02.27.]

Log

[LDF 2013.01.11.] Added this section.
 [LDF 2013.02.27.] Now using *curr_handle* and *handle_vector*.

```
<generate_pids definition 1581> +≡
  if (handle_vector ≠ 0) { status = curr_handle_value.set(pid_str, 300, "HS_ADMIN", admin_data,
    admin_data_length, 0, 86400, time(0), 0, 0, 1, 1, 1, 0, user_id, handle_id, handle_value_id, false, time(0), 0);
  if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'generate_pids': Handle_Value_Type::set' failed, returning " <<
      status << "." << endl << "Will try to unlock_database_tables before " <<
        "existing function unsuccessfully." << endl;
    unlock_cerr_mutex();
    failed = true;
    ret_val = 1;
    goto UNLOCK_TABLES;
  } /* if (status ≠ 0) */
#endif /* DEBUG_COMPILE
else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'generate_pids': Handle_Value_Type::set' succeeded, returning 0." << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1608.

Log

[LDF 2013.01.14.] BUG FIX: Now setting *curr_handle_value.data* and *curr_handle_value.refs* to 0. This avoids “corruption or double free” when the **Handle_Value_Type** destructor is called on the elements of *handle_value_vector*.

[LDF 2013.01.31.] BUG FIX: Now incrementing *handle_id* and *handle_value_id* here. Previously, they were incremented before calling **Handle_Value_Type::set** on *curr_handle_value*.

[LDF 2013.02.27.] BUG FIX: Now setting *curr_handle.handle* and *curr_handle.handle_id*.

```
<generate_pids definition 1581> +≡
  curr_handle.handle = curr_handle_value.handle;
  curr_handle.handle_id = curr_handle_value.handle_id;
  curr_handle.handle_value_map[curr_handle_value.idx] = curr_handle_value;
  handle_vector.push_back(curr_handle);
  ++handle_id;
  ++handle_value_id;
  curr_handle_value.data = 0;
  curr_handle_value.refs = 0; } /* if (handle_vector ≠ 0) */
} /* for */
END_CREATE_PID_STR:
```

1609. Insert rows for the handles into the `handles` table, so that the names are reserved. A single handle value is created with type `HS_ADMIN`. !! PLEASE NOTE: This code assumes that the handle `<prefix>/ADMIN_SAVE` exists and that there is a handle value for this handle with index 100 and type `HS_ADMIN` containing valid data. This handle value is copied (except for the `handle` field, of course) for the new handle, so that the latter has an administrator. This makes it possible to access it using the means provided by the Handle System. Otherwise, it would only be accessible by using the means provided by the database software.

The best way to generate the handle `<prefix>/ADMIN_SAVE` is by using `hdl-admintool`. [LDF 2012.09.28.]

```

⟨ generate_pids definition 1581 ⟩ +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "sql_strm_1.str() == " << sql_strm_1.str() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
status = submit_mysql_query(sql_strm_1.str(), result, mysql_ptr, 0, 0, &affected_rows);
if (status != 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'generate_pids': " << endl <<
        "'submit_mysql_query' failed, returning " << status << ":" << endl <<
        mysql_error(mysql_ptr) << endl << "Failed to insert rows for PIDs into ";
    if (standalone_hs) cerr << "'handlesystem_standalone.handles'";
    else cerr << "'handlesystem.handles'";
    cerr << "database_table." << endl << "Exiting function unsuccessfully with re\
        turn_value_1." << endl;
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    result = 0;
    failed = true;
    ret_val = 1;
    goto UNLOCK_TABLES;
} /* if (status != 0) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'generate_pids': Inserted rows for PIDs into ";
        if (standalone_hs) cerr << "'handlesystem_standalone.handles'";
        else cerr << "'handlesystem.handles'";
        cerr << "table successfully." << endl << "'affected_rows' == " << affected_rows << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
sql_strm_1.str("");
mysql_free_result(result);
result = 0;

```

1610. Update handles, insert data. [LDF 2012.10.05.]**Log**

[LDF 2012.10.05.] Added this section. The UPDATE command and the `admin_data` table are needed because of problems with subqueries when using the MySQL C-API: It didn't work to use a subquery when multiple rows were inserted using `INSERT`. It might be possible to get rid of the `admin_data` table and just use the `handles` table instead, but there were problems locking the latter using an alias. I think it may be simpler to just use the `admin_data` table.

```

⟨ generate_pids definition 1581 ⟩ +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "sql_strm_2.str()=" << sql_strm_2.str() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
status = submit_mysql_query(sql_strm_2.str(), result, mysql_ptr, 0, 0, &affected_rows);
if (status != 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'generate_pids': " << endl <<
        "'submit_mysql_query' failed, returning " << status << ":" << endl <<
        mysql_error(mysql_ptr) << endl << "Failed to update rows for PIDs in";
if (standalone_hs) cerr << "'handlesystem_standalone.handles'";
else cerr << "'handlesystem.handles'";
cerr << "database_table." << endl << "Exiting function unsuccessfully with re\"
    turn_value_1." << endl;
unlock_cerr_mutex();
if (result) mysql_free_result(result);
result = 0;
failed = true;
ret_val = 1;
goto UNLOCK_TABLES;
} /* if (status != 0) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'generate_pids': Updated rows for PIDs in";
        if (standalone_hs) cerr << "'handlesystem_standalone.handles'";
        else cerr << "'handlesystem.handles'";
        cerr << "table successfully." << endl << "'affected_rows'=" << affected_rows << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
sql_strm_2.str("");
mysql_free_result(result);
result = 0;

```

1611. Unlock tables. [LDF 2012.09.28.]

(generate_pids definition 1581) +≡

UNLOCK_TABLES:

```
if (result) {
    mysql_free_result(result);
    result = 0;
}
sqlStrm.str("");
sqlStrm << "unlock_tables";
status = submit_mysql_query(sqlStrm.str(), result, mysql_ptr);
if (status != 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'generate_pids':"
        << endl <<
        "'submit_mysql_query' failed, returning"
        << status << ":" << endl <<
        mysql_error(mysql_ptr) << endl <<
        "Failed to unlock_tables." << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    return 1;
} /* if (status != 0) */
#endif /* DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'generate_pids': Unlocked_tables successfully."
        << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
mysql_free_result(result);
result = 0;
sqlStrm.str("");
```

1612.**Log**

[LDF 2013.02.22.] Added this section. Now exiting after trying to unlock the database tables before continuing. Formerly, this function exited at this point. Now, it may write the PID or PIDs to a FIFO.

```
< generate_pids definition 1581 > +≡
if (failed) {
    lock_cerr_mutex();
    cerr << "In 'generate_pids': failed' == 'true'." << endl <<
        "Exiting function unsuccessfully with return value" << ret_val << "." << endl;
    unlock_cerr_mutex();
    return ret_val;
}
#if DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'generate_pids': failed' == 'false'. Continuing." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1613.

```
<generate_pids definition 1581> +≡
  if (!fifo_pathname.empty()) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'generate_pids': 'fifo_pathname' is non-empty: " <<
      endl << "'fifo_pathname'" << fifo_pathname << ". " << endl <<
      "Will try to open and write to it." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  errno = 0;
  status = access(fifo_pathname.c_str(), F_OK);
  if (status == -1) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'generate_pids': 'access' failed, returning -1: " << endl <<
      strerror(errno) << endl << "Failed to check for existence of FIFO " << fifo_pathname <<
      ". " << endl << "Not writing to FIFO " << fifo_pathname << ". " << endl;
    unlock_cerr_mutex();
    failed = true;
    ret_val = 1;
    goto GENERATE_PIDS_POST_FIFO;
  } /* if (status != 0) */
#endif DEBUG_COMPILE
  else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'generate_pids': 'access' succeeded, returning 0. " << endl <<
      "Checked for existence of FIFO " << fifo_pathname << " successfully. " << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1614.

```
<generate_pids definition 1581> +≡
errno = 0;
fd = open(fifo_pathname.c_str(), O_WRONLY | O_NONBLOCK);
if (status ≡ -1) {
    lock_cerr_mutex();
    cerr ≡ "ERROR! In 'generate_pids': 'open' failed, returning -1:" ≡ endl ≡
        strerror(errno) ≡ endl ≡ "Failed to open FIFO " ≡ fifo_pathname ≡
        "' for writing." ≡ endl;
    unlock_cerr_mutex();
    failed = true;
    ret_val = 1;
    goto GENERATE_PIDS_POST_FIFO;
} /* if (status ≡ -1) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≡ "In 'generate_pids': 'open' succeeded." ≡ endl ≡ "Opened FIFO " ≡
            fifo_pathname ≡ "' for writing successfully." ≡ endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1615.

```

⟨ generate_pids definition 1581 ⟩ +≡
  if (pid_vector_ptr ∧ pid_vector_ptr->size() > 0) {
    for (vector<string>::const_iterator iter = pid_vector_ptr->begin(); iter ≠ pid_vector_ptr->end();
         ++iter) {
      errno = 0;
      status = write(fd, iter->c_str(), iter->size());
      if (status ≡ -1) {
        lock_cerr_mutex();
        cerr ≪ "WARNING! In 'generate_pids': 'write' failed, returning -1" ≪ endl ≪
              strerror(errno) ≪ endl ≪ "Failed to write to FIFO" ≪ fifo_pathname ≪
              '.' ≪ endl ≪ "It may just not be open for writing." ≪ endl ≪
              "This occurs, for example, when 'input' is called directly," ≪ endl ≪
              "i.e., not from a program that opens the FIFO for reading." ≪ endl;
        unlock_cerr_mutex();
        close(fd);
        goto GENERATE_PIDS_POST_FIFO;
      } /* if (status ≡ -1) */
      else if (status ≡ 0) {
        lock_cerr_mutex();
        cerr ≪ "ERROR! In 'generate_pids': 'write' succeeded, but returned 0" ≪ endl ≪
              "Wrote 0 bytes to FIFO" ≪ fifo_pathname ≪ '.' ≪ endl;
        unlock_cerr_mutex();
        failed = true;
        close(fd);
        ret_val = 1;
        goto GENERATE_PIDS_POST_FIFO;
      } /* if (status ≡ 0) */
    #if DEBUG_COMPILE
    else
      if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ "In 'generate_pids': 'write' succeeded, returning " ≪ status ≪ "."
              ≪ endl ≪ "Wrote to FIFO" ≪ fifo_pathname ≪ ", successfully." ≪ endl;
        unlock_cerr_mutex();
      } /* else if (DEBUG) */
    #endif /* DEBUG_COMPILE */
    } /* for */
  } /* if (pid_vector_ptr ∧ pid_vector_ptr->size() > 0) */

```

1616.

```

⟨ generate_pids definition 1581 ⟩ +≡
else
  if (pid_str.size() > 0) {
    errno = 0;
    status = write(fd, pid_str.c_str(), pid_str.size());
    if (status == -1) {
      lock_cerr_mutex();
      cerr << "WARNING! In 'generate_pids': 'write' failed, returning -1" << endl <<
        strerror(errno) << endl << "Failed to write to FIFO" << fifo_pathname <<
        "." << endl << "It may just not be open for writing." << endl <<
        "This occurs, for example, when 'input' is called directly," << endl <<
        "i.e., not from a program that opens the FIFO for reading." << endl;
      unlock_cerr_mutex();
      close(fd);
      goto GENERATE_PIDS_POST_FIFO;
    } /* if (status == -1) */
  else if (status == 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'generate_pids': 'write' succeeded, but returned 0:" << endl <<
      "Wrote 0 bytes to FIFO" << fifo_pathname << "." << endl;
    unlock_cerr_mutex();
    close(fd);
    failed = true;
    ret_val = 1;
    goto GENERATE_PIDS_POST_FIFO;
  } /* if (status == 0) */
#endif DEBUG_COMPILE
else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'generate_pids': 'write' succeeded, returning " << status << "." <<
      endl << "Wrote to FIFO" << fifo_pathname << " successfully." << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* else if (pid_str.size() > 0) */
else {
  lock_cerr_mutex();
  cerr << "ERROR! In 'generate_pids': 'pid_ctr' is empty and 'pid_vector' is" <<
    "NULL or empty." << endl << "No PIDs to write to FIFO" << fifo_pathname << "." <<
    endl;
  unlock_cerr_mutex();
  close(fd);
  failed = true;
  ret_val = 1;
  goto GENERATE_PIDS_POST_FIFO;
} /* else */
close(fd); } /* if (!fifo_pathname.empty()) */
#endif DEBUG_COMPILE
else
  if (DEBUG) {

```

```

lock_cerr_mutex();
cerr << "In 'generate_pids': 'fifo_pathname' is empty." << endl <<
"Not writing to FIFO." << endl;
unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
GENERATE_PIDS_POST_FIFO: ;

```

1617.

```

⟨ generate_pids definition 1581 ⟩ +≡
if (failed) {
    lock_cerr_mutex();
    cerr << "In 'generate_pids': 'failed' == 'true'." << endl <<
        "Exiting function unsuccessfully with return value" << ret_val << "." << endl;
    unlock_cerr_mutex();
    return ret_val;
}
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'generate_pids': 'failed' == 'false'. Continuing." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1618. Exit successfully. [LDF 2012.09.28.]

```

⟨ generate_pids definition 1581 ⟩ +≡
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "Exiting 'generate_pids' successfully with return value 0." << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
return 0; } /* generate_pids */

```

1619. Check prefix. [LDF 2012.10.08.]

Log

[LDF 2012.10.08.] Added this function.

Return values:

- 0: Success
- 1: MySQL error, i.e., query or fetch row failed
- 2: Prefix not found, i.e., no database entry for prefix
- 3: Prefix disabled

```

⟨ check_prefix declaration 1619 ⟩ ≡
int check_prefix(MYSQL *mysql_ptr, string prefix_str, unsigned int &prefix_id);

```

This code is used in sections 1624 and 1625.

1620.

```
⟨ check_prefix definition 1620 ⟩ ≡
int check_prefix(MYSQL *mysql_ptr, string prefix_str, unsigned int &prefix_id){ bool DEBUG = false;
    /* true */
    set_debug_level(DEBUG, 0, 0);
    int status = 0;
    stringstream sql strm;
    MYSQL_RES *result = 0;
    MYSQL_ROW curr_row;
    unsigned int row_ctr = 0;
    unsigned int field_ctr = 0;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Entering 'check_prefix'." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

See also sections 1621, 1622, and 1623.

This code is used in section 1624.

1621.**Log**

[LDF 2012.11.22.] BUG FIX: Now specifying the database `gwirdsif` where `Prefixes` is located. Previously, the query failed because another database was selected as the default when this function was called.

```

⟨ check_prefix definition 1620 ⟩ +≡
sql_strm << "select_prefix_id,enabled from gwirdsif.Prefixes where prefix='"
prefix_str << "'";
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'check_prefix': sql_strm.str() == " << sql_strm.str() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
status = submit_mysql_query(sql_strm.str(), result, mysql_ptr, &row_ctr, &field_ctr);
if (status != 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'check_prefix': submit_mysql_query failed, returning "
        ". " << endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    return 1;
} /* if (status != 0) */
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "In 'check_prefix': submit_mysql_query succeeded, returning 0."
                " 'row_ctr' == " << row_ctr << endl << "'field_ctr' == " << field_ctr << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1622.

```

⟨ check_prefix definition 1620 ⟩ +≡
if (row_ctr == 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'check_prefix': 'row_ctr' == 0. "
        "Prefix " << prefix_str <<
        "' not found." << endl << "Exiting function unsuccessfully with return value 2." <<
        endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    result = 0;
    return 2;
} /* if (row_ctr == 0) */

```

1623.

```

⟨ check_prefix definition 1620 ⟩ +≡
  if ((curr_row = mysql_fetch_row(result)) == 0) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'check_prefix': mysql_fetch_row failed:" << endl <<
      mysql_error(mysql_ptr) << endl << "Exiting 'check_prefix' unsuccessfully\n"
      iwith_return_value_1." << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    result = 0;
    return 1;
  } /* if (curr_row = mysql_fetch_row(result) == 0) */
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "In 'check_prefix': mysql_fetch_row succeeded." << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  prefix_id = atoi(curr_row[0]);
  unsigned int enabled = atoi(curr_row[1]);
  if (enabled == 0) {
    lock_cerr_mutex();
    cerr << "WARNING! In 'check_prefix': Prefix " << prefix_str <<
      "'is'" << "disabled.enabled'" <= " " << enabled << endl <<
      "Exiting 'check_prefix' unsuccessfully with return value 3." << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    result = 0;
    return 3;
  } /* if (enabled == 0) */
  mysql_free_result(result);
  result = 0;
#endif DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'check_prefix': prefix_id'" <= " " << prefix_id << endl <<
      "'prefix_str'" <= " " << prefix_str << endl <<
      "'enabled'" <= " " << enabled << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "Exiting 'check_prefix' successfully with return value 0." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  return 0; } /* End of check_prefix definition */

```

1624. Putting utility functions together. [LDF 2008.12.05.]

```
<Include files 3>
using namespace std;
using namespace gwrdifpk;
<generate_pids declaration 1580>
<check_prefix declaration 1619>
<generate_pids definition 1581>
<check_prefix definition 1620>
```

1625. This is what's written to the header file `pidfncs.h`. [LDF 2008.12.05.]

```
<pidfncs.h 1625>≡
#ifndef PIDFNCS_H
#define PIDFNCS_H 1
using namespace std;
using namespace gwrdifpk;
<generate_pids declaration 1580>
<check_prefix declaration 1619>
#endif
```

1626. Functions for use in the parsers. [LDF 2013.03.06.]

1627. Include files.

```
<Include files 3> +≡
#include <stdlib.h>
#include <stdio.h>
#include <stdarg.h>
#include <limits.h>
#include <ctype.h>
#include <algorithm>
#include <bitset>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <iostream>
#include <iterator>
#include <time.h>
#include <math.h>
#include <sstream>
#include <map>
#include <vector>
#include <deque>
#include <stack>
#include <set>
#include <pthread.h>
#if HAVE_CONFIG_H
#include "config.h"
#endif
#include <gcrypt.h> /* for gcry-control */
#include <gnutls/gnutls.h>
#include <gnutls/x509.h>
#include <mysql.h>
#include <expat.h>
#undef NAME_LEN
#undef LOCAL_HOST
#include "glblcnst.h++"
#include "glblvrbl.h++"
#include "utilfncs.h++"
#include "grouptp.h++"
#include "hdlvltp.h++"
#include "irdsavtp.h++"
#include "pidfncs.h++"
#include "tanfncs.h++"
#include "parser.h++"
#include "prsrlnt.h++"
#include "scanner.h++"
#include "scnrlnt.h++"
#include "rspnstp.h++"
#include "irdsobtp.h++"
#include "hdltype.h++"
#include "dcmtddtp.h++"
#include "x509cert.h++"
#include "dstngnmt.h++"
#include "scprpmtp.h++"
```

```
#include "usrinftp.h++"
```

1628. *client_sending_file_rule_func.* [LDF 2013.03.06.]

Log

[LDF 2013.03.06.] Added this function.

⟨ Parser function declarations 1628 ⟩ ≡

```
int client_sending_file_rule_func(Scan_Parse_Parameter_Type *param, string filename, int
reference);
```

See also sections 1636 and 1644.

This code is used in sections 1657 and 1658.

1629.

⟨ *client_sending_file_rule_func* definition 1629 ⟩ ≡

```
int client_sending_file_rule_func(Scan_Parse_Parameter_Type *param, string filename, int
reference){ int status;
bool DEBUG = false; /* true */
set_debug_level(DEBUG, 0, 0);
string thread_str;
stringstream temp_strm;
if (param->thread_ctr > 0) {
    temp_strm << "[Thread" << param->thread_ctr << "]";
    thread_str = temp_strm.str();
    temp_strm.str("");
}
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "Entering 'client_sending_file_rule_func'." << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

See also sections 1630, 1631, 1632, 1633, 1634, and 1635.

This code is used in section 1657.

1630.

⟨ *client_sending_file_rule_func* definition 1629 ⟩ +≡

```
#if DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'client_sending_file_rule_func':" << endl << "'filename' == " <<
        filename << endl << "'reference' == " << reference << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
Response_Type response;
string new_local_filename;
string temp_filename;
```

1631.

```
{ client_sending_file_rule_func definition 1629 } +≡
    pthread_mutex_lock(&param->response_map_mutex);
    map<unsigned int, Response_Type>::iterator iter = param->response_map.find(reference);
    if (iter == param->response_map.end()) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR!" << endl << "In 'client_sending_file_rule_func':"
            << endl << "Failed to find reference" << reference << " in 'param->response_map'."
            << endl << "Can't 'put' file" << filename << '.' << endl <<
            "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        ++param->errors_occurred;
        pthread_mutex_unlock(&param->response_map_mutex);
        return 1;
    } /* if (iter == param->response_map.end()) */
#endif /* DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'client_sending_file_rule_func': " << endl << "Found reference"
            << reference << " in 'param->response_map': " << endl;
        iter->second.show();
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1632.

```

⟨ client_sending_file_rule_func definition 1629 ⟩ +≡
    pthread_mutex_unlock(&param→response_map_mutex);
    response = iter→second;
    /* Not passing response.local_filename to Scan_Parse_Parameter_Type::receive_file! A temporary
       file should be created. Then we use response.local_filename in the “put” command. [LDF 2012.11.22.]
    */
    status = param→receive_file("", "", false, 0, &temp_filename);
    if (status ≠ 0) {
        lock_cerr_mutex();
        cerr ≪ thread_str ≪ "ERROR! In ‘client_sending_file_rule_func’:"
        ≪ endl ≪
            "‘Scan_Parse_Parameter_Type::receive_file’ failed, returning " ≪ status ≪ "."
        ≪ endl ≪ "Exiting function unsuccessfully with return value 1."
        ≪ endl;
        unlock_cerr_mutex();
        return 1;
    } /* if (status ≠ 0) */
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr ≪ thread_str ≪ "In ‘client_sending_file_rule_func’:"
            ≪ endl ≪
                "‘Scan_Parse_Parameter_Type::receive_file’ succeeded, returning 0."
            ≪ endl ≪
                "'temp_filename' == " ≪ temp_filename ≪ endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1633.

```

⟨ client_sending_file_rule_func definition 1629 ⟩ +≡
    response.temporary_filename = temp_filename;
#ifndef 0 /* 1 */
    response.show("response:");
#endif

```

1634.

```

⟨ client_sending_file_rule_func definition 1629 ⟩ +≡
  if (response.type == Response_Type::RECEIVE_PUT_FILE_TYPE) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'client_sending_file_rule_func':"
    << endl <<
      " 'response.type' == 'Response_Type::RECEIVE_PUT_FILE_TYPE' "
    << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  status = param->put(response);
  if (status != 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR!" << endl <<
      "In 'client_sending_file_rule_func':"
    << endl <<
      " 'Scan_Parse_Parameter_Type::put' failed, returning "
    << status << ". "
    << endl <<
      "Exiting function unsuccessfully with return value 1."
    << endl;
    unlock_cerr_mutex();
    ++param->errors_occurred;
    return 1;
  } /* if (status != 0) */
#endif DEBUG_COMPILE
  else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'client_sending_file_rule_func':"
    << endl <<
      " 'Scan_Parse_Parameter_Type::put' succeeded, returning 0."
    << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  param->temp_file_vector.push_back(response.temporary_filename);
  pthread_mutex_lock(&param->response_map_mutex);
  param->response_map.erase(iter);
  pthread_mutex_unlock(&param->response_map_mutex);
} /* if (response.type == Response_Type::RECEIVE_PUT_FILE_TYPE) */
else if (response.type == Response_Type::RECEIVE_METADATA_FILE_TYPE) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'client_sending_file_rule_func':"
    << endl <<
      " 'response.type' == 'Response_Type::RECEIVE_METADATA_FILE_TYPE' "
    << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  status = param->add_metadata(response);
  if (status != 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR!" << endl <<
      "In 'client_sending_file_rule_func':"
    << endl <<
      " 'Scan_Parse_Parameter_Type::add_metadata' failed, returning "
    << status << ". "
    << endl <<
      "Exiting function unsuccessfully with return value 1."
    << endl;
    unlock_cerr_mutex();
    ++param->errors_occurred;
  }
}

```

```

        return 1;
    } /* if (status != 0) */
#endif DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'client_sending_file_rule_func':"
        << endl <<
        "'Scan_Parse_Parameter_Type::add_metadata' succeeded, returning 0."
        << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
param->temp_file_vector.push_back(response.temporary_filename);
pthread_mutex_lock(&param->response_map_mutex);
param->response_map.erase(iter);
pthread_mutex_unlock(&param->response_map_mutex);
} /* else if (response.type == Response_Type::RECEIVE_METADATA_FILE_TYPE) */
else {
    lock_cerr_mutex();
    cerr << thread_str << "WARNING! In 'client_sending_file_rule_func':"
        << endl <<
        "'response.type'" == " "
        << Response_Type::typename_map[response.type] <<
        ", "
        << endl << "This case hasn't been accounted for."
        << endl <<
        "Exiting function unsuccessfully with return value 2."
        << endl;
    unlock_cerr_mutex();
    return 2;
} /* else */

```

1635.

```

⟨ client_sending_file_rule_func definition 1629 ⟩ +≡
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "Exiting 'client_sending_file_rule_func' successfully with"
        << "return_value 0."
        << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
return 0; } /* End of client_sending_file_rule_func definition */

```

1636. distinguished_name_rule_func. [LDF 2013.05.22.]**Log**

[LDF 2013.05.22.] Added this function.

```

⟨ Parser function declarations 1628 ⟩ +≡
int distinguished_name_rule_func(Scan_Parse_Parameter_Type *param, const char
    *distinguished_name_str);

```

1637.

```

⟨ distinguished_name_rule_func definition 1637 ⟩ ≡
int distinguished_name_rule_func(Scan_Parse_Parameter_Type *param, const char
    *distinguished_name_str){ int status;
bool DEBUG = false; /* true */
set_debug_level(DEBUG, 0, 0);
Response_Type response;
response.type = Response_Type::COMMAND_ONLY_TYPE;
string thread_str;
stringstream temp_strm;
User_Info_Type user_info;
if (param->thread_ctr > 0) {
    temp_strm << "Thread[" << param->thread_ctr << "] ";
    thread_str = temp_strm.str();
    temp_strm.str("");
}
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "Entering 'distinguished_name_rule_func'." << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

See also sections 1638, 1639, 1640, 1641, 1642, and 1643.

This code is used in section 1657.

1638.

```

⟨ distinguished_name_rule_func definition 1637 ⟩ +≡

```

1639. $\text{param-user_id} \leq 0$. User not already set. This can only occur when the connection from the client uses one of the methods *without* authentication, i.e., a local connection via a socket or a TLS connection using GnuTLS' “anonymous” authentication. These possibilities are for **testing only!** [LDF 2013.05.14.]

```

⟨ distinguished_name_rule_func definition 1637 ⟩ +≡
  if (param->user_id ≤ 0) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << param->thread_ctr << "] In 'distinguished_name_rule_func':"
    endl << "param->user_id" <=> param->user_id << "(≤ 0)" << endl <<
    "Calling Scan_Parse_Parameter_Type::get_user'." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  status = param->get_user(0, distinguished_name_str);
  if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << "[Thread" << param->thread_ctr << "] ERROR! In 'distinguished_name_rule_func':"
    endl << "Scan_Parse_Parameter_Type::get_user' failed, returning" << status <<
    "." << endl << "Can't set user. Exiting function with return value 2"
    "(authentication_error)." << endl;
    unlock_cerr_mutex();
    return 2;
  } /* if (status ≠ 0) */
#endif DEBUG_COMPILE
  else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << param->thread_ctr << "] In 'distinguished_name_rule_func':"
    endl << "Scan_Parse_Parameter_Type::get_user' succeeded."
    << endl << "param->user_id" <=> param->user_id << endl <<
    "param->username" <=> param->username << endl <<
    "param->irods_password_encrypted" <=> param->irods_password_encrypted << endl <<
    "param->irods_homedir" <=> param->irods_homedir << endl <<
    "param->irods_zone" <=> param->irods_zone << endl <<
    "param->irods_default_resource" <=> param->irods_default_resource << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  status = param->set_user_info(user_info);
  if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << "[Thread" << param->thread_ctr << "] ERROR! In 'distinguished_name_rule_func':"
    endl << "Scan_Parse_Parameter_Type::set_user_info' failed, returning" << status <<
    "." << endl << "Failed to check Distinguished Name" << distinguished_name_str <<
    "," << endl << "Exiting function unsuccessfully with return value 2 (authentication_error)." << endl;
    unlock_cerr_mutex();
    ++param->warnings_occurred;
    return 2;
  } /* if (status ≠ 0) */
else {
#endif DEBUG_COMPILE

```

```

if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << param->thread_ctr << "] In 'distinguished_name_rule_func':"
    endl << 'Scan_Parse_Parameter_Type::set_user_info' succeeded:" << endl;
    user_info.show("user_info:");

    bitset<sizeof(unsigned int)*8> temp_bitset(param->privileges);
    bitset<sizeof(unsigned int)*8> show_distinguished_names_bitset(Scan_Parse_Parameter_Type::SHOW_DIS-
    cerr << "temp_bitset" ==_uuuuuuuuuuuuuuuuuuuu" << temp_bitset << endl <<
        "show_distinguished_names_bitset" ==_u" << show_distinguished_names_bitset <<
        endl << "(param->privileges&" << "Scan_Parse_Parameter\-
        r_Type::SHOW_DISTINGUISHED_NAMES_PRIVILEGE)" ==_u" << (param->privileges &
        Scan_Parse_Parameter_Type::SHOW_DISTINGUISHED_NAMES_PRIVILEGE) << endl;
    unlock_cerr_mutex();
}
/* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* else */
} /* if (param->user_id ≤ 0) */

```

1640. *param->user_id > 0*. User already set. In this case, we call **Scan_Parse_Parameter_Type::get_user** with a **User_Info_Type** argument in order to get the user info corresponding to the Distinguished Name in *distinguished_name_str*. It doesn't matter how the user was set; it could have been set from a certificate or this rule may have been matched previously, if a connection without authentication is being used. !! PLEASE NOTE: Connections without authentication are for **testing only!** [LDF 2013.05.14.]

```

⟨ distinguished_name_rule_func definition 1637 ⟩ +≡
else /* param->user_id > 0 */
{

```

1641.

Log

[LDF 2013.05.19.] Added this section.

```

⟨ distinguished_name_rule_func definition 1637 ⟩ +≡
Distinguished_Name_Type d;
d.set(distinguished_name_str);
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    param->distinguished_name.show("param->distinguished_name:");
    d.show("d:");
    cerr << "(d==param->distinguished_name)" << (d == param->distinguished_name) << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread]" << param->thread_ctr << "]In" 'distinguished_name_rule_func':"
        "param->user_id" << param->user_id << "(>0)" << endl << "Useralreadyset."
        endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
if (param->distinguished_name ≡ distinguished_name_str) {
    /* User requesting information about his or her own Distinguished Name. [LDF 2013.05.19.] */
    status = param->set_user_info(user_info);
}
else {
    status = param->get_user(0, distinguished_name_str, "", &user_info);
}
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << "[Thread]" << param->thread_ctr << "]WARNING!In" 'distinguished_name_rule_func':"
        endl;
    if (param->distinguished_name ≡ distinguished_name_str)
        cerr << "Scan_Parse_Parameter_Type::set_user_info" ↴failed, ↴returning";
    else cerr << "Scan_Parse_Parameter_Type::get_user" ↴failed, ↴returning";
    cerr << status << "." << endl << "Failed" ↴to" ↴check" ↴DistinguishedName" <<
        distinguished_name_str << "." << endl << "Exiting" ↴function" ↴unsuccessfully" ↴with" ↴re\
        turn" ↴value" ↴1." << endl;
    unlock_cerr_mutex();
    temp_strm.str("");
    temp_strm << "DISTINGUISHED_NAME_RESPONSE" ↴1" << distinguished_name_str << "\n" << "-1";
    if (param->distinguished_name ≡ distinguished_name_str)
        temp_strm << "\"(Failed" ↴to" ↴set" ↴user" ↴info)\\"";
    else temp_strm << "\"(Username" ↴not" ↴found)\\"";
    response.command = temp_strm.str();
    temp_strm.str("");
    param->response_deque.push_back(response);
}

```

```

++param->warnings_occurred;
return 1;
} /* if (status != 0) */
else {
#endif DEBUG_COMPILE
if (DEBUG) {
lock_cerr_mutex();
cerr << "[Thread" << param->thread_ctr << "] In 'distinguished_name_rule_func':"
endl << "'Scan_Parser_Parameter_Type::get_user' or "
"'Scan_Parser_Parameter_Type::set_user_info' succeeded:" << endl;
user_info.show("user_info:");

bitset<sizeof(unsigned int)*8> temp_bitset(param->privileges);
bitset<sizeof(unsigned int)*8> show_distinguished_names_bitset(Scan_Parser_Parameter_Type::SHOW_DISTIN
cerr << "temp_bitset==ooooooooooooooooooooo" << temp_bitset << endl <<
"show_distinguished_names_bitset==" << show_distinguished_names_bitset <<
endl << "(param->privileges&" << "Scan_Parser_Parameter\
r_Type::SHOW_DISTINGUISHED_NAMES_PRIVILEGE)" <= " << (param->privileges &
Scan_Parser_Parameter_Type::SHOW_DISTINGUISHED_NAMES_PRIVILEGE) << endl;
unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
if (param->user_id == user_info.user_id || (param->privileges &
Scan_Parser_Parameter_Type::SHOW_DISTINGUISHED_NAMES_PRIVILEGE) > 0_U)
{
temp_strm.str("");
temp_strm << "DISTINGUISHED_NAME_RESPONSE\0\" << distinguished_name_str << "\""
<< user_info.user_id << "\" << user_info.username << "\"";
response.command = temp_strm.str();
temp_strm.str("");
param->response_deque.push_back(response);
} /* if */
else {
lock_cerr_mutex();
cerr << "[Thread" << param->thread_ctr << "] WARNING! In 'distinguished_name_rule\
_func':"
endl << "User" << param->username << "(ID" << param->user_id << ")" <<
"not permitted to check the Distinguished Names of other users."
<< endl <<
"Exiting function unsuccessfully with return value 1." << endl;
unlock_cerr_mutex();
temp_strm.str("");
temp_strm << "DISTINGUISHED_NAME_RESPONSE\1\" << distinguished_name_str << "\""
<< "-1\"(Action not permitted)\\"";
response.command = temp_strm.str();
temp_strm.str("");
param->response_deque.push_back(response);
++param->warnings_occurred;
return 1;
} /* else */
} /* else */
} /* else (param->user_id > 0) */

```

1642.

```
< distinguished_name_rule_func definition 1637 > +≡
  if (param->user_id_map.find(user_info.username) ≡ param->user_id_map.end())
    param->user_id_map[user_info.username] = user_info.user_id;
  if (param->user_info_map.find(user_info.user_id) ≡ param->user_info_map.end())
    param->user_info_map[user_info.user_id] = user_info;
```

1643.

Log

[LDF 2013.07.18.] Now putting **User_Info_Type** *user_info* onto *global_user_info_map*.

```
< distinguished_name_rule_func definition 1637 > +≡
  pthread_mutex_lock(&global_user_info_map_mutex);
  global_user_info_map[user_info.user_id] = user_info;
  pthread_mutex_unlock(&global_user_info_map_mutex);
#endif DEBUG_COMPILE
if (DEBUG) {
  lock_cerr_mutex();
  cerr ≪ thread_str ≪ "Exiting 'distinguished_name_rule_func' successfully with "
  "return value 0." ≪ endl;
  unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
return 0; } /* End of distinguished_name_rule_func definition */
```

1644. *get_user_info_func*. [LDF 2013.05.22.]

Log

[LDF 2013.05.22.] Added this function.

```
< Parser function declarations 1628 > +≡
int get_user_info_func(Scan_Parse_Parameter_Type *param, const char *curr_username = 0);
```

1645.

```

⟨ get_user_info_func definition 1645 ⟩ ≡
int get_user_info_func(Scan_Parse_Parameter_Type *param, const char *curr_username){ int
    status;
    bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    Response_Type response;
    response.type = Response_Type::COMMAND_ONLY_TYPE;
    string thread_str;
    stringstream temp_strm;
    User_Info_Type user_info;
    if (param->thread_ctr > 0) {
        temp_strm << "Thread" << param->thread_ctr << "]";
        thread_str = temp_strm.str();
        temp_strm.str("");
    }
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering 'get_user_info_func'." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

See also sections 1646, 1647, 1648, 1649, 1650, 1651, 1652, 1653, 1654, 1655, and 1656.

This code is used in section 1657.

1646.

```

⟨ get_user_info_func definition 1645 ⟩ +≡
  string username;
  if (curr_username ≡ 0) username = param->username;
  else username = curr_username;
  User_Info_Type *user_info_ptr = 0;
  map<string, int>::const_iterator iter = param->user_id_map.find(username);
  int curr_id = 0;
  if (iter ≠ param->user_id_map.end()) {
    curr_id = iter->second;
    map<int, User_Info_Type>::iterator iter_1 = param->user_info_map.find(curr_id);
    if (iter_1 ≠ param->user_info_map.end()) user_info_ptr = &iter_1->second;
  } /* if */
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "In 'get_user_info_func': " ≪ endl;
    if (user_info_ptr ≠ 0) {
      cerr ≪ "'user_info_ptr' != 0. Found 'User_Info': " ≪ endl ≪ user_info_ptr->user_id ≪
        ", " ≪ user_info_ptr->username ≪ endl;
    }
    else cerr ≪ "'user_info_ptr' == 0. " ≪ endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1647. Current user has requested info about a different user, but doesn't have the "show user info" privilege. Send a failure notice to the client. [LDF 2013.05.19.]

```

⟨ get_user_info_func definition 1645 ⟩ +≡
  if (param->username ≠ username ∧ (param->privileges &
    Scan_Parse_Parameter_Type::SHOW_USER_INFO_PRIVILEGE) ≡ 0_U) {
    temp_strm.str("");
    temp_strm ≪ "GET_USER_INFO_RESPONSE_1";
    response.command = temp_strm.str();
    lock_cerr_mutex();
    cerr ≪ "[Thread_]" ≪ param->thread_ctr ≪ "] " ≪ "ERROR! In 'get_user_info_func': " ≪ endl ≪
      "Current user has requested info about a different user, but doesn't have " ≪
      "the \"show user info\" privilege." ≪ endl ≪
      "Exiting function unsuccessfully with return value 1." ≪ endl;
    unlock_cerr_mutex();
    temp_strm.str("");
    temp_strm ≪ "GET_USER_INFO_RESPONSE_1";
    response.command = temp_strm.str();
    param->response_deque.push_back(response);
    ++param->errors_occurred;
    return 1;
  } /* if */

```

1648.

```

⟨ get_user_info_func definition 1645 ⟩ +≡
  if (user_info_ptr == 0) {
    if (username ≡ param→username) {
      status = param→set_user_info(user_info);
      if (status ≠ 0) {
        lock_cerr_mutex();
        cerr << "[Thread" << param→thread_ctr << "] " << "ERROR! In 'get_user_info_func':"
        endl << "'Scan_Parser_Parameter_Type::set_user_info' failed, returning" << status <<
          ". " << endl << "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        temp_strm.str("");
        temp_strm << "GET_USER_INFO_RESPONSE_1";
        response.command = temp_strm.str();
        param→response_deque.push_back(response);
        ++param→errors_occurred;
        return 1;
      } /* if (status ≠ 0) */
    #if DEBUG_COMPILE
    else
      if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param→thread_ctr << "] " << "In 'get_user_info_func':"
        endl << "'Scan_Parser_Parameter_Type::set_user_info' succeeded, returning 0." << endl;
        unlock_cerr_mutex();
      } /* if (DEBUG) */
    #endif /* DEBUG_COMPILE */
  } /* if (username ≡ param→username) */

```

1649.

```

⟨ get_user_info_func definition 1645 ⟩ +≡
  else if (param→privileges & Scan_Parse_Parameter_Type::SHOW_USER_INFO_PRIVILEGE) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << param→thread_ctr << "] " << "In 'get_user_info_func':" << endl <<
      "User" << param→user_id << " has the " << "show_user_info\" privilege." << endl <<
      "Calling 'Scan_Parse_Parameter_Type::get_user' to retrieve user info" << endl <<
      "for user " << username << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
/* !! TODO: LDF 2013.05.19. For admins, maybe make a way of querying for all users at once. */
status = param→get_user(0, 0, string(username), &user_info);
if (status ≠ 0) {
  lock_cerr_mutex();
  cerr << "[Thread" << param→thread_ctr << "] " << "ERROR! In 'get_user_info_func':" <<
    endl << "'Scan_Parse_Parameter_Type::get_user' failed, returning " << status << ". " <<
    endl << "Exiting function unsuccessfully with return value 1." << endl;
  unlock_cerr_mutex();
  temp_strm.str("");
  temp_strm << "GET_USER_INFO_RESPONSE_1";
  response.command = temp_strm.str();
  param→response_deque.push_back(response);
  ++param→errors_occurred;
  return 1;
} /* if (status ≠ 0) */
#ifndef DEBUG_COMPILE
else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << param→thread_ctr << "] " << "In 'get_user_info_func':" << endl <<
      "'Scan_Parse_Parameter_Type::get_user' succeeded, returning 0." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1650.

Log

[LDF 2013.07.18.] Added this section.

```

⟨ get_user_info_func definition 1645 ⟩ +≡
  pthread_mutex_lock(&global_user_info_map_mutex);
  global_user_info_map[user_info.user_id] = user_info;
  pthread_mutex_unlock(&global_user_info_map_mutex);

```

1651.

```

⟨ get_user_info_func definition 1645 ⟩ +≡
}      /* else if */
else {
    lock_cerr_mutex();
    cerr << "[Thread" << param->thread_ctr << "] " << "ERROR! In 'get_user_info_func':"
    endl << "This can't happen! User requesting info about other user, but "
    "doesn't have the \"show_user_info\" privilege." << endl <<
    "This case should have been caught above." << endl <<
    "Exiting function unsuccessfully with return value 1." << endl;
unlock_cerr_mutex();
temp_strm.str("");
temp_strm << "GET_USER_INFO_RESPONSE_1";
response.command = temp_strm.str();
param->response_deque.push_back(response);
++param->errors_occurred;
return 1;
}      /* else */
}      /* if (user_info_ptr == 0) */

```

1652.

```

⟨ get_user_info_func definition 1645 ⟩ +≡
if (user_info_ptr == 0) {
    user_info_ptr = &user_info;
    param->user_id_map[user_info.username] = user_info.user_id;
    param->user_info_map[user_info.user_id] = user_info;
}

```

1653.

```

⟨ get_user_info_func definition 1645 ⟩ +≡
temp_strm.str("");
temp_strm << "GET_USER_INFO_RESPONSE_0_USER_ID" << user_info_ptr->user_id << "U"
<< "USER_NAME\"" << user_info_ptr->username << "\" " << "CERTIFICATE_ID" <<
user_info_ptr->certificate.certificate_id << "U" << "SERIAL_NUMBER" <<
user_info_ptr->certificate.serialNumber << "U" << "COMMON_NAME\"" <<
user_info_ptr->certificate.commonName << "\" " << "LOCALITY_NAME\"" <<
user_info_ptr->certificate.localityName << "\" " << "STATE_OR_PROVINCE_NAME" <<
"\"" << user_info_ptr->certificate.stateOrProvinceName << "\" " << "COUNTRY_NAME\"" <<
user_info_ptr->certificate.countryName << "\" " << "ORGANIZATION\"" <<
user_info_ptr->certificate.organization << "\" " << "ORGANIZATIONAL_UNIT_NAME" <<
"\"" << user_info_ptr->certificate.organizationalUnitName << "\" " << "ISSUER_CERT_ID" <<
user_info_ptr->certificate.issuer_cert_id << "U" << "VALIDITY_NOT_BEFORE" <<
user_info_ptr->certificate.Validity_notBefore << "UL" << "VALIDITY_NOT_AFTER" <<
user_info_ptr->certificate.Validity_notAfter << "UL" << "IS_CA" << user_info_ptr->certificate.is_ca <<
"U" << "IS_PROXY" << user_info_ptr->certificate.is_proxy << "U" << "PRIVILEGES" <<
user_info_ptr->privileges << "U" << "IRODS_HOMEDIR\"" << user_info_ptr->irods_homedir << "\" "
"IRODS_ZONE\"" << user_info_ptr->irods_zone << "\" " << "IRODS_DEFAULT_RESOURCE\"" <<
user_info_ptr->irods_default_resource << "\" ";

```

1654. The current iRODS directory is only known for the current user, because it's not stored in the database. Doing this would require locking and unlocking the database whenever a user changes his or her current iRODS directory (i.e., with the CD command), and when querying the database for the user info, because the other user might be communicating with the server at the same time. [LDF 2013.05.22.]

```
< get_user_info_func definition 1645 > +≡
if (user_info_ptr->user_id == param->user_id)
    temp_strm << "IRODS_CURRENT_DIR\" " << param->irods_current_dir << "\" ";
temp_strm << "DEFAULT_HANDLE_PREFIX_ID\" " << user_info_ptr->default_handle_prefix_id <<
    "U\" " << "DEFAULT_HANDLE_PREFIX\" " << user_info_ptr->default_handle_prefix <<
    "\" " << "DEFAULT_INSTITUTE_ID\" " << user_info_ptr->default_institute_id << "U\" " <<
    "DEFAULT_INSTITUTE_NAME\" " << user_info_ptr->default_institute_name << "\" ";
response.command = temp_strm.str();
param->response_deque.push_back(response);
temp_strm.str("");
```

1655.

```
< get_user_info_func definition 1645 > +≡
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << param->thread_ctr << "] " << "In 'get_user_info_func': " << endl;
    bitset<sizeof(unsigned int)*8> b(user_info_ptr->privileges);
    cerr << "privileges_bitset:" <= << b << endl;
    user_info_ptr->show("user_info:");
    cerr << "'response.command'" <= << endl << response.command << endl;
}
#endif
param->show("*param:");
#endif
unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1656.

```
< get_user_info_func definition 1645 > +≡
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "Exiting 'get_user_info_func' successfully with return value 0. " <<
        endl;
    unlock_cerr_mutex();
}
#endif /* DEBUG_COMPILE */
return 0; } /* End of get_user_info_func definition */
```

1657. Putting parser functions together. [LDF 2013.03.06.]

```
⟨ Include files 3 ⟩  
using namespace std;  
using namespace gwrdifpk;  
⟨ Parser function declarations 1628 ⟩  
⟨ client_sending_file_rule_func definition 1629 ⟩  
⟨ distinguished_name_rule_func definition 1637 ⟩  
⟨ get_user_info_func definition 1645 ⟩
```

1658. This is what's written to the header file `prsrfncts.h`. [LDF 2008.12.05.]

```
⟨ prsrfncts.h 1658 ⟩ ≡  
#ifndef PRSRFNCS_H  
#define PRSRFNCS_H 1  
using namespace std;  
using namespace gwrdifpk;  
⟨ Parser function declarations 1628 ⟩  
#endif
```

1659. TAN functions (`tanfncts.web`).

Log

[LDF 2012.07.19.] Added this file.

1660. Include files.

```
<Include files 3> +≡
#ifndef _XOPEN_SOURCE
#define _XOPEN_SOURCE
#endif
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <sys/types.h>
#include <dirent.h>
#include <string.h>
#include <pwd.h>
#include <errno.h>
#include <pthread.h>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <iostream>
#include <string>
#include <sstream>
#include <deque>
#include <map>
#include <vector>
#if HAVE_CONFIG_H
#include <config.h>
#endif
#include <mysql.h>
#include "gblcnst.h++"
#include "gblvrb1.h++"
#include "excptntp.h++"
#include "utilfncs.h++"
```

1661. Generate TANs. [LDF 2012.07.16.]

Log

[LDF 2012.07.16.] Added this function.

```
<generate_tans declaration 1661> ≡
int generate_tans(MYSQL * mysql_ptr, unsigned int target = 1000, unsigned int
tans_per_iteration = 100, string thread_ctr_str = "");
```

This code is used in sections 1672 and 1673.

1662.

```

⟨ generate_tans definition 1662 ⟩ ≡
  int generate_tans(MYSQL *mysql_ptr, unsigned int target, unsigned int tans_per_iteration, string
                     thread_ctr_str){ bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "Entering 'generate_tans'." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

  int status;
  FILE *fp = 0;
  stringstream sql_strm;
  string optpsgen_path = "/home/lfinsto/irods_proj/irods/Finston/gwrdifpk/src/optpsgen";
  stringstream cmd_strm;
  cmd_strm << optpsgen_path << "--extra-random--length=29--min-block-size=4" <<
    "--max-block-size=4--alnum--iter" << tans_per_iteration <<
    "--delimiters='\\2>/dev/null";

  unsigned int local_buffer_size = 32 * tans_per_iteration + 1;
  char buffer[local_buffer_size];
  string insert_str;
  size_t insert_str_len;

#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'generate_tans': " << "local_buffer_size=" <<
      local_buffer_size << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

  string comma_str;
  MYSQL_RES *result = 0;
  MYSQL_ROW curr_row;
  unsigned int row_ctr = 0;
  unsigned int field_ctr = 0;
  long affected_rows = 0;
  unsigned int TAN_ctr = 0;

```

See also sections 1663, 1664, 1665, 1666, 1667, 1668, 1669, 1670, and 1671.

This code is used in section 1672.

1663. Lock TANs table. [LDF 2012.07.16.]

```
{ generate_tans definition 1662 } +≡
sql_strm << "lock_tables_TANs_write";
status = submit_mysql_query(sql_strm.str( ), result, mysql_ptr, 0, 0, 0, thread_ctr_str);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'generate_tans':"
        << endl <<
        "'submit_mysql_query' failed, returning" << status << ":" << endl <<
        mysql_error(mysql_ptr) << endl << "Failed to lock 'TANs' database table." << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    return 1;
} /* if (status ≠ 0) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_ctr_str << "In 'generate_tans': Locked 'TANs' table successfully."
            << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
mysql_free_result(result);
```

1664. Check how many unassigned TANs are already in the ‘TANs’ database table. [LDF 2012.07.13.]

```

⟨ generate_tans definition 1662 ⟩ +≡
  sql_strm.str("");
  sql_strm << "select_count(TAN) from TANs where user_id = 0";
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'generate_tans': " << sql_strm.str() == " " << sql_strm.str() <<
      endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  status = submit_mysql_query(sql_strm.str(), result, mysql_ptr, &row_ctr, &field_ctr, 0, thread_ctr_str);
  if (status != 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'generate_tans': "
      "submit_mysql_query failed, returning " << status << "." << endl <<
      "Failed to retrieve number of TANs from database table 'gwirdsif.TANs'." << endl <<
      "Exiting 'generate_tans' unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    if (result) {
      mysql_free_result(result);
      result = 0;
    }
    submit_mysql_query("unlock_tables", result, mysql_ptr, 0, 0, 0, thread_ctr_str);
    /* Try to unlock TANs table. [LDF 2012.07.18.] */
    if (result) mysql_free_result(result);
    return 1;
  } /* if (status != 0) */
#endif DEBUG_COMPILE
  else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'generate_tans': 'submit_mysql_query' succeeded. " << endl <<
      "'row_ctr' == " << row_ctr << "." << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1665.

```

⟨ generate_tans definition 1662 ⟩ +≡
if ((curr_row = mysql_fetch_row(result)) == 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'generate_tans': mysql_fetch_row failed:" <<
        endl << mysql_error(mysql_ptr) << endl << "Exiting 'generate_tans' unsuccessfully \
        with return_value 1." << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    submit_mysql_query("unlock_tables", result, mysql_ptr, 0, 0, 0, thread_ctr_str);
    /* Try to unlock TANs table. [LDF 2012.07.18.] */
    if (result) mysql_free_result(result);
    return 1;
} /* if (curr_row = mysql_fetch_row(result) == 0) */
#endif /* DEBUG_COMPILE */

else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'generate_tans': mysql_fetch_row succeeded." << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1666.

```

⟨ generate_tans definition 1662 ⟩ +≡
if (curr_row[0] == 0 || strlen(curr_row[0]) == 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'generate_tans': " << endl <<
        "'curr_row[0]' is NULL or empty. Can't set 'TAN_ctr'." << endl <<
        "Exiting 'generate_tans' unsuccessfully with return_value 1." << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    submit_mysql_query("unlock_tables", result, mysql_ptr, 0, 0, 0, thread_ctr_str);
    /* Try to unlock TANs table. [LDF 2012.07.18.] */
    if (result) mysql_free_result(result);
    return 1;
} /* if (curr_row[0] == 0 || strlen(curr_row[0]) == 0) */
TAN_ctr = atoi(curr_row[0]);
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'generate_tans': " << "'TAN_ctr' == " << TAN_ctr << endl;
    cerr << "cmd_strm.str() == " << cmd_strm.str() << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
mysql_free_result(result);
result = 0;

```

1667.

```
<generate_tans definition 1662> +≡
  if (TAN_ctr ≥ target) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'generate_tans': " << "'TAN_ctr'=>'target'" <<
      endl << "'TAN_ctr'== " << TAN_ctr << endl << "'target'== " << target << endl <<
      "Will unlock 'TANs_table_and_exit_function_successfully.'" << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  goto UNLOCK_TANS_TABLE;
} /* if (TAN_ctr ≥ target) */
```

1668.**Log**

[LDF 2012.07.19.] No longer setting expiration date for new TANs. It should be set when the TANs are assigned to a user. There is no reason why an unassigned TAN should expire. If an administrator suspects that the TANs table has been compromised, then all of the rows should be deleted.

```

<generate_tans definition 1662> +≡
insert_str = "insert ignore into TANs (TAN) values ";
insert_str_len = insert_str.length(); while (TAN_ctr < target) { memset(buffer, 0, local_buffer_size);
errno = 0;
fp = popen(cmd_strm.str().c_str(), "r");
comma_str = "";
if (fp == 0) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "ERROR! In 'generate_tans' : "
        << "popen' failed. Failed to execute 'optpsgen'. " << endl << "popen error: "
        strerror(errno) << endl << "Exiting function unsuccessfully with return value 1. " <<
        endl;
    unlock_cerr_mutex();
    if (result) {
        mysql_free_result(result);
        result = 0;
    }
    submit_mysql_query("unlock_tables", result, mysql_ptr, 0, 0, 0, thread_ctr_str);
    /* Try to unlock TANs table. [LDF 2012.07.18.] */
    if (result) mysql_free_result(result);
    return 1;
}
#endif DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_ctr_str << "In 'generate_tans' : "
        << "'popen' succeeded. " << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
sql_strm.str("");
sql_strm << insert_str;
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "feof(fp) == " << feof(fp) << endl;
    cerr << "ferror(fp) == " << ferror(fp) << endl;
    cerr << "strlen(buffer) == " << strlen(buffer) << endl;
    cerr << "buffer == " << buffer << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
while (!(feof(fp) || ferror(fp) || !fgets(buffer, local_buffer_size - 1, fp))) {
#endif DEBUG_COMPILE
if (DEBUG) {

```

```

lock_cerr_mutex();
cerr << thread_ctr_str << "In 'generate_tans': buffer'" << endl << buffer << endl;
unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
if (buffer[strlen(buffer) - 1] == '\n') buffer[strlen(buffer) - 1] = '\0';
sql_strm << comma_str << "(" << buffer << ")";
if (comma_str.empty()) comma_str = ",";
}
pclose(fp);
fp = 0;
#ifif DEBUG_COMPILE
if (DEBUG) {
lock_cerr_mutex();
cerr << thread_ctr_str << "In 'generate_tans': sql_strm.str()'" << sql_strm.str() <<
endl;
unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1669.

```

< generate_tans definition 1662 > +≡
if (sql_strm.str().length() == insert_str_len) {
lock_cerr_mutex();
cerr << thread_ctr_str << "ERROR! In 'generate_tans':"
<< endl <<
"sql_strm.str().length()' == 'insert_str_len'" << insert_str_len <<
"." << endl << "Failed to set values for MySQL insert command."
<< endl <<
"Exiting 'generate_tans' unsuccessfully with return value 1." << endl;
unlock_cerr_mutex();
if (result) {
mysql_free_result(result);
result = 0;
}
submit_mysql_query("unlock_tables", result, mysql_ptr, 0, 0, 0, thread_ctr_str);
/* Try to unlock TANs table. [LDF 2012.07.18.] */
if (result) mysql_free_result(result);
return 1;
} /* if (sql_strm.str().length() == test_len) */
#endif DEBUG_COMPILE
else
if (DEBUG) {
lock_cerr_mutex();
cerr << thread_ctr_str << "In 'generate_tans':"
<< endl <<
"sql_strm.str().length()' != 'insert_str_len'." << endl <<
"Set values for MySQL insert command successfully." << endl;
unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1670.

```
<generate_tans definition 1662> +≡
status = submit_mysql_query(sql_strm.str(), result, mysql_ptr, 0, 0, &affected_rows, thread_ctr_str);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ thread_ctr_str ≪ "ERROR! In 'generate_tans': "
        ≪ "'submit_mysql_query' failed, returning" ≪ status ≪ "."
        ≪ endl ≪ "Failed to write TANs to database table 'gwirdsif.TANs'."
        ≪ endl ≪ "Exiting 'generate_tans' unsuccessfully with return value 1."
        ≪ endl;
    unlock_cerr_mutex();
if (result) {
    mysql_free_result(result);
    result = 0;
}
submit_mysql_query("unlock_tables", result, mysql_ptr, 0, 0, 0, thread_ctr_str);
/* Try to unlock TANs table. [LDF 2012.07.18.] */
if (result) mysql_free_result(result);
return 1;
} /* if (status ≠ 0) */
#if DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ thread_ctr_str ≪ "In 'generate_tans': 'submit_mysql_query' succeeded."
        ≪ endl ≪ "'affected_rows' == "
        ≪ affected_rows ≪ "."
        ≪ endl;
    unlock_cerr_mutex();
}
/* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
TAN_ctr += affected_rows;
mysql_free_result(result);
result = 0; } /* while */
```

1671. Unlock TANs table. [LDF 2012.07.16.]

```
< generate_tans definition 1662 > +≡
UNLOCK_TANS_TABLE:
if (result) mysql_free_result(result);
status = submit_mysql_query("unlock_table", result, mysql_ptr, 0, 0, 0, thread_ctr_str);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ thread_ctr_str ≪ "ERROR! In 'generate_tans':"
    ≪ endl ≪
    " 'submit_mysql_query' failed, returning " ≪ status ≪ ":" ≪ endl ≪
    mysql_error(mysql_ptr) ≪ endl ≪ "Failed to unlock 'TANs' database table." ≪ endl ≪
    "Exiting function unsuccessfully with return value 1." ≪ endl;
    unlock_cerr_mutex();
    if (result) mysql_free_result(result);
    return 1;
} /* if (status ≠ 0) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ thread_ctr_str ≪ "In 'generate_tans': Unlocked 'TANs' table successfully."
        ≪ endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
mysql_free_result(result);
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ thread_ctr_str ≪ "Exiting 'generate_tans' successfully with return value 0."
        ≪ endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
return 0; } /* End of generate_tans definition */
```

1672. Putting utility functions together. [LDF 2008.12.05.]

```
< Include files 3 >
using namespace std;
using namespace gwrdifpk;
< generate_tans declaration 1661 >
< generate_tans definition 1662 >
```

1673. This is what's written to the header file `tanfncs.h`. [LDF 2008.12.05.]

```
< tanfncs.h 1673 > ≡
#ifndef TANFNCS_H
#define TANFNCS_H 1
using namespace std;
using namespace gwrdifpk;
< generate_tans declaration 1661 >
#endif
```

1674. Functions for culling and purging. [LDF 2013.04.09.]

1675. Include files.

```
<Include files 3> +≡
#ifndef _GNU_SOURCE
#define _GNU_SOURCE
#endif
#include <stdlib.h> /* Standard Library for C */
#include <stdio.h>
#include <errno.h>
#include <unistd.h>
#if 0
#include <sys/stat.h>
#include <sys/types.h>
#include <limits.h>
#endif
#include <string.h>
#include <algorithm> /* Standard Template Library (STL) for C++ */
#include <fstream>
#include <iomanip>
#include <iostream>
#include <iostream>
#include <map>
#include <string>
#include <time.h>
#include <math.h>
#include <sstream>
#include <vector>
#include <pthread.h> /* POSIX threads */
#include <mysql.h>
#if HAVE_CONFIG_H
#include <config.h>
#endif
#include "glblcnst.h++"
#include "glblvrbl.h++"
#include "excptntp.h++"
#include "utilfncts.h++"
```

1676. Purge log files function for server. [LDF 2013.04.09.]

```
<purge_server_logs declaration 1676> ≡
void *purge_server_logs(void *v);
```

This code is used in sections 1712 and 1713.

1677.

```
< purge_server_logs definition 1677 > ≡
void *purge_server_logs(void *v){ bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int sleep_value = *static_cast(int *)(v);
#if DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Entering 'purge_server_logs'." << endl << "'sleep_value'" <= sleep_value << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

See also sections 1678, 1679, 1680, 1681, 1682, 1683, 1684, 1685, 1686, 1687, 1688, 1689, 1690, 1691, 1692, and 1693.

This code is used in section 1712.

1678. The following shell command calls `ls` to output information on the temporary files with name `gwirdsif` or `gwirdsif_a` and an extension consisting of 6 “random” characters. The temporary files are sorted according to the time of last access (“`atime`”). The sort order is descending, i.e., oldest first.

The reason that `atime` is used is that the files containing the users' "scrambled" iRODS passwords must have their original last modification time ("`mtime`"); otherwise, the iRODS server considers them invalid. The timestamp is therefore stored in the `gwirdsif.Users` database table, along with the scrambled password itself. After the temporary file is created, `touch` is called to set the modification time. This is done in `Scan_Parse_Parameter_Type::get_user`.

If there are no matching files, `ls` fails. We must catch this error, otherwise it will appear that `popen` has failed. We therefore store the output in `a` and the exit status of `ls` in `r`. The output of `ls` is processed by `tr` and `cut` to extract the timestamps and filenames. The timestamps are extracted in the form of “seconds since the Epoch” for purposes of comparison with `time_t limit`, but also in the form of human-readable strings for debugging purposes (`date_str` and `time_str`).

If `DEBUG_COMPILE` is non-null and `DEBUG` \equiv `true`, some output from the shell commands are output to `stderr`. Otherwise, it's sent to `/dev/null`. [LDF 2013.04.09.]

Log

[LDF 2013.04.19.] Now using *purge_limit* to set *sec_val*. *purge_limit* is a global variable, which can be set using the option **--purge-limit**. The default is 14, i.e., days.

```

<purge_server_logs definition 1677> +≡
    unsigned int sec_val = (60 * 60 * 24 * purge_logs_limit);
    stringstream temp_strm;
    string err_out_str = ">/dev/null"; /* Send error output to /dev/null, unless DEBUG_COMPILE is
        non-null and DEBUG ≡ true. [LDF 2013.04.17.] */
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        err_out_str = ">&2";
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
temp_strm << "a='ls -l -t --reverse--time-style='+%s%Y-%m-%d%H:%M:%S' "
    "/tmp/gwirdsif\.\??????>/dev/null';" << "r=$?;" << "echo -e \"r==$r\""
err_out_str << ";" << "if test $r -eq 0; then echo -e \"r==0\nname=$a\""
err_out_str << ";" << "b='echo \"$a\" | tr -s \"\\n\" | cut -f6-d\"\\n\"';" <<
    "r=$?; echo \"After 'cut': $r\"" << err_out_str << ";" <<
    "if test $r -eq 0; then echo -e \"'cut' succeeded.\nb=$b\";" << err_out_str << ";" <<
    "echo \"$b\";" << "else echo -e \"'tr' or 'cut' failed.\";" << err_out_str << ";" << "fi;" <<
    "else echo \"r<>0.No temporary files found.\";" << err_out_str << ";" << "echo; fi";
string delete_temp_files_str = temp_strm.str();
temp_strm.str("");
unsigned int line_num;
unsigned int timestamp;
char filename[256];
int status;
int status_1;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'purge_server_logs': sec_val==" << sec_val << endl <<
            "'delete_temp_files_str'" << delete_temp_files_str << endl;
        unlock_cerr_mutex();
    }

```

```

    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1679. Main loop. Purge, delete, cull, rotate, etc. [LDF 2013.04.09.]

<purge_server_logs definition 1677> +≡

```
char buffer[2048]; for ( ; ; ) {
```

1680. Disable cancellation of this thread. [LDF 2013.04.18.]

Log

[LDF 2013.04.18.] Added this section.

```

<purge_server_logs definition 1677> +≡
status = pthread_setcancelstate(PTHREAD_CANCEL_DISABLE, 0);
if (status != 0) {
    lock_cerr_mutex();
    cerr << "WARNING! In 'purge_server_logs': pthread_setcancelstate' failed, "
        << "returning " << status << ":" << endl << strerror(status) << endl <<
        "Failed to disable thread cancellation. Will try to continue." << endl;
    unlock_cerr_mutex();
} /* if (status != 0) */
#endif DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'purge_server_logs': pthread_setcancelstate' succeeded, "
        << "returning 0." << endl << "Disabled thread cancellation successfully." << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1681.

```

⟨ purge_server_logs definition 1677 ⟩ +≡
    memset(buffer, 0, 2048);
    errno = 0;
    time_t now = time(0);
    if (now == (time_t) -1) {
        lock_cerr_mutex();
        cerr << "ERROR! In 'purge_server_logs': 'time' failed, returning -1:" <<
            endl << strerror(errno) << endl << "Exiting thread func\"
            tion unsuccessfully with exit status 0." << endl <<
            "PLEASE NOTE: Logs, temporary files, etc., will not be purged\"
            "until 'gwirdsif' is restarted." << endl;
        unlock_cerr_mutex();
        pthread_exit(0);
    } /* if */
    time_t limit = now - sec_val;
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'purge_server_logs':" << endl << "'now' == " << now << endl <<
            "'sec_val' == " << sec_val << endl << "'limit' == " << limit << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1682.

```

⟨ purge_server_logs definition 1677 ⟩ +≡
    errno = 0;
    FILE *fp = fopen(delete_temp_files_str.c_str(), "r");
    if (fp == 0) {
        lock_cerr_mutex();
        cerr << "ERROR! In 'purge_server_logs': 'fopen' failed, returning NULL:" <<
            endl << strerror(errno) << endl << "Exiting thread func\"
            tion unsuccessfully with exit status 0." << endl <<
            "PLEASE NOTE: Logs, temporary files, etc., will not be purged\"
            "until 'gwirdsif' is restarted." << endl;
        unlock_cerr_mutex();
        pthread_exit(0);
    }
#endif DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "In 'purge_server_logs': 'fopen' succeeded." << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    memset(buffer, 0, 2048);
    char date_str[16];
    char time_str[16];

```

1683.

```

<purge_server_logs definition 1677> +=

    do { memset(date_str, 0, 16);
        memset(time_str, 0, 16);
        status = fscanf(fp, "%u%u%u%u", &timestamp, date_str, time_str, filename); if (status > 0) {
#if DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "filename:" << filename << endl << "timestamp:" << timestamp << endl <<
            "date_str:" << date_str << endl << "time_str:" << time_str << endl;
        unlock_cerr_mutex();
    }
#endif /* DEBUG_COMPILE */
    if (timestamp < limit) {
#if DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "timestamp<limit." << "Deleting" << filename << endl << "timestamp=="
                << timestamp << endl << "limit==" << limit << endl << "limit->timestamp==" <<
                (limit - timestamp) << endl;
            unlock_cerr_mutex();
        }
#endif /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        errno = 0;
        status_1 = unlink(filename);
        if (status_1 == -1) {
            lock_cerr_mutex();
            cerr << "WARNING! In 'purge_server_logs': 'unlink' failed, returning -1:" <<
                endl << strerror(errno) << endl << "Failed to delete file" << filename <<
                "'." << "Will try to continue." << endl;
            unlock_cerr_mutex();
        }
    } /* if (status_1 == -1) */
}

```

1684. *unlink* returns only 0 (upon success) or -1 (upon error), so we just assume here that it returned 0.
[LDF 2013.04.09.]

```
<purge_server_logs definition 1677> +≡
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "In 'purge_server_logs': 'unlink' succeeded, returning " << status_1 << "."
                endl << "Deleted file " << filename << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    } /* if (timestamp < limit) */
    else /* timestamp ≥ limit */
    {
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "timestamp >= limit. Not deleting " << filename << endl << "Breaking."
            endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    break;
} /* else (timestamp ≥ limit) */
} /* if (status > 0) */
} /* do */
while (status > 0 & & !feof(fp) & & !ferror(fp)) ;
```

1685. Read any remaining shell output, so `cut` won't complain about a "write error". [LDF 2013.04.09.]

```
<purge-server-logs definition 1677> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG & status > 0 & !feof(fp) & !ferror(fp)) {
        lock_cerr_mutex();
        cerr << "There may be unread output. Will read and discard." << endl;
        unlock_cerr_mutex();
    }
#endif /* DEBUG_COMPILE */
    while (!!(status == 0 || feof(fp) || ferror(fp))) {
        memset(buffer, 0, 2048);
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Input remaining in stream ("FILE *fp") . Reading into 'buffer' ." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    status = fread(buffer, 1, 2047, fp);
#endif DEBUG_COMPILE
    if (status > 0 & DEBUG) {
        lock_cerr_mutex();
        cerr << "buffer:" << buffer << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    } /* while */
    pclose(fp);
    fp = 0;
#endif /* 1 */
    goto PURGE_SERVER_LOGS_SLEEP; /* Uncomment to skip rotating files. [LDF 2013.04.17.] */
#endif
```

1686. Rotate log and error log files. [LDF 2013.04.10.]

```

⟨purge_server_logs definition 1677⟩ +≡
    status = rotate_log_file(log_filename, now, limit, log_strm_mutex);
    if (status ≡ 2) {
#if DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "'rotate_log_file'"_succeeded,_returning_2." << endl << "Log_file_" <<
                log_filename << "'_does_not_exist.'" << endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    } /* if (status ≡ 2) */
    else if (status ≡ 4) {
#if DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "'rotate_log_file'"_succeeded,_returning_4." << endl << "Log_file_" <<
                log_filename << "'_did_not_need_to_be_rotated.'" << endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    } /* else if (status ≡ 4) */
    else if (status ≠ 0) {
        lock_cerr_mutex();
        cerr << "ERROR! In 'purge_server_logs': 'rotate_log_file'"_failed,_returning_" <<
            status << "." << endl << "Failed_to_rotate_" << log_filename << "." <<
            endl << "Exiting_thread_function_unsuccessfully_with_exit_status_0." <<
            endl << "PLEASE_NOTE: Logs,_temporary_files,_etc.,_will_not_be_purged_" <<
            "until_gwirdsif_is_restarted." << endl;
        unlock_cerr_mutex();
        pthread_exit(0);
    } /* else if (status ≠ 0) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "'rotate_log_file'"_succeeded,_returning_0." << endl << "Rotated_log_file_" <<
            log_filename << "'_successfully.'" << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1687. Error log file. [LDF 2013.04.18.]

```

⟨purge_server_logs definition 1677⟩ +≡
    status = rotate_log_file(err_log_filename, now, limit, err_log_strm_mutex);
    if (status ≡ 2) {
#if DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "'rotate_log_file'" "succeeded," "returning" 2. " << endl << "Error log file" " <<
                err_log_filename << "' does not exist." << endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    } /* if (status ≡ 2) */
    else if (status ≡ 4) {
#if DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "'rotate_log_file'" "succeeded," "returning" 4. " << endl << "Error log file" " <<
                err_log_filename << "' did not need to be rotated." << endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    } /* else if (status ≡ 4) */
    else if (status ≠ 0) {
        lock_cerr_mutex();
        cerr << "ERROR! In 'purge_server_logs': 'rotate_log_file' failed, returning" " <<
            status << "." << endl << "Failed to rotate" " << err_log_filename << "." <<
            endl << "Exiting thread function unsuccessfully with exit status 0." <<
            endl << "PLEASE NOTE: Logs, temporary files, etc., will not be purged" " <<
            "until 'gwirdsif' is restarted." << endl;
        unlock_cerr_mutex();
        pthread_exit(0);
    } /* else if (status ≠ 0) */
#endif /* DEBUG_COMPILE */
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "'rotate_log_file'" "succeeded," "returning" 0. " << endl << "Rotated log file" " <<
            err_log_filename << "' successfully." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1688. Standard output file. This file will exist if the output to standard output, i.e., *stdout* or *cout*, is redirected to the file <log directory>/**gwirdsif.stdout**. [LDF 2013.04.18.]

Log

[LDF 2013.04.18.] Added this section.

```
<purge_server_logs definition 1677> +≡
    string stdout_filename = log_dir;
    stdout_filename += '/';
    stdout_filename += "gwirdsif.stdout";
    status = rotate_log_file(stdout_filename, now, limit, cout_mutex);
    if (status == 2) {
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "'rotate_log_file'" << "succeeded, returning 2." << endl <<
                "Standard output log file '" << stdout_filename << "' does not exist." << endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    } /* if (status == 2) */
    else if (status == 4) {
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "'rotate_log_file'" << "succeeded, returning 4." << endl <<
                "Standard output log file '" << stdout_filename << "'"
                " did not need to be rotated." << endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    } /* else if (status == 4) */
    else if (status != 0) {
        lock_cerr_mutex();
        cerr << "ERROR! In 'purge_server_logs': 'rotate_log_file' failed, returning "
            "status " << "." << endl << "Failed to rotate '" << stdout_filename << "'."
            endl << "Exiting thread function unsuccessfully with exit status 0." <<
            endl << "PLEASE NOTE: Logs, temporary files, etc., will not be purged"
            " until 'gwirdsif' is restarted." << endl;
        unlock_cerr_mutex();
        pthread_exit(0);
    } /* else if (status != 0) */
#endif /* DEBUG_COMPILE */
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "'rotate_log_file'" << "succeeded, returning 0." << endl <<
            "Rotated log file '" << stdout_filename << "' successfully." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1689. Standard error file. This file will exist if the output to standard error, i.e., *stderr* or *cerr*, is redirected to the file <log directory>/**gwirdsif.stderr**. [LDF 2013.04.18.]

Log

[LDF 2013.04.18.] Added this section.

```
<purge_server_logs definition 1677> +≡
    string stderr_filename = log_dir;
    stderr_filename += '/';
    stderr_filename += "gwirdsif.stderr";
    status = rotate_log_file(stderr_filename, now, limit, cerr_mutex, &cout_mutex, &cout);
    if (status ≡ 2) {
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr ≪ "'rotate_log_file' succeeded, returning 2." ≪ endl ≪
                "Standard_error_log_file" ≪ stderr_filename ≪ "' does not exist." ≪ endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    } /* if (status ≡ 2) */
    else if (status ≡ 4) {
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr ≪ "'rotate_log_file' succeeded, returning 4." ≪ endl ≪
                "Standard_error_log_file" ≪ stderr_filename ≪ "'"
                "did not need to be rotated." ≪ endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    } /* else if (status ≡ 4) */
    else if (status ≠ 0) {
        lock_cerr_mutex();
        cerr ≪ "ERROR! In 'purge_server_logs': 'rotate_log_file' failed, returning "
        status ≪ "." ≪ endl ≪ "Failed to rotate" ≪ stderr_filename ≪ "." ≪
            endl ≪ "Exiting thread function unsuccessfully with exit status 0." ≪
            endl ≪ "PLEASE NOTE: Logs, temporary files, etc., will not be purged"
            "until 'gwirdsif' is restarted." ≪ endl;
        unlock_cerr_mutex();
        pthread_exit(0);
    } /* else if (status ≠ 0) */
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr ≪ "'rotate_log_file' succeeded, returning 0." ≪ endl ≪ "Rotated log file"
                stderr_filename ≪ "' successfully." ≪ endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1690. End of code for rotating log and error log files. [LDF 2013.04.18.]

```
<purge_server_logs definition 1677> +≡
PURGE_SERVER_LOGS_SLEEP:
```

1691. Enable cancellation of this thread. [LDF 2013.04.18.]

Log

[LDF 2013.04.18.] Added this section.

```
<purge_server_logs definition 1677> +≡
status = pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, 0);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ "WARNING! In 'purge_server_logs': pthread_setcancelstate failed, "
          ≪ "returning " ≪ status ≪ ":" ≪ endl ≪ strerror(status) ≪ endl ≪
          "Failed to enable thread cancellation. Will try to continue." ≪ endl;
    unlock_cerr_mutex();
} /* if (status ≠ 0) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ "In 'purge_server_logs': pthread_setcancelstate succeeded, "
              ≪ "returning 0." ≪ endl ≪ "Enabled thread cancellation successfully." ≪ endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1692. End of loop. Go to sleep. [LDF 2013.04.18.]

```
<purge_server_logs definition 1677> +≡
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "In 'purge_server_logs': Going to sleep for " ≪ sleep_value ≪ " seconds." ≪ endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
sleep(sleep_value);
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "In 'purge_server_logs': Woke up after sleeping for " ≪ sleep_value ≪
          " seconds." ≪ endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* for */
```

1693. Exit. This code will normally never be reached. [LDF 2013.04.09.]

```
<purge_server_logs definition 1677> +≡
pthread_exit(0); /* End of purge_server_logs definition */
```

1694. Rotate log file. [LDF 2013.04.11.]

Log

[LDF 2013.04.11.] Added this function.

[LDF 2013.04.18.] Added optional arguments `pthread_mutex_t *mutex_1_ptr = 0` and `ostream *out_strm = 0`. They're needed when rotating log files into which the output to standard output and standard error are redirected.

(rotate_log_file declaration 1694) ≡

```
int rotate_log_file(string &filename, time_t now, time_t limit, pthread_mutex_t
&mutex, pthread_mutex_t *mutex_1_ptr = 0, ostream *out_strm = 0);
```

This code is used in sections 1712 and 1713.

1695.

(rotate_log_file definition 1695) ≡

```
int rotate_log_file(string &filename, time_t now, time_t limit, pthread_mutex_t
&mutex, pthread_mutex_t *mutex_1_ptr, ostream *out_strm){ bool DEBUG = false;
/* true */
set_debug_level(DEBUG, 0, 0);
if (mutex_1_ptr == 0) mutex_1_ptr = &cerr_mutex;
if (out_strm == 0) out_strm = &cerr;
#ifndef DEBUG_COMPILE
if (DEBUG) {
    pthread_mutex_lock(mutex_1_ptr);
    *out_strm << "Entering 'rotate_log_file'." << endl << "'filename'" <=
filename << endl;
    pthread_mutex_unlock(mutex_1_ptr);
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
FILE *fp = 0;
stringstream temp_strm;
string rotate_str;
unsigned int line_num;
unsigned int timestamp;
int status = 0;
```

See also sections 1696, 1697, 1698, 1699, 1700, 1701, 1702, 1703, 1704, 1705, 1706, 1707, 1708, 1709, 1710, and 1711.

This code is used in section 1712.

1696.**Log**

[LDF 2013.04.18.] Added this section.

```
<rotate_log_file definition 1695> +≡
  errno = 0;
  status = access(filename.c_str(), F_OK);
  if (status ≡ -1 ∧ errno ≡ ENOENT) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    pthread_mutex_lock(mutex_1_ptr);
    *out_strm ≡ "In 'rotate_log_file': file " ≡ filename ≡ "' doesn't exist." ≡ endl ≡
      "Exiting function successfully with return value 2." ≡ endl;
    pthread_mutex_unlock(mutex_1_ptr);
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  return 2;
} /* if (status ≡ -1) */
else if (status ≡ -1) {
  pthread_mutex_lock(mutex_1_ptr);
  *out_strm ≡ "In 'rotate_log_file': 'access' failed, returning -1: " ≡ endl ≡
    strerror(errno) ≡ endl ≡ "Failed to test existence of " ≡ filename ≡ ". " ≡ endl ≡
      "Exiting function successfully with return value 1." ≡ endl;
  pthread_mutex_unlock(mutex_1_ptr);
  return 1;
} /* else if (status ≡ -1) */
```

1697.

```

<rotate_log_file definition 1695> +≡
    pthread_mutex_lock(&mutex);
vector<unsigned int> line_num_vector;
char buffer[2048];
struct tm tmp;
char date_ext_str[64];
memset(date_ext_str, 0, 64);
if (gmtime_r(&now, &tmp) ≡ 0) {
    pthread_mutex_lock(mutex_1_ptr);
    *out_strm ≪ "ERROR! In 'rotate_log_file': 'gmtime_r' failed, returning 0:" ≪
        "Exiting function unsuccessfully with return value 1." ≪ endl;
    pthread_mutex_unlock(mutex_1_ptr);
    pthread_mutex_unlock(&mutex);
    return 1;
}
if (strftime(date_ext_str, sizeof(date_ext_str), "%Y.%m.%d", &tmp) ≡ 0) {
    pthread_mutex_lock(mutex_1_ptr);
    *out_strm ≪ "ERROR! In 'rotate_log_file': 'strftime' failed, returning 0." ≪ endl ≪
        "Exiting function unsuccessfully with return value 1." ≪ endl;
    pthread_mutex_unlock(mutex_1_ptr);
    pthread_mutex_unlock(&mutex);
    return 1;
}
#if DEBUG_COMPILE
else
    if (DEBUG) {
        pthread_mutex_lock(mutex_1_ptr);
        *out_strm ≪ "date_ext_str == " ≪ date_ext_str ≪ endl;
        pthread_mutex_unlock(mutex_1_ptr);
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1698. Setup commands for rotating log and error log files. [LDF 2013.04.10.]

Log

[LDF 2013.04.10.] Added this section.

```

<rotate_log_file definition 1695> +≡
  temp_strm ≪ "cat" ≪ filename ≪ " | " ≪ "grep -n -e \"^#\" | cut -f1-10 -d \" \" | "
    | tr -d "#\" | tr -s \" \"";
#endif DEBUG_COMPILE
  if (DEBUG) {
    pthread_mutex_lock(mutex_1_ptr);
    *out_strm ≪ "temp_strm.str() == " ≪ temp_strm.str() ≪ endl;
    pthread_mutex_unlock(mutex_1_ptr);
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  rotate_str = temp_strm.str();
  temp_strm.str("");

```

1699.

```
<rotate_log_file definition 1695> +≡
    errno = 0;
    fp = popen(rotate_str.c_str(), "r");
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        pthread_mutex_lock(mutex_1_ptr);
        *out strm ≪ "rotate_str=="
        ≪ rotate_str ≪ endl;
        pthread_mutex_unlock(mutex_1_ptr);
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    if (fp ≡ 0) {
        pthread_mutex_lock(mutex_1_ptr);
        *out strm ≪ "ERROR! In 'rotate_log_file': 'popen' failed, returning NULL: "
        ≪ endl ≪ strerror(errno) ≪ endl ≪ "Command: "
        ≪ rotate_str ≪ endl ≪
        "Exiting function unsuccessfully with return value 1." ≪ endl;
        pthread_mutex_unlock(mutex_1_ptr);
        pthread_mutex_unlock(&mutex);
        return 1;
    } /* for */
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            pthread_mutex_lock(mutex_1_ptr);
            *out strm ≪ "In 'rotate_log_file': 'popen' succeeded."
            pthread_mutex_unlock(mutex_1_ptr);
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1700.

```

⟨ rotate_log_file definition 1695 ⟩ +≡
    char date_str[64]; do { errno = 0;
    memset(date_str, 0, 64);
    status = fscanf(fp, "%u:%u%[a-zA-Z0-9. ,:] ", &line_num, &timestamp, date_str);
    if (status == EOF) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        pthread_mutex_lock(mutex_1_ptr);
        *out_strm << "Reached end of input. Breaking." << endl;
        pthread_mutex_unlock(mutex_1_ptr);
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    break;
}
else if (status != 3) {
    pthread_mutex_lock(mutex_1_ptr);
    *out_strm << "Error! 'fscanf' failed, returning" << status << ":" << endl <<
        strerror(errno) << endl << "Breaking." << endl;
    pthread_mutex_unlock(mutex_1_ptr);
    break;
}
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        pthread_mutex_lock(mutex_1_ptr);
        *out_strm << "'fscanf' succeeded, returning" << 3 << endl << "'line_num'" <=
            line_num << endl << "'timestamp'" <= timestamp << endl << "'date_str'" <=
            date_str << endl;
        pthread_mutex_unlock(mutex_1_ptr);
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1701.

```

⟨ rotate_log_file definition 1695 ⟩ +≡
    if (timestamp ≥ limit) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        pthread_mutex_lock(mutex_1_ptr);
        *out_strm << "'timestamp' >= limit'" << endl << "timestamp == " << timestamp << endl <<
            "limit == " << limit << endl << "Breaking." << endl;
        pthread_mutex_unlock(mutex_1_ptr);
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    break;
} /* if (timestamp ≥ limit) */

```

1702.

```

⟨ rotate_log_file definition 1695 ⟩ +≡
  else /* timestamp < limit */
  {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    pthread_mutex_lock(mutex_1_ptr);
    *out_strm << "timestamp<limit:" << endl << "timestamp=="
      << timestamp << endl << "limit=="
      << limit << endl << "Continuing." << endl;
    pthread_mutex_unlock(mutex_1_ptr);
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  line_num_vector.push_back(line_num);
} /* else (timestamp < limit) */
}
while (status > 0 & & !feof(fp) & & !ferror(fp)) ;

```

1703. Read any unread output. [LDF 2013.04.10.]

```

⟨ rotate_log_file definition 1695 ⟩ +≡
  while (status > 0 & & !(feof(fp) || ferror(fp))) {
    memset(buffer, 0, 2048);
    status = fread(buffer, 1, 2047, fp);
    if (status == 0) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
      pthread_mutex_lock(mutex_1_ptr);
      *out_strm << "'fread' returned 0." << endl;
      pthread_mutex_unlock(mutex_1_ptr);
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    }
    else {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
      pthread_mutex_lock(mutex_1_ptr);
      *out_strm << "'fread' returned" << status << endl << "buffer=="
        << buffer << endl;
      pthread_mutex_unlock(mutex_1_ptr);
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    } /* else */
  } /* while */
  pclose(fp);
  fp = 0;

```

1704.

```
<rotate_log_file definition 1695> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        pthread_mutex_lock(mutex_1_ptr);
        *out_strm << "line_num_vector.size() == " << line_num_vector.size() << endl;
        if (line_num_vector.size() > 0) *out_strm << "Showing 'line_num_vector': " << endl;
        else *out_strm << "'line_num_vector' is empty. Not showing." << endl;
        pthread_mutex_unlock(mutex_1_ptr);
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        for (vector<unsigned int>::iterator iter = line_num_vector.begin(); iter != line_num_vector.end();
            ++iter) {
            pthread_mutex_lock(mutex_1_ptr);
            *out_strm << "*iter == " << *iter << endl;
            pthread_mutex_unlock(mutex_1_ptr);
        } /* for */
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1705.

```

⟨ rotate_log_file definition 1695 ⟩ +≡
  string old_filename = filename;
  old_filename += ".";
  old_filename += date_ext_str;
  int i = 0;
  for ( ; i < 6; ++i) {
    temp_strm.str("");
    temp_strm << old_filename;
    if (i > 0) temp_strm << "." << i;
    errno = 0;
    status = access(temp_strm.str().c_str(), F_OK);
    if (status == -1 & errno == ENOENT) {
      if (i > 0) old_filename = temp_strm.str();
    #if DEBUG_COMPILE
      if (DEBUG) {
        pthread_mutex_lock(mutex_1_ptr);
        *out_strm << "Found\ua\u name\u for\u 'old_filename':\u" << old_filename << endl <<
          "Breaking." << endl;
        pthread_mutex_unlock(mutex_1_ptr);
      } /* if (DEBUG) */
    #endif /* DEBUG_COMPILE */
      break;
    } /* if */
    else if (status == 0) {
    #if DEBUG_COMPILE
      if (DEBUG) {
        pthread_mutex_lock(mutex_1_ptr);
        *out_strm << "File\u ' " << temp_strm.str() << "'\u already\u exists.\u" <<
          "Continuing\u to\u search." << endl;
        pthread_mutex_unlock(mutex_1_ptr);
      } /* if (DEBUG) */
    #endif /* DEBUG_COMPILE */
      continue;
    } /* else if (status == 0) */
    else {
      pthread_mutex_lock(mutex_1_ptr);
      *out_strm << "ERROR! \u In\u 'rotate_log_file':\u 'access'\u failed,\u returning\u -1\u and\u " <<
        "'errno'\u !=\u 'ENOENT':\u" << endl << strerror(errno) << endl <<
        "Failed\u to\u rotate\u log\u file\u ' " << filename << '.' << endl <<
        "Exiting\u function\u unsuccessfully\u with\u return\u value\u 1." << endl;
      pthread_mutex_unlock(mutex_1_ptr);
      pthread_mutex_unlock(&mutex);
      return 1;
    } /* else */
  } /* for */
}

```

1706.

```

⟨ rotate_log_file definition 1695 ⟩ +≡
  if (i == 6) {
    pthread_mutex_lock(mutex_1_ptr);
    *out_strm << "ERROR! In 'rotate_log_file': Files named "
    << old_filename << endl <<
    "with suffixes '1' through '6' already exist." << endl <<
    "This shouldn't happen." << endl << "Failed to rotate log file"
    << filename << endl << "Exiting function unsuccessfully with return value 1."
    << endl;
    pthread_mutex_unlock(mutex_1_ptr);
    pthread_mutex_unlock(&mutex);
    return 1;
  }
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      pthread_mutex_lock(mutex_1_ptr);
      *out_strm << "Found a name for a file that doesn't exist yet:"
      << endl <<
      "old_filename=" << old_filename << endl;
      pthread_mutex_unlock(mutex_1_ptr);
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1707.

```

⟨ rotate_log_file definition 1695 ⟩ +≡
  if (line_num_vector.size() > 0 & line_num_vector.back() > 1) { temp_strm.str("");
  temp_strm << "a='mktemp" << log_dir << "/temp_file.XXXXXX'&&" << "head-n"
  << (line_num_vector.back() - 1) << " " << filename << " >" << old_filename << " "
  << "&&tail-n+" << line_num_vector.back() << " " << filename << " >$a&&mv$a" << filename;
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    pthread_mutex_lock(mutex_1_ptr);
    *out_strm << "temp_strm.str()=" << temp_strm.str() << endl;
    pthread_mutex_unlock(mutex_1_ptr);
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1708.

```

⟨ rotate_log_file definition 1695 ⟩ +≡
  status = system(temp_strm.str().c_str());
  if (status == -1 || !WIFEXITED(status)) {
    pthread_mutex_lock(mutex_1_ptr);
    *out_strm << "ERROR! In 'rotate_log_file': 'system' failed, returning"
    << status << endl;
    if (WIFEXITED(status)) *out_strm << "WEXITSTATUS(status)=" << WEXITSTATUS(status) << endl;
    else *out_strm << "Process failed to exit." << endl;
    *out_strm << "Exiting function unsuccessfully with return value 1."
    << endl;
    pthread_mutex_unlock(mutex_1_ptr);
    pthread_mutex_unlock(&mutex);
    return 1;
  } /* (status == -1 || !WIFEXITED(status)) */

```

1709.

```

⟨ rotate_log_file definition 1695 ⟩ +≡
else
  if (WEXITSTATUS(status) ≠ 0) {
    pthread_mutex_lock(mutex_1_ptr);
    *out_strm << "ERROR! In 'rotate_log_file': " <<
      "'mktemp', 'head', 'tail' or 'mv' command " << "(called via 'system') failed, "
      "returning " << WEXITSTATUS(status) << ". " << endl << "Failed to rotate logfile " <<
      filename << ". " << endl << "Exiting function unsuccessfully with return value 1. " <<
      endl;
    pthread_mutex_unlock(mutex_1_ptr);
    pthread_mutex_unlock(&mutex);
    return 1;
  } /* else if (WEXITSTATUS(status) ≠ 0) */
#endif DEBUG_COMPILE
else
  if (DEBUG) {
    pthread_mutex_lock(mutex_1_ptr);
    *out_strm << "In 'rotate_log_file': system succeeded, returning 0. " << endl <<
      "Rotated logfile " << filename << " successfully. " << endl;
    pthread_mutex_unlock(mutex_1_ptr);
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
temp_strm.str(""); } /* if (line_num_vector.size() > 0 & line_num_vector.back() > 0) */

```

1710.

```

⟨ rotate_log_file definition 1695 ⟩ +≡
else {
#endif DEBUG_COMPILE
  if (DEBUG) {
    pthread_mutex_lock(mutex_1_ptr);
    *out_strm << "In 'rotate_log_file': " << endl << "'line_num_vector.size()' == " <<
      line_num_vector.size() << endl;
    if (line_num_vector.size() > 0)
      *out_strm << "'line_num_vector.back()' == " << line_num_vector.back() << endl;
    *out_strm << "'line_num_vector.size()' == 0 or 'line_num_vector.back()' == 0" <<
      endl << "Not rotating logfile " << filename << ". " << endl <<
      "Exiting function successfully with return value 2. " << endl;
    pthread_mutex_unlock(mutex_1_ptr);
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  pthread_mutex_unlock(&mutex);
  return 4;
} /* else */

```

1711.

```

⟨ rotate_log_file definition 1695 ⟩ +≡
    pthread_mutex_unlock(&mutex);
#if DEBUG_COMPILE
    if (DEBUG) {
        pthread_mutex_lock(mutex_1_ptr);
        *out_strm << "Exiting 'rotate_log_file' successfully with return value 0." << endl;
        pthread_mutex_unlock(mutex_1_ptr);
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; } /* End of rotate_log_file definition */

```

1712. Putting purgfncs together.

This is what's compiled. [LDF 2013.04.09.]

```

using namespace std;
⟨ Include files 3 ⟩
using namespace gwrdifpk;
⟨ purge_server_logs declaration 1676 ⟩
⟨ rotate_log_file declaration 1694 ⟩
⟨ purge_server_logs definition 1677 ⟩
⟨ rotate_log_file definition 1695 ⟩

```

1713. This is what's written to the header file purgfncs.h. [LDF 2013.04.09.]

```

⟨ purgfncs.h 1713 ⟩ ≡
#ifndef PURGFNCS_H
#define PURGFNCS_H 1
using namespace std;
⟨ purge_server_logs declaration 1676 ⟩
⟨ rotate_log_file declaration 1694 ⟩
#endif

```

1714. Utility functions (utilfncts.web).

Log

[LDF 2012.06.27.] Copied from “quali” project and modified.

1715. Include files.

```
<Include files 3> +≡
#ifndef _XOPEN_SOURCE
#define _XOPEN_SOURCE
#endif
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <sys/types.h>
#include <dirent.h>
#include <string.h>
#include <pwd.h>
#include <errno.h>
#include <limits.h>
#include <fcntl.h>
#include <sys/wait.h>
#include <sys/mman.h>
#include <pthread.h>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <string>
#include <sstream>
#include <deque>
#include <map>
#include <vector>
#if HAVE_CONFIG_H
#include <config.h>
#endif
#include <mysql.h>
#include "glblcnst.h++"
#include "glblvrb1.h++"
#include "excptntp.h++"
```

1716. Mutex functions. [LDF 2012.06.27.]

Log

[LDF 2012.06.27.] Added this section.

[LDF 2012.07.27.] Removed the definitions of *lock_cerr_mutex* and *unlock_cerr_mutex*. They are now defined in *otptfnc0.c++* and *otptfnc1.c++*. *cerr_mutex* isn't useful for FastCGI applications, so I don't want to lock and unlock it in functions when they're called from such an application, e.g., *gwrdbap.fcgi*, whereas I must lock and unlock it, when these functions are called from a "normal" application, e.g., *gwirdcli* or *gwirdsif*.

The definitions in *otptfnc0.c++* are therefore empty, except for a **return** statement, and I hope that the compiler will optimize the function calls out.

1717. Lock *cerr_mutex*. [LDF 2012.06.27.]

Log

[LDF 2012.06.27.] Added this function.

1718.

⟨Mutex function declarations 1718⟩ ≡

void *lock_cerr_mutex(void)*;

See also sections 1719, 1721, and 1722.

This code is used in sections 1786 and 1787.

1719. Unlock *cerr_mutex*. [LDF 2012.06.27.]

Log

[LDF 2012.06.27.] Added this function.

⟨Mutex function declarations 1718⟩ +≡

void *unlock_cerr_mutex(void)*;

1720. Lock *cout_mutex*. [LDF 2013.05.02.]

Log

[LDF 2013.05.02.] Added this function. It is defined in *otptfnc0.cpp* and *otptfnc1.cpp*.

1721.

⟨Mutex function declarations 1718⟩ +≡

void *lock_cout_mutex(void)*;

1722. Unlock *cout_mutex*. [LDF 2013.05.02.]

Log

[LDF 2013.05.02.] Added this function. It is defined in *otptfnc0.cpp* and *otptfnc1.cpp*.

⟨Mutex function declarations 1718⟩ +≡

void *unlock_cout_mutex(void)*;

1723. Submit SQL query. [LDF 2012.07.11.]

Log

[LDF 2012.07.11.] Added this function. I've copied it from the version I wrote for the OptiNum-Grid project and modified it slightly.

[LDF 2012.07.16.] Moved this function from `scprpmtp.web` to this file (`utilfncs.web`). Changed it, so that it is no longer a member function of `Scan_Parse_Parameter_Type`. Added the required argument `MYSQL * mysql_ptr` and the optional argument `string thread_ctr_str` with default `""`. About to add a corresponding member function for `Scan_Parse_Parameter_Type` that calls this function.

[LDF 2012.09.20.] BUG FIX: Now locking and unlocking `pthread_mutex_t sql_mutex`. Previously, the connection to the MySQL server was lost when I called the client multiple times.

[LDF 2013.04.18.] Now calling `pthread_setcancelstate` to disable and reenable thread cancellation. It's disabled when this function is entered and reenabled when it exits. Thread cancellation should not occur while this function is executing, otherwise the database may be left in an invalid state.

[LDF 2013.04.19.] When disabling thread cancellation, now saving old thread cancellation state in `int old_cancel_state`. When `pthread_setcancelstate` is called again before exiting, the old state is restored. Previously, cancellation was enabled. !! PLEASE NOTE: The result of the call to `pthread_setcancelstate` when attempting to restore the old state is only checked when this function exits successfully. That is, upon error exits, the result of `pthread_setcancelstate` is not checked. However, normally it should succeed.

[LDF 2013.07.14.] BUG FIX: Changed the argument `MYSQL * mysql_ptr` to `MYSQL * &mysql_ptr`. This ensures that the pointer variable passed as `mysql_ptr` is really set to 0. Previously, this wasn't happening for `Scan_Parse_Parameter_Type::mysql_ptr` and a segmentation fault error occurred in the `Scan_Parse_Parameter_Type` destructor.

```
{ submit_mysql_query declaration 1723 } ≡
int submit_mysql_query(string query, MYSQL_RES *&result, MYSQL *&mysql_ptr, unsigned int
    *row_ctr = 0, unsigned int *field_ctr = 0, long *affected_rows = 0, string thread_str = "");
```

This code is used in sections 1786 and 1787.

1724.

```

⟨ submit_mysql_query definition 1724 ⟩ ≡
int submit_mysql_query(string query, MYSQL_RES *&result, MYSQL * &mysql_ptr, unsigned int
    *row_ctr, unsigned int *field_ctr, long *affected_rows, string thread_str){ int status;
int DEBUG = false; /* true */
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "Entering 'submit_mysql_query'." << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
int old_cancel_state;
status = pthread_setcancelstate(PTHREAD_CANCEL_DISABLE, &old_cancel_state);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << "WARNING! In 'submit_mysql_query': 'pthread_setcancelstate' failed, "
        "returning " << status << ":" << endl << strerror(status) << endl <<
        "Failed to disable thread cancellation. Will try to continue." << endl;
    unlock_cerr_mutex();
} /* if (status ≠ 0) */
#endif DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'submit_mysql_query': 'pthread_setcancelstate' succeeded, "
        "returning 0." << endl << "Disabled thread cancellation successfully." << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
pthread_mutex_lock(&sql_mutex);
if (affected_rows) *affected_rows = 0L;
status = mysql_query(mysql_ptr, query.c_str());
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'submit_mysql_query':"
        "mysql_query failed, returning " << status << ":" << endl << "Error: "
        mysql_error(mysql_ptr) << endl << "Error number: " << mysql_errno(mysql_ptr) << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    pthread_mutex_unlock(&sql_mutex);
    pthread_setcancelstate(old_cancel_state, 0);
    return 1;
} /* if (mysql_query failed.) */
result = mysql_store_result(mysql_ptr);
if (result ≡ 0) {
    if (row_ctr) {
        lock_cerr_mutex();
        cerr << thread_str << "WARNING! In 'submit_mysql_query':"
            "mysql_store_result returned " << result << endl << mysql_error(mysql_ptr) <<
            endl << "Exiting function with return value 0." << endl;
        unlock_cerr_mutex();
        pthread_mutex_unlock(&sql_mutex);
    }
}

```

```

pthread_setcancelstate(old_cancel_state, 0);
return 0;
} /* if (row_ctr) */
else if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'submit_mysql_query':"
        << endl <<
        "'mysql_store_result' returned 0." << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
} /* if (No result) */
if (row_ctr == 0 || field_ctr == 0) {
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'submit_mysql_query':"
            << endl <<
            "'row_ctr' and/or 'field_ctr' is NULL."
            << endl <<
            "Not storing rows or columns returned." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
} /* if (row_ctr == 0 || field_ctr == 0) */
else {
    *row_ctr = mysql_num_rows(result);
    *field_ctr = mysql_num_fields(result);
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'submit_mysql_query':"
            << endl << "*row_ctr" <=>
            *row_ctr << endl << "*field_ctr" <=>
            *field_ctr << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
} /* else */

```

See also sections 1725, 1726, and 1727.

This code is used in section 1786.

1725.

Log

[LDF 2012.04.30.] Added this section.

```

⟨ submit_mysql_query definition 1724 ⟩ +≡
if (affected_rows != 0) {
    *affected_rows = (long) mysql_affected_rows(mysql_ptr);
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'submit_mysql_query':"
            << "*affected_rows" <=>
            *affected_rows << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
} /* if (affected_rows != 0) */

```

1726.

```
( submit_mysql_query definition 1724 ) +≡
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "Exiting 'submit_mysql_query' "
      << "successfully with return value 0." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
  pthread_mutex_unlock(&sql_mutex);
```

1727.

```
( submit_mysql_query definition 1724 ) +≡
  status = pthread_setcancelstate(old_cancel_state, 0);
  if (status != 0) {
    lock_cerr_mutex();
    cerr << "WARNING! In 'submit_mysql_query': pthread_setcancelstate failed, "
      << "returning " << status << ":" << endl << strerror(status) << endl <<
      "Failed to restore previous thread cancellation state. Will try to continue."
      << endl;
    unlock_cerr_mutex();
  } /* if (status != 0) */
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "In 'submit_mysql_query': pthread_setcancelstate succeeded, "
        << "returning 0." << endl << "Restored previous thread cancellation state"
        << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  return 0; } /* End of submit_mysql_query */
```

1728. Submit multiple SQL queries (*submit_mysql_queries*). [LDF 2013.08.19.]

Log

[LDF 2013.08.19.] Added this function.

[LDF 2013.09.03.] Changed the **vector** arguments so that references are passed (instead of passing them by value, which entails copying them and their contents).

```
( submit_mysql_queries declaration 1728 ) ≡
  int submit_mysql_queries(vector<string> &query_vector, MYSQL_RES **result_array, MYSQL * &mysql_ptr,
    vector<unsigned int *> &row_ctr_vector, vector<unsigned int *> &field_ctr_vector, vector<long int *> &affected_rows_vector, bool continue_on_error = false, string thread_str = "");
```

This code is used in sections 1786 and 1787.

1729.

```
{ submit_mysql_queries definition 1729 }≡
int submit_mysql_queries(vector<string> &query_vector, MYSQL_RES **result_array,
    MYSQL *&mysql_ptr, vector<unsigned int *> &row_ctr_vector, vector<unsigned int *>
    &field_ctr_vector, vector<long int *> &affected_rows_vector, bool continue_on_error, string
    thread_str){ bool DEBUG = false; /* true */
set_debug_level(DEBUG, 0, 0);
int status;
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "Entering 'submit_mysql_queries'." << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
```

See also section 1730.

This code is used in section 1786.

1730.

```

⟨ submit_mysql_queries definition 1729 ⟩ +≡
    int i = 0;
    for (vector<string>::iterator iter = query_vector.begin(); iter ≠ query_vector.end(); ++iter) {
        if (iter->empty()) {
#if DEBUG_COMPILE
            if (DEBUG) {
                lock_cerr_mutex();
                cerr << thread_str << "In 'submit_mysql_queries': " << endl << "query_vector[" << i <<
                    "]' is empty." << "Not calling 'submit_mysql_query'." << endl;
                unlock_cerr_mutex();
            } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        }
        ++i;
        continue;
    } /* if (iter->empty()) */
    status = submit_mysql_query(*iter, result_array[i], mysql_ptr, row_ctr_vector[i], field_ctr_vector[i],
                                affected_rows_vector[i], thread_str);
    if (status ≠ 0) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'submit_mysql_queries': " << endl <<
            "'submit_mysql_query' failed, returning " << status << "." << endl <<
            "'i'(iteration_counter)== " << i << endl << "Query: " << *iter << endl;
        if (continue_on_error) {
            cerr << "'continue_on_error'== 'true'. Will try to continue." << endl;
        }
        else {
            cerr << "'continue_on_error'== 'false'." << endl <<
                "Exiting function unsuccessfully with return value 1." << endl;
        }
        unlock_cerr_mutex();
        if (result_array[i]) {
            mysql_free_result(result_array[i]);
            result_array[i] = 0;
        }
        if (continue_on_error) {
            ++i;
            continue;
        }
        else return 1;
    } /* if (status ≠ 0) */
#endif DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_str << "In 'submit_mysql_queries': " << endl <<
                "'submit_mysql_query' succeeded, returning 0." << endl <<
                "'i'(iteration_counter)== " << i << endl << "Query: " << *iter << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    ++i;

```

```

} /* for */
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "Exiting `submit_mysql_queries' successfully with return value 0." <<
        endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
return 0; } /* End of submit_mysql_queries definition */

```

1731. Get datestamp. [LDF 2010.03.11.]

Log

[LDF 2010.03.11.] Added this function.

[LDF 2011.12.22.] Copied this function from `glblfncs.web` in the `dbsrvcli/src` directory and modified it:

1. Now outputting timestamp in GMT format for use as the value of the `expire` field of a cookie.
2. Added optional `int hour_offset` and `int min_offset` arguments. The default for both is 0.

[LDF 2012.07.23.] Copied this function from `lbrack...]/optinum/Installer/optwbsrv/src/utilfncs.web` of the OptiNum-Grid project.

[LDF 2012.07.24.] Added optional argument `time_t *seconds` with default 0. If non-zero, the number of seconds since the epoch corresponding to the timestamp returned is stored in `*seconds`.

[LDF 2013.02.08.] Added calls to `lock_cerr_mutex` and `unlock_cerr_mutex`. Now conditionally compiling the debugging output.

[LDF 2013.04.10.] Added optional arguments `unsigned int sse = 0`, `string separator = " "`, `string delim_0 = ""` and `string delim_1 = ""`. “sse” stands for “seconds since the Epoch”. If `sse ≡ 0` the seconds since the Epoch aren’t output. If it’s `sse ≡ 1`, they are output by themselves. If `sse > 1`, they are output followed by `separator` and the human-readable timestamp. The latter is preceded by `delim_0` and followed by `delim_1`. If `delim_1` is the empty string and `delim_0` is not, then `delim_1` is set to `delim_0`.

!! TODO Look up `clock_t`, `times`, etc., to get processor times. This isn’t urgent. [LDF 2010.03.11.]

⟨ `get_datestamp` declaration 1731 ⟩ ≡

```

string get_datestamp(int hour_offset = 0, int min_offset = 0, time_t *seconds = 0, unsigned int
    sse = 0, string delim_0 = "", string delim_1 = "", string separator = " ");

```

This code is used in sections 1786 and 1787.

1732.

```

⟨ get_datestamp definition 1732 ⟩ ≡
string get_datestamp(int hour_offset, int min_offset, time_t *seconds, unsigned int sse, string
    delim_0, string delim_1, string separator){ bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    string s = "";
    char outstr[64];
    memset(outstr, 0, 64);
    time_t t;
    struct tm tmp;
    t = time(Λ);
    if (hour_offset > 0) t += 3600 * hour_offset;
    if (min_offset > 0) t += 600 * min_offset;
    if (gmtime_r(&t, &tmp) ≡ 0) {
        lock_cerr_mutex();
        cerr ≪ "ERROR! In 'get_datestamp': 'gmtime_r' failed, returning 0: " ≪
            endl ≪ "Not creating datestamp. Exiting function with empty 'string' " ≪
            "as return value." ≪ endl;
        unlock_cerr_mutex();
        if (seconds) *seconds = 0;
        return s;
    }
    if (sse > 0) {
        if (strftime(outstr, sizeof(outstr), "%s", &tmp) ≡ 0) {
            lock_cerr_mutex();
            cerr ≪ "ERROR! In 'get_datestamp': 'strftime' failed, returning 0. " ≪
                endl ≪ "Not creating datestamp. Exiting function with empty 'string' " ≪
                "as return value." ≪ endl;
            unlock_cerr_mutex();
            if (seconds) *seconds = 0;
            return s;
        }
        else {
            s = outstr;
        }
    }
    #if DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr ≪ "Result string is " ≪ s ≪ endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
    #endif /* DEBUG_COMPILE */
}
} /* if (sse > 0) */
if (sse ≡ 1) {
    if (seconds) *seconds = t;
    return s;
}

```

See also section 1733.

This code is used in section 1786.

1733.

```
<get_datestamp definition 1732> +≡
  if (¬delim_0.empty() ∧ delim_1.empty()) delim_1 = delim_0;
  if (strftime(outstr, sizeof(outstr), "%a,%d%b%Y%T%Z", &tmp) ≡ 0) {
    lock_cerr_mutex();
    cerr ≪ "ERROR! In 'get_datestamp': strftime failed, returning 0." ≪
      endl ≪ "Not creating timestamp. Exiting function with empty string" ≪
      "as return value." ≪ endl;
    unlock_cerr_mutex();
    if (seconds) *seconds = 0;
    return string("");
  }
  else {
    if (sse ≥ 2) {
      s += separator;
      s += delim_0;
      s += outstr;
      s += delim_1;
    }
    else {
      s += delim_0;
      s = outstr;
      s += delim_1;
    }
  }
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "Result string is" ≪ s ≪ endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  if (seconds) *seconds = t;
  return s;
} /* else */
} /* End of get_datestamp definition */
```

1734. Convert seconds. [LDF 2013.02.08.]

This function creates a **string** from a **time_t**. The **string** can be used in SQL commands for setting a field of type **timestamp**. [LDF 2013.07.10.]

If the argument **bool timezone** ≡ *true*, the three-letter abbreviation for the timezone is included at the end of the **string** returned by this function.

If the argument **bool utc** ≡ *true*, “Universal Time”, i.e., “UTC” is used. Otherwise, localtime is used, which is what the MySQL database expects. [LDF 2013.07.25.]

Log

[LDF 2013.02.08.] Added this function.

[LDF 2013.07.10.] Made the argument **time_t seconds** optional, with default **static_cast<time_t>(0)**.

[LDF 2013.07.15.] Now calling *localtime_r* instead of *gmtime_r*. Added optional argument **bool timezone** = *false*.

⟨ *convert_seconds* declaration 1734 ⟩ ≡

```
string convert_seconds(time_t seconds = static_cast<time_t>(0), bool timezone = false, bool  
                      utc = false);
```

This code is used in sections 1786 and 1787.

1735.

```

⟨ convert_seconds definition 1735 ⟩ ≡
string convert_seconds(time_t seconds, bool timezone, bool utc)
{
    bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    string s;
    char outstr[64];
    memset(outstr, 0, 64);
    struct tm tmp;
    if (seconds ≡ static_cast<time_t>(0)) {
        errno = 0;
        seconds = time(0);
        if (seconds ≡ static_cast<time_t>(-1)) {
            lock_cerr_mutex();
            cerr << "ERROR! In 'convert_seconds': 'time' failed, returning -1: " <<
                endl << strerror(errno) << endl << "Can't create timestamp. Exiting function\\
                on with empty string" << "as return value." << endl;
            unlock_cerr_mutex();
            return s;
        }
        /* if (seconds ≡ static_cast<time_t>(0)) */
        if (utc) {
            if (gmtime_r(&seconds, &tmp) ≡ 0) {
                lock_cerr_mutex();
                cerr << "ERROR! In 'convert_seconds': 'gmtime_r' failed, returning 0: " <<
                    endl << "Not creating timestamp. Exiting function with empty string" <<
                    "as return value." << endl;
                unlock_cerr_mutex();
                return s;
            }
        }
        else {
            if (localtime_r(&seconds, &tmp) ≡ 0) {
                lock_cerr_mutex();
                cerr << "ERROR! In 'convert_seconds': 'localtime_r' failed, returning 0: " <<
                    endl << "Not creating timestamp. Exiting function with empty string" <<
                    "as return value." << endl;
                unlock_cerr_mutex();
                return s;
            }
        }
        /* else (¬utc) */
    }
    string format_str = "%Y-%m-%d %H:%M:%S";
    if (timezone ∧ utc) format_str += " UTC";
    else if (timezone) format_str += " %Z";
    if (strftime(outstr, sizeof(outstr), format_str.c_str(), &tmp) ≡ 0) {
        lock_cerr_mutex();
        cerr << "ERROR! In 'convert_seconds': 'strftime' failed, returning 0. " <<
            endl << "Not creating timestamp. Exiting function with empty 'string' " <<
            "as return value." << endl;
    }
}
```

```

    unlock_cerr_mutex();
    return s;
}
else {
    s = outstr;
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Result_string_is_" << s << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return s;
} /* else */
} /* End of convert_seconds definition */

```

This code is used in section 1786.

1736. Get seconds since the Epoch. [LDF 2013.05.15.]

Log

[LDF 2013.05.15.] Added this function.

```

⟨ get_seconds_since_epoch declaration 1736 ⟩ ≡
int get_seconds_since_epoch(const char *timestamp, time_t &sse,
    string format_str = "%Y-%m-%d %H:%M:%S");

```

This code is used in sections 1786 and 1787.

1737.

```

⟨ get_seconds_since_epoch definition 1737 ⟩ ≡
int get_seconds_since_epoch(const char *timestamp, time_t &sse, string format_str)
{
    size_t status;
    struct tm tm;
    char buffer[16];
    memset(buffer, 0, 16);
    char *temp_str = strptime(timestamp, format_str.c_str(), &tm);
    if (temp_str == 0) {
        lock_cerr_mutex();
        cerr << "ERROR! In 'get_seconds_since_epoch': 'strptime' failed, returning NULL." <<
            endl << "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        return 1;
    }
    status = strftime(buffer, 16, "%s", &tm);
    if (status == 0) {
        lock_cerr_mutex();
        cerr << "ERROR! In 'get_seconds_since_epoch': 'strftime' failed, returning 0." <<
            endl << "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        return 1;
    }
    errno = 0;
    unsigned long temp_val = strtoul(buffer, 0, 10);
    if (temp_val == ULONG_MAX) {
        lock_cerr_mutex();
        cerr << "ERROR! In 'get_seconds_since_epoch': 'strtoul' failed, returning ULONG_MAX:" <<
            endl << strerror(errno) << endl <<
            "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        return 1;
    }
    sse = temp_val;
    return 0;
} /* End of get_seconds_since_epoch definition */

```

This code is used in section 1786.

1738. Convert time specification. [LDF 2013.08.07.]

Log

[LDF 2013.08.07.] Added this function.

```

⟨ convert_time_spec declaration 1738 ⟩ ≡
unsigned long int convert_time_spec(string time_spec);

```

This code is used in sections 1786 and 1787.

1739.

```
{ convert_time_spec definition 1739 } ≡
  unsigned long int convert_time_spec(string time_spec){ bool DEBUG = false;      /* true */
    set_debug_level(DEBUG, 0, 0);
    unsigned long int ret_val = 0UL;
    string temp_str;
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "Entering 'convert_time_spec'. 'time_spec' == " << time_spec << endl;
    unlock_cerr_mutex();
  }      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */
```

See also sections 1740, 1741, 1742, 1743, 1744, 1745, 1746, and 1747.

This code is used in section 1786.

1740.

```
{ convert_time_spec definition 1739 } +≡
  unsigned long int temp_val[4];
  for (int i = 0; i < 4; ++i) temp_val[i] = 0UL;
  size_t pos;
  int i = 0;
  int j = 0;
  pos = time_spec.find(":");
  if (pos ≡ string::npos) {
    lock_cerr_mutex();
    cerr << "ERROR! In 'convert_time_spec': No ':' characters in 'strin\g_time_spec' argument." << endl << "This shouldn't be possible." << endl << "Exiting function unsuccessfully with return value 'ULONG_MAX'." << endl;
    unlock_cerr_mutex();
    return ULONG_MAX;
  }      /* if (pos ≡ string::npos) */
```

1741.

```
{ convert_time_spec definition 1739 } +≡
  for (int i = 0; i < 4; ++i) {
    if (time_spec.length() ≡ 0) {
#ifndef DEBUG_COMPILE
      if (DEBUG) {
        lock_cerr_mutex();
        cerr << "'time_spec' is empty. Breaking." << endl;
        unlock_cerr_mutex();
      }      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */
      break;
    }      /* if (time_spec.length() ≡ 0) */
```

1742.

```
⟨ convert_time_spec definition 1739 ⟩ +≡
  pos = time_spec.find(":");
  if (pos ≡ 0) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "‘pos’==0. First character == ‘:’." << "Setting ‘temp_val["
      << i <<
      "]’ to 0UL and continuing." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  time_spec.erase(0,1);
  temp_val[i] = 0UL;
  continue;
} /* if (pos ≡ 0) */
```

1743.

```

⟨ convert_time_spec definition 1739 ⟩ +≡
else {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "pos" << "=";
        if (pos == string::npos) cerr << "string::npos";
        else cerr << pos << ">0";
        cerr << "First character != :" << "Will set temp_val[" << i << "] ." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    temp_str = "";
    j = 0;
    while (isdigit(time_spec[j])) {
        temp_str += time_spec[j++];
    }
    time_spec.erase(0, j + 1);
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "j==" << j << endl << "temp_str==" << temp_str << endl <<
            "time_spec==" << time_spec << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    temp_val[i] = strtoul(temp_str.c_str(), 0, 10);
    if (temp_val[i] == ULONG_MAX) {
        lock_cerr_mutex();
        cerr << "ERROR! In 'convert_time_spec': strtoul failed, returning ULONG_MAX." <<
            endl << "Failed to convert 'temp_str'==" << temp_str <<
            " to an unsigned long int." << endl <<
            "Exiting function unsuccessfully with return value ULONG_MAX." << endl;
        unlock_cerr_mutex();
        return ULONG_MAX;
    } /* if (temp_val[i] == ULONG_MAX) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'convert_time_spec': strtoul succeeded, returning" << temp_val[i] <<
            "." << endl << "Set temp_val[" << i << "] to" << temp_val[i] << "." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* else */
if (pos == string::npos) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "pos==" << "string::npos" << "Breaking." << endl;

```

```

        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
break;
}
} /* for */

```

1744.

```

⟨ convert_time_spec definition 1739 ⟩ +≡
#ifndef DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    for (int i = 0; i < 4; ++i) {
        cerr << "temp_val[" << i << "] " << temp_val[i] << endl;
    }
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1745. !! TODO: Add error handling to ensure that *ret_val* < ULONG_MAX. The multiplication may cause overflow, but only if unreasonable values are used. [LDF 2013.08.07.]

```

⟨ convert_time_spec definition 1739 ⟩ +≡
temp_val[0] *= 86400UL; /* days */
temp_val[1] *= 3600UL; /* hours */
temp_val[2] *= 60UL; /* minutes */
ret_val = temp_val[0];
for (int i = 1; i < 4; ++i) {
    if (ULONG_MAX - ret_val > temp_val[i]) ret_val += temp_val[i];
    else {
        lock_cerr_mutex();
        cerr << "ERROR! In 'convert_time_spec': Time specification >= 'ULONG_MAX'." <<
            endl << "This isn't permitted." << endl <<
            "Exiting function unsuccessfully with return value 'ULONG_MAX'." << endl;
        unlock_cerr_mutex();
        return ULONG_MAX;
    }
} /* for */

```

1746.

```

⟨ convert_time_spec definition 1739 ⟩ +≡
#ifndef DEBUG_COMPILE
if (DEBUG) {
    time_t plus_spec = time(0) + ret_val;
    time_t minus_spec = time(0) - ret_val;
    lock_cerr_mutex();
    cerr << "plus_spec=" << plus_spec << " " << convert_seconds(plus_spec,
        true) << endl << "minus_spec=" << minus_spec << " " << convert_seconds(minus_spec,
        true) << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1747.

```
< convert_time_spec definition 1739 > +≡
#if DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "Exiting ‘convert_time_spec’ successfully with return value " ≪ ret_val ≪ "."
    endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  return ret_val; } /* End of convert_time_spec definition */
```

1748. Convert binary data to hexadecimal string and vice versa (*hexl_encode* and *hexl_decode*). [LDF 2013.04.26.] ■

Log

[LDF 2013.04.26.] Added this section.

1749. *hexl_encode*. [LDF 2013.04.26.]

Log

[LDF 2013.04.26.] Added this function.

```
< hexl_encode declaration 1749 > ≡
int hexl_encode(const char *buffer, unsigned int buffer_size, string &result, int delimiter = -1, int delimiter_1 = -1);
```

This code is used in sections 1786 and 1787.

1750.

```

⟨ hexl_encode definition 1750 ⟩ ≡
int hexl_encode(const char *buffer, unsigned int bufferSize, string &result, int delimiter, int
    delimiter_1){ bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    stringstream temp_strm;
    int status;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Entering 'hexl_encode'." << endl;
        cerr << "buffer == " << endl;
        fwrite(buffer, 1, bufferSize, stderr);
        cerr << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    if (delimiter > -1) {
        temp_strm << static_cast<unsigned char>(delimiter);
    }
    unsigned char c;
    for (int i = 0; i < bufferSize; ++i) {
        c = buffer[i];
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "c(hex) == " << setw(2) << setfill('0') << hex << static_cast<unsigned
                int>(c) << endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        temp_strm << setw(2) << setfill('0') << hex << static_cast<unsigned int>(c);
    } /* for */
    if (delimiter_1 > -1) {
        temp_strm << static_cast<unsigned char>(delimiter_1);
    }
    else if (delimiter > -1) {
        temp_strm << static_cast<unsigned char>(delimiter);
    }
#endif /* DEBUG_COMPILE */
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'hexl_encode': temp_strm.str() == " << temp_strm.str() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    result = temp_strm.str();

```

See also section 1751.

This code is used in section 1786.

1751.

```
( hexl_encode definition 1750 ) +≡
#if DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "Exiting 'hexl_encode' successfully with return value 0." ≪ endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  return 0; } /* End of hexl_encode definition */
```

1752. hexl_decode. [LDF 2013.04.26.]

Log

[LDF 2013.04.26.] Added this function.

[LDF 2013.04.28.] Replaced arguments **char *buffer** and **unsigned int *buffer_size** with **string &dest** and **unsigned int &dest_length**.

(hexl_decode declaration 1752) ≡

```
int hexl_decode(string &source, string &dest, unsigned int &dest_length);
```

This code is used in sections 1786 and 1787.

1753.

(hexl_decode definition 1753) ≡

```
int hexl_decode(string &source, string &dest, unsigned int &dest_length){ bool DEBUG = false;
  /* true */
  set_debug_level(DEBUG, 0, 0);
  stringstream temp_strm;
  int status;
#if DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "Entering 'hexl_decode' ." ≪ endl;
    cerr ≪ "source== " ≪ endl ≪ source ≪ endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

See also sections 1754 and 1755.

This code is used in section 1786.

1754.

```

⟨ hexl_decode definition 1753 ⟩ +≡
unsigned long curr_val;
char curr_str[3];
curr_str[2] = '\0';
dest = "";
dest_length = 0;
for (int i = 0; i < source.length(); i += 2) {
#if DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "source[" << i << "]'" <= source[i] << endl << "source[" << (i + 1) << "]'" <=
            source[i + 1] << endl << "Current_hexadecimal_number_string:'" <= source[i] <<
                source[i + 1] << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    curr_str[0] = source[i];
    curr_str[1] = source[i + 1];
    errno = 0;
    curr_val = strtoul(curr_str, 0, 16);
    if (curr_val == ULONG_MAX) {
        lock_cerr_mutex();
        cerr << "ERROR! In 'hexl_decode': 'strtoul' failed, returning 'ULONG_MAX': " <<
            endl << strerror(errno) << endl << "Failed to convert 'curr_str' == '" <<
                curr_str << "\\' to a'" << "numerical value." << endl <<
                    "Exiting function unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        return 1;
    } /* if (curr_val == ULONG_MAX) */
#if DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "In 'hexl_decode': 'strtoul' succeeded, returning" << curr_val << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    dest += static_cast<unsigned char>(curr_val);
    ++dest_length;
} /* for */
#if DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "dest ='" << endl;
        fwrite(dest.c_str(), 1, dest_length, stderr);
        cerr << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1755.

```
< hexl_decode definition 1753 > +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Exiting 'hexl_decode' successfully with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; } /* hexl_decode */
```

1756. Initialize signal maps. [LDF 2013.05.02.]

```
< initialize_signal_maps declaration 1756 > ≡
void initialize_signal_maps(void);
```

This code is used in sections 1786 and 1787.

1757.

```
< initialize_signal_maps definition 1757 > ≡
void initialize_signal_maps(void)
{
    signal_name_map[1] = "SIGHUP";
    signal_name_map[2] = "SIGINT";
    signal_name_map[3] = "SIGQUIT";
    signal_name_map[4] = "SIGILL";
    signal_name_map[5] = "SIGTRAP";
    signal_name_map[6] = "SIGABRT";
    signal_name_map[7] = "SIGBUS";
    signal_name_map[8] = "SIGFPE";
    signal_name_map[9] = "SIGKILL";
    signal_name_map[10] = "SIGUSR1";
    signal_name_map[11] = "SIGSEGV";
    signal_name_map[12] = "SIGUSR2";
    signal_name_map[13] = "SIGPIPE";
    signal_name_map[14] = "SIGALRM";
    signal_name_map[15] = "SIGTERM";
    signal_name_map[16] = "SIGSTKFLT";
    signal_name_map[17] = "SIGCHLD";
    signal_name_map[18] = "SIGCONT";
    signal_name_map[19] = "SIGSTOP";
    signal_name_map[20] = "SIGTSTP";
    signal_name_map[21] = "SIGTTIN";
    signal_name_map[22] = "SIGTTOU";
    signal_name_map[23] = "SIGURG";
    signal_name_map[24] = "SIGXCPU";
    signal_name_map[25] = "SIGXFSZ";
    signal_name_map[26] = "SIGVTALRM";
    signal_name_map[27] = "SIGPROF";
    signal_name_map[28] = "SIGWINCH";
    signal_name_map[29] = "SIGIO";
    signal_name_map[30] = "SIGPWR";
    signal_name_map[31] = "SIGSYS";
    signal_name_map[34] = "SIGRTMIN";
    signal_name_map[64] = "SIGRTMAX";
    signal_number_map.insert(make_pair("SIGHUP", 1));
    signal_number_map.insert(make_pair("SIGINT", 2));
    signal_number_map.insert(make_pair("SIGQUIT", 3));
    signal_number_map.insert(make_pair("SIGILL", 4));
    signal_number_map.insert(make_pair("SIGTRAP", 5));
    signal_number_map.insert(make_pair("SIGABRT", 6));
    signal_number_map.insert(make_pair("SIGBUS", 7));
    signal_number_map.insert(make_pair("SIGFPE", 8));
    signal_number_map.insert(make_pair("SIGKILL", 9));
    signal_number_map.insert(make_pair("SIGUSR1", 10));
    signal_number_map.insert(make_pair("SIGSEGV", 11));
    signal_number_map.insert(make_pair("SIGUSR2", 12));
    signal_number_map.insert(make_pair("SIGPIPE", 13));
    signal_number_map.insert(make_pair("SIGALRM", 14));
    signal_number_map.insert(make_pair("SIGTERM", 15));
```

```
signal_number_map.insert(make_pair("SIGSTKFLT", 16));
signal_number_map.insert(make_pair("SIGCHLD", 17));
signal_number_map.insert(make_pair("SIGCONT", 18));
signal_number_map.insert(make_pair("SIGSTOP", 19));
signal_number_map.insert(make_pair("SIGTSTP", 20));
signal_number_map.insert(make_pair("SIGTTIN", 21));
signal_number_map.insert(make_pair("SIGTTOU", 22));
signal_number_map.insert(make_pair("SIGURG", 23));
signal_number_map.insert(make_pair("SIGXCPU", 24));
signal_number_map.insert(make_pair("SIGXFSZ", 25));
signal_number_map.insert(make_pair("SIGVTALRM", 26));
signal_number_map.insert(make_pair("SIGPROF", 27));
signal_number_map.insert(make_pair("SIGWINCH", 28));
signal_number_map.insert(make_pair("SIGIO", 29));
signal_number_map.insert(make_pair("SIGPWR", 30));
signal_number_map.insert(make_pair("SIGSYS", 31));
signal_number_map.insert(make_pair("SIGRTMIN", 34));
signal_number_map.insert(make_pair("SIGRTMAX", 64));
signal_number_map.insert(make_pair("HUP", 1));
signal_number_map.insert(make_pair("INT", 2));
signal_number_map.insert(make_pair("QUIT", 3));
signal_number_map.insert(make_pair("ILL", 4));
signal_number_map.insert(make_pair("TRAP", 5));
signal_number_map.insert(make_pair("ABRT", 6));
signal_number_map.insert(make_pair("BUS", 7));
signal_number_map.insert(make_pair("FPE", 8));
signal_number_map.insert(make_pair("KILL", 9));
signal_number_map.insert(make_pair("USR1", 10));
signal_number_map.insert(make_pair("SEGV", 11));
signal_number_map.insert(make_pair("USR2", 12));
signal_number_map.insert(make_pair("PIPE", 13));
signal_number_map.insert(make_pair("ALRM", 14));
signal_number_map.insert(make_pair("TERM", 15));
signal_number_map.insert(make_pair("STKFLT", 16));
signal_number_map.insert(make_pair("CHLD", 17));
signal_number_map.insert(make_pair("CONT", 18));
signal_number_map.insert(make_pair("STOP", 19));
signal_number_map.insert(make_pair("TSTP", 20));
signal_number_map.insert(make_pair("TTIN", 21));
signal_number_map.insert(make_pair("TTOU", 22));
signal_number_map.insert(make_pair("URG", 23));
signal_number_map.insert(make_pair("XCPU", 24));
signal_number_map.insert(make_pair("XFSZ", 25));
signal_number_map.insert(make_pair("VTALRM", 26));
signal_number_map.insert(make_pair("PROF", 27));
signal_number_map.insert(make_pair("WINCH", 28));
signal_number_map.insert(make_pair("IO", 29));
signal_number_map.insert(make_pair("PWR", 30));
signal_number_map.insert(make_pair("SYS", 31));
signal_number_map.insert(make_pair("RTMIN", 34));
signal_number_map.insert(make_pair("RTMAX", 64));
signal_number_map.insert(make_pair("sighup", 1));
```

```
signal_number_map.insert(make_pair("sigint", 2));
signal_number_map.insert(make_pair("sigquit", 3));
signal_number_map.insert(make_pair("sigill", 4));
signal_number_map.insert(make_pair("sigtrap", 5));
signal_number_map.insert(make_pair("sigabrt", 6));
signal_number_map.insert(make_pair("sigbus", 7));
signal_number_map.insert(make_pair("sigfpe", 8));
signal_number_map.insert(make_pair("sigkill", 9));
signal_number_map.insert(make_pair("sigusr1", 10));
signal_number_map.insert(make_pair("sigsegv", 11));
signal_number_map.insert(make_pair("sigusr2", 12));
signal_number_map.insert(make_pair("sigpipe", 13));
signal_number_map.insert(make_pair("sigalrm", 14));
signal_number_map.insert(make_pair("sigterm", 15));
signal_number_map.insert(make_pair("sigstkflt", 16));
signal_number_map.insert(make_pair("sigchld", 17));
signal_number_map.insert(make_pair("sigcont", 18));
signal_number_map.insert(make_pair("sigstop", 19));
signal_number_map.insert(make_pair("sigtstp", 20));
signal_number_map.insert(make_pair("sigttin", 21));
signal_number_map.insert(make_pair("sigttou", 22));
signal_number_map.insert(make_pair("sigurg", 23));
signal_number_map.insert(make_pair("sigxcpu", 24));
signal_number_map.insert(make_pair("sigxfsz", 25));
signal_number_map.insert(make_pair("sigvtalarm", 26));
signal_number_map.insert(make_pair("sigprof", 27));
signal_number_map.insert(make_pair("sigwinch", 28));
signal_number_map.insert(make_pair("sigio", 29));
signal_number_map.insert(make_pair("sigpwr", 30));
signal_number_map.insert(make_pair("sigsys", 31));
signal_number_map.insert(make_pair("sigrtmin", 34));
signal_number_map.insert(make_pair("sigrtmax", 64));
signal_number_map.insert(make_pair("hup", 1));
signal_number_map.insert(make_pair("int", 2));
signal_number_map.insert(make_pair("quit", 3));
signal_number_map.insert(make_pair("ill", 4));
signal_number_map.insert(make_pair("trap", 5));
signal_number_map.insert(make_pair("abrt", 6));
signal_number_map.insert(make_pair("bus", 7));
signal_number_map.insert(make_pair("fpe", 8));
signal_number_map.insert(make_pair("kill", 9));
signal_number_map.insert(make_pair("usr1", 10));
signal_number_map.insert(make_pair("segv", 11));
signal_number_map.insert(make_pair("usr2", 12));
signal_number_map.insert(make_pair("pipe", 13));
signal_number_map.insert(make_pair("alrm", 14));
signal_number_map.insert(make_pair("term", 15));
signal_number_map.insert(make_pair("stkflt", 16));
signal_number_map.insert(make_pair("chld", 17));
signal_number_map.insert(make_pair("cont", 18));
signal_number_map.insert(make_pair("stop", 19));
signal_number_map.insert(make_pair("tstp", 20));
```

```

signal_number_map.insert(make_pair("ttin",21));
signal_number_map.insert(make_pair("ttoi",22));
signal_number_map.insert(make_pair("urg",23));
signal_number_map.insert(make_pair("xcpu",24));
signal_number_map.insert(make_pair("xfsz",25));
signal_number_map.insert(make_pair("vtalrm",26));
signal_number_map.insert(make_pair("prof",27));
signal_number_map.insert(make_pair("winch",28));
signal_number_map.insert(make_pair("io",29));
signal_number_map.insert(make_pair("pwr",30));
signal_number_map.insert(make_pair("sys",31));
signal_number_map.insert(make_pair("rtmin",34));
signal_number_map.insert(make_pair("rtmax",64));
return;
} /* End of initialize_signal_maps definition */

```

This code is used in section 1786.

1758. Set debug level. [LDF 2013.05.10.]

Log

[LDF 2013.05.10.] Added this function. Copied from `dbsrvcli` (OptiNum-Grid).

[LDF 2013.07.25.] Now using `int global_debug_level` instead of `int trace_value`. Both of these are global variables, declared in `glblvrbl.web`. `trace_value` is currently not used for anything.

[LDF 2013.07.25.] Added the optional arguments `int turn_on_value = 0` and `int turn_off_value = 0`. The callers of this function can use them to set when debugging output is enabled and/or disabled.

(set_debug_level declaration 1758) ≡

`int set_debug_level(bool &DEBUG, int turn_on_value = 0, int turn_off_value = 0);`

This code is used in sections 1786 and 1787.

1759.

```

⟨ set_debug_level definition 1759 ⟩ ≡
int set_debug_level(bool &DEBUG, int turn_on_value, int turn_off_value)
{
    bool debug_level; /* Using this variable prevents the assignment below from being found by grep,
                        when I search for places in the code where debugging has been turned on. [LDF 2012.02.07.] */
    if (turn_on_value < 0) {
        lock_cerr_mutex();
        cerr << "ERROR! In 'set_debug_level': Invalid value for 'turn_on_value' " <<
            turn_on_value << endl << "Value must be >= 0." << endl <<
            "Not changing 'DEBUG'. Exiting function unsuccessfully with return value 1." <<
            endl;
        unlock_cerr_mutex();
        return 1;
    } /* if (turn_on_value < 0) */
    if (turn_off_value > 0) {
        lock_cerr_mutex();
        cerr << "ERROR! In 'set_debug_level': Invalid value for 'turn_off_value' " <<
            turn_off_value << endl << "Value must be <= 0." << endl <<
            "Not changing 'DEBUG'. Exiting function unsuccessfully with return value 1." <<
            endl;
        unlock_cerr_mutex();
        return 1;
    } /* if (turn_off_value < 0) */
    if (global_debug_level ≡ 0) /* Don't change the value of DEBUG */
        return 0;
    else if (global_debug_level ≥ turn_on_value) {
        debug_level = true;
        DEBUG = debug_level;
    }
    else if (global_debug_level ≤ turn_off_value) {
        debug_level = false;
        DEBUG = debug_level;
    }
    return 0;
} /* End of set_debug_level definition */

```

This code is used in section 1786.

1760. Check iRODS server (*check_irods_server*). Restart, if necessary. [LDF 2013.06.07.]

!! PLEASE NOTE: This doesn't seem to be completely fool-proof: It's possible for the iRODS server to be started without the (PostgreSQL) database server being started. When this happened, the call to *system* still returned 0. [LDF 2013.06.07.]

Log

[LDF 2013.06.07.] Added this function.

```

⟨ check_irods_server declaration 1760 ⟩ ≡
int check_irods_server(int &pid, int thread_ctr = -1);

```

This code is used in sections 1786 and 1787.

1761.

```
< check_irods_server definition 1761 > ≡
  int check_irods_server(int &pid, int thread_ctr){ bool DEBUG = false;      /* true */
    set_debug_level(DEBUG, 0, 0);
    string thread_str;
    stringstream temp_strm;
    if (thread_ctr ≥ 0) {
      temp_strm ≪ "[Thread" ≪ thread_ctr ≪ "]";
      thread_str = temp_strm.str();
      temp_strm.str("");
    }      /* if (thread_ctr ≥ 0) */
#endif DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ thread_str ≪ "Entering 'check_irods_server'." ≪ endl;
    unlock_cerr_mutex();
  }      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */
  int status;
```

See also sections 1762, 1763, 1764, 1765, 1766, 1767, and 1768.

This code is used in section 1786.

1762. If the iRODS server is running, get its PID using `ps` via `popen`. Otherwise, try to start it and then get its PID. The loop is for getting back to the call to `popen` after trying to start the iRODS server using `system`. It will only try again once, i.e., the loop will iterate at most two times. [LDF 2013.06.07.]

```
< check_irods_server definition 1761 > +≡
  char buffer[32];
  int ret_val = -1;
  string temp_str;
  string temp_str_1;
  FILE *fp = 0;
```

1763.

```

⟨ check_irods_server definition 1761 ⟩ +≡
  long int temp_val;
  size_t pos; for (int i = 0; i < 2; ++i) { temp_val = 0L;
  fp = 0;
  temp_strm.str("");
  temp_strm << "a='ps-C-irodsReServer-o pid='; r=$?; echo \"$r:$a\"";
#ifndef 0
  cerr << "temp_strm.str() == " << temp_strm.str() << endl;
#endif
  fp = popen(temp_strm.str().c_str(), "r");
  if (fp == 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'check_irods_server': "
      "'popen' failed, returning NULL." << endl <<
      "Failed to execute 'ps' in shell to find iRODS server PID." << endl <<
      "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    return 1;
  } /* if (fp == 0) */
}

```

1764.

```

⟨ check_irods_server definition 1761 ⟩ +≡
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << thread_str << "In 'check_irods_server': 'popen' succeeded." << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  memset(buffer, 0, 32);
  status = fread(buffer, 1, 32, fp);
  if (status == 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'check_irods_server': 'fread' failed, returning 0." <<
      endl << "Failed to read shell output." << endl <<
      "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    pclose(fp);
    return 1;
  } /* if (status == 0) */
  else if (status == 32) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'check_irods_server': 'fread' returned 32." <<
      endl << "Too many bytes in shell output. This isn't permitted." << endl <<
      "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    pclose(fp);
    return 1;
  } /* if (status == 32) */
}

```

1765.

```
<check_irods_server definition 1761> +≡
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_str << "In 'check_irods_server': " << "'fread' succeeded, returning" <<
                status << "." << endl << "Read read shell output successfully." << endl <<
                "'buffer'" << buffer << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
pclose(fp);
fp = 0;
```

1766.

```

⟨ check_irods_server definition 1761 ⟩ +≡
temp_strm.str("");
temp_str = "";
temp_str_1 = "";
temp_str = buffer;
temp_str_1 = buffer;
pos = temp_str.find(':');
if (pos ≡ string::npos) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'check_irods_server': string::find' failed, "
        "returning 'string::npos'." << endl << "Failed to find ':' character in 'popen' \
        output." << endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    return 1;
} /* if (pos ≡ string::npos) */
temp_str.erase(pos);
temp_str_1.erase(0, pos + 1);
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "temp_str== " << temp_str << endl << "temp_str_1== " << temp_str_1 << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
errno = 0;
temp_val = strtol(temp_str.c_str(), 0, 10);
if (temp_val ≡ LONG_MIN ∨ temp_val ≡ LONG_MAX) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'check_irods_server': 'strtol' failed, "
        "returning ";
    cerr << ((temp_val ≡ LONG_MIN) ? "LONG_MIN" : "LONG_MAX") << ":" << endl;
    cerr << strerror(errno) << endl << "Failed to get return value of 'ps' command. "
        "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    return 1;
} /* if (temp_val ≡ LONG_MIN ∨ temp_val ≡ LONG_MAX) */
ret_val = temp_val;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "ret_val== " << ret_val << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
if (ret_val ≡ 0) {
    temp_val = 0L;
    errno = 0;
    temp_val = strtol(temp_str_1.c_str(), 0, 10);
    if (temp_val ≡ LONG_MIN ∨ temp_val ≡ LONG_MAX) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'check_irods_server': 'strtol' failed, "
            "returning ";
    }
}

```

```

cerr << ((temp_val == LONG_MIN) ? "LONG_MIN" : "LONG_MAX") << ":" << endl;
cerr << strerror(errno) << endl << "Failed_to_get_iRODS_server_PID_from_popen' output." <<
    endl << "Exiting_function_unsuccessfully_with_return_value_1." << endl;
unlock_cerr_mutex();
return 1;
} /* if (temp_val == LONG_MIN ∨ temp_val == LONG_MAX) */
pid = temp_val;
#endif /* DEBUG_COMPILE */
#if DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "pid==" << pid << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
#if DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'check_irods_server':" << endl << "ret_val==" << ret_val <<
        endl << "pid==" << pid << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
if (pid < 2) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'check_irods_server': pid has invalid value:" <<
        pid << endl << "Exiting_function_unsuccessfully_with_return_value_1." << endl;
    unlock_cerr_mutex();
    return 1;
}
else break;
} /* if (ret_val == 0) */
else /* ret_val ≠ 0 */
{
    temp_strm.str("");
    temp_strm << irods_server_dir << "/irodsctl_start";
#endif 0
cerr << "temp_strm.str() == " << temp_strm.str() << endl;
#endif
status = system(temp_strm.str().c_str());
if (status == -1 ∨ !WIFEXITED(status)) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'check_irods_server':" << endl <<
        "'system' failed, returning" << status << "." << endl;
    if (WIFEXITED(status)) cerr << "WEXITSTATUS(status) == " << WEXITSTATUS(status) << endl;
    else cerr << "Process failed_to_exit." << endl;
    cerr << "Exiting_function_unsuccessfully_with_return_value_1." << endl;
    unlock_cerr_mutex();
    return 1;
} /* (status == -1 ∨ !WIFEXITED(status)) */
else if (WEXITSTATUS(status) ≠ 0) {
    lock_cerr_mutex();
}

```

```

    cerr << thread_str << "ERROR! In 'check_irods_server':" << endl <
        "'irodsctl_start' failed, returning" << WEXITSTATUS(status) <<
        "." << endl << "Failed to start iRODS server." << endl <<
        "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    return 1;
} /* else if (WEXITSTATUS(status) != 0) */
else {
    temp_strm.str("");
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "In 'check_irods_server': system succeeded,\n"
            " returning 0." << endl << "Started iRODS server successfully" << endl <<
            "Iterating again, to try to get PID." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    continue;
} /* else */
} /* else (ret_val != 0) */
} /* for */

```

1767.

```

< check_irods_server definition 1761 > +≡
if (ret_val != 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'check_irods_server':" << endl << "'ret_val' == " << ret_val <<
        " (!= 0)" << endl << "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    return 1;
} /* if (ret_val != 0) */

```

1768.

```

< check_irods_server definition 1761 > +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Exiting 'check_irods_server' successfully with return value 0." <<
            endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0; } /* End of check_irods_server definition */

```

1769. Decrypt (*decrypt*). [LDF 2013.09.24.]

This function can be used to decrypt files that have encrypted with GPG public keys. It could be adapted to decrypt text from strings, as well.

It's needed in order to avoid making the passphrase visible in the `/proc/<PID>/cmdline` entry for the process in which `gpg` runs.

In order to accomplish this, this function writes the passphrase to a FIFO and `gpg` reads it out of the FIFO. The name of the FIFO is passed to `gpg` as the argument to its `--passphrase-file` option. [LDF 2013.09.25.]

Log

[LDF 2013.09.24.] Added this function.

[LDF 2013.09.26.] Added code for decrypting using a secret key without a passphrase. Added optional argument `char end_char = '\n'`. This makes it possible to decrypt texts that contain newlines. !! TODO: Test this!

[LDF 2013.09.27.] Added `bool is_file` argument. Removed code for creating and opening FIFO. This is now done in `main` for `gwirdsif`. I will probably do the same for `gwirdcli`, too. Added required argument `size_t plain_text_length`.

`<decrypt declaration 1769>` ≡

```
int decrypt(string encrypted_text, bool is_file, char *&plain_text, size_t plain_text_length, char
           *passphrase = 0, char end_char = '\n', string thread_str = "");
```

This code is used in sections 1786 and 1787.

1770.

`<decrypt definition 1770>` ≡

```
int decrypt(string encrypted_text, bool is_file, char *&plain_text, size_t plain_text_length, char
           *passphrase, char end_char, string thread_str){ bool DEBUG = false; /* true */
  set_debug_level(DEBUG, 0, 0); /* WARNING! Turning on debugging may cause the passphrase
                                and the decrypted text to be written to standard error! In most places it's commented-out, but
                                there's no guarantee for this. [LDF 2013.09.27.] */
  stringstream temp_strm;
  int status = 0;
  size_t buffer_size = (plain_text_length > 1024) ? (plain_text_length + 512) : 1024;
  char temp_buffer[buffer_size];
  memset(temp_buffer, 0, buffer_size);
#ifndef DEBUG_COMPILE
  if (DEBUG) { /* WARNING! Turning on debugging will cause the passphrase and the decrypted
                text to be written to standard error! [LDF 2013.09.27.] */
    lock_cerr_mutex();
    cerr << thread_str << "Entering 'decrypt'." << endl;
    if (passphrase == 0) cerr << "passphrase==0" << endl;
    else cerr << "passphrase!=0" << endl;
    if (passphrase == 0) cerr << "'passphrase' is NULL." << endl;
    else if (strlen(passphrase) == 0) cerr << "'passphrase' is non-null, but empty." << endl;
    else cerr << "passphrase==" << passphrase << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

See also sections 1771, 1772, 1773, 1774, 1775, 1776, 1777, 1778, 1779, and 1780.

This code is used in section 1786.

1771.

```

⟨ decrypt definition 1770 ⟩ +≡
    char buffer[buffer_size];
    memset(buffer, 0, buffer_size);
    size_t buffer_contents_length = 0;
    errno = 0;
    status = mlock(buffer, buffer_size);
    if (status == -1) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'decrypt': 'mlock' failed, returning -1:" <<
            endl << strerror(errno) << endl << "Failed to lock memory for 'buffer'." << endl <<
            "Exiting 'decrypt' unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        return 1;
    } /* if (status == -1) */
#endif /* DEBUG_COMPILE */
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'decrypt': 'mlock' succeeded, returning 0." << endl <<
        "Locked memory for 'buffer' successfully." << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1772. If *passphrase* doesn't end in a newline (and it shouldn't), a newline character must be appended to it. Otherwise, *gw_gpg_decrypt.sh* will block, presumably because of the way gpg reads from the file specified as the argument to its `--passphrase-file` option. [LDF 2013.09.25.]

Log

[LDF 2013.09.26.] BUG FIX: Now copying *passphrase* to *buffer*, which has been locked into memory with *mlock*. Previously, it was stored in *temp_strm.str()*, which could have been swapped out to disk.

```

⟨ decrypt definition 1770 ⟩ +≡
if (gpg_passphrase != 0 & strlen(gpg_passphrase) > 0) { temp_strm.str("");}
strncpy(buffer, passphrase, strlen(passphrase));
buffer[strlen(passphrase)] = '\n';
errno = 0;
status = write(gpg_passphrase_fifo_fd, buffer, strlen(buffer));
if (status == -1) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'decrypt': 'write' failed, returning -1:" <<
        endl << strerror(errno) << endl << "Failed to write passphrase to FIFO '" <<
        gpg_passphrase_fifo_name << "'." << endl << "Exiting 'decrypt' unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    return 1;
}

```

1773.

```
( decrypt definition 1770 ) +≡
else
  if (status ≠ strlen(buffer)) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'decrypt': write returned" << status <<
      " " << "(!= 'strlen(buffer)' ==)" << strlen(buffer) << ")" << endl <<
      "Failed to write passphrase to FIFO" << gpg_passphrase_fifo_name << '.' << endl <<
      "Exiting 'decrypt' unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    return 1;
  }
```

1774.

```
( decrypt definition 1770 ) +≡
#ifndef DEBUG_COMPILE
else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'decrypt': write succeeded, returning" << status << "."
    endl << "Wrote passphrase to FIFO" << gpg_passphrase_fifo_name << "' successfully."
    endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (gpg_passphrase ≠ 0 ∧ strlen(gpg_passphrase) > 0) */
```

1775.

```

⟨ decrypt definition 1770 ⟩ +≡
    memset(buffer, 0, buffer_size);
    temp_strm.str("");
    if (!is_file) temp_strm << "a='echo\\" << encrypted_text << "\\|\"";
    else temp_strm << "a='";
    temp_strm << "gpg--batch--decrypt";
    if (!gpg_passphrase_fifo_name.empty())
        temp_strm << "--passphrase-file" << gpg_passphrase_fifo_name << ',';
    if (is_file) temp_strm << "," << encrypted_text << ',';
    temp_strm << "2>/dev/null"; r=$?; echo $r; echo $"a"';
#endif /* DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'decrypt': " << endl << "temp_strm.str() == " << temp_strm.str() << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
errno = 0;
FILE *fp = popen(temp_strm.str().c_str(), "r");
if (fp == 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'decrypt': 'popen' failed, returning NULL: " << endl <<
        strerror(errno) << endl << "Call to 'gpg' in shell failed." << endl;
    if (is_file) cerr << "Failed to decrypt file " << encrypted_text << ". " << endl;
    else cerr << "Failed to decrypt 'encrypted_text'." << endl;
    cerr << "Exiting 'decrypt' unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    memset(buffer, 0, buffer_size);
    munlock(buffer, buffer_size);
    return 1;
}
#endif /* DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'decrypt': 'popen' succeeded." << endl;
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1776.

```

⟨ decrypt definition 1770 ⟩ +≡
status = fread(buffer, 1, buffer_size - 1, fp);
if (status ≡ 0) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'decrypt': 'fread' failed, returning 0:" << endl <<
        "Failed to read output of the call to 'gpg' in shell." << endl;
    if (is_file) cerr << "Failed to decrypt file " << encrypted_text << ". " << endl;
    else cerr << "Failed to decrypt 'encrypted_text'." << endl;
    cerr << "Exiting 'decrypt' unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    pclose(fp);
    memset(buffer, 0, buffer_size);
    munlock(buffer, buffer_size);
    return 1;
} /* if (status ≡ 0) */
else if (status ≡ (buffer_size - 1)) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'decrypt': 'fread' returned " <<
        status << " << "(buffer_size - 2) << ":" << endl <<
        "Output of the call to 'gpg' in shell exceeds the maximum size, " <<
        "i.e., " << (buffer_size - 2) << " characters." << endl; /* WARNING! If enabled, this will
cause the decrypted text to be written to standard error! [LDF 2013.09.27.] */
#endif
    cerr << "Exiting 'decrypt' unsuccessfully with return value 1." << endl;
unlock_cerr_mutex();
pclose(fp);
memset(buffer, 0, buffer_size);
munlock(buffer, buffer_size);
return 1;
} /* if (status ≡ (buffer_size - 1)) */
#endif DEBUG_COMPILE
else
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'decrypt': 'fread' succeeded, returning " << status << ". " <<
        endl << "Read output of the call to 'gpg' in shell successfully." << endl;
    /* WARNING! If enabled, this will cause the decrypted text to be written to standard error!
[LDF 2013.09.27.] */
#endif 0
    cerr << "buffer" <=> buffer << endl << buffer << endl;
#endif
    unlock_cerr_mutex();
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
buffer_contents_length = status;
pclose(fp);
fp = 0;

```

1777. Process the output from gpg. [LDF 2013.09.25.]

Log

[LDF 2013.09.25.] Added this section.

```

⟨ decrypt definition 1770 ⟩ +≡
int i = 0;
temp_strm.str("");
while (isspace(buffer[i])) /* Skip over leading whitespace */
    ++i;
if (!isdigit(buffer[i])) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'decrypt': First non-whitespace character in 'b\
        uffer' is not a digit:" << endl << "buffer[" << i << "]' == " <<
        buffer[i] << endl << "Failed to read return value of 'gpg'." << endl <<
        "Exiting 'decrypt' unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    memset(buffer, 0, buffer_size);
    munlock(buffer, buffer_size);
    return 1;
} /* if */
while (isdigit(buffer[i])) temp_strm << buffer[i++];
#endif /* DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "temp_strm.str() == " << temp_strm.str() << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
unsigned long int temp_val = strtoul(temp_strm.str().c_str(), 0, 10);
if (temp_val == ULONG_MAX) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'decrypt': First non-whitespace character in 'b\
        uffer' is not a digit:" << endl << "buffer[" << i << "]' == " <<
        buffer[i] << endl << "Failed to read return value of 'gpg'." << endl <<
        "Exiting 'decrypt' unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    memset(buffer, 0, buffer_size);
    munlock(buffer, buffer_size);
    return 1;
}
#endif /* DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "temp_val == " << temp_val << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1778.

```

⟨ decrypt definition 1770 ⟩ +≡
  if (temp_val ≠ 0UL) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'decrypt':"
    << endl <<
    "gpg' failed, returning 'temp_val'" << temp_val << "."
    << endl;
    if (is_file) cerr << "Failed to decrypt file"
    << encrypted_text << "."
    << endl;
    else cerr << "Failed to decrypt 'encrypted_text'."
    << endl; /* WARNING! If enabled, this
      will cause the decrypted text to be written to standard error! [LDF 2013.09.27.] */
  #if 0
    cerr << "'buffer'" << endl << buffer << endl;
  #endif
    cerr << "Exiting 'decrypt' unsuccessfully with return value 1."
    << endl;
    unlock_cerr_mutex();
    memset(buffer, 0, buffer_size);
    munlock(buffer, buffer_size);
    return 1;
  } /* if (temp_val ≠ 0UL) */
  #if DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << thread_str << "In 'decrypt':"
      << endl << "'gpg' succeeded, returning 0."
      << endl;
      if (is_file) cerr << "Decrypted file"
      << encrypted_text << "' successfully."
      << endl;
      else cerr << "Decrypted 'encrypted_text' successfully."
      << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
  #endif /* DEBUG_COMPILE */

```

1779.

```

⟨ decrypt definition 1770 ⟩ +≡
  while (i < buffer_contents_length ∧ buffer[i] ≠ '\n')      /* Skip to next newline */
    ++i;
  if (i < buffer_contents_length - 1)      /* Skip over the newline */
    ++i;
  temp_strm.str("");
  int j = 0;
  memset(plain_text, 0, plain_text_length);
  while (i < buffer_contents_length ∧ buffer[i] ≠ end_char) {
    if (j > plain_text_length - 2) {
      lock_cerr_mutex();
      cerr << thread_str << "ERROR! In 'decrypt':"
      << endl << "Length of decrypted text exceeds 'plain_text_length'-1:" << endl <<
      "'j' == " << j << endl << "'plain_text_length'" == "
      plain_text_length << endl << "'plain_text_length-2'" ==
      (plain_text_length - 2) << endl << "Failed to set 'plain_text'." << endl <<
      "Exiting 'decrypt' unsuccessfully with return value 1." << endl;
      unlock_cerr_mutex();
      memset(plain_text, 0, plain_text_length);
      memset(buffer, 0, buffer_size);
      munlock(buffer, buffer_size);
      return 1;
    }
    else plain_text[j++] = buffer[i++];
  }
#if DEBUG_COMPILE
  if (DEBUG) {      /* WARNING! This writes the decrypted text to standard error! [LDF 2013.09.27.] */
#if 0
  lock_cerr_mutex();
  cerr << "plain_text == " << plain_text << endl;
  unlock_cerr_mutex();
#endif
  }      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */

```

1780.

```

⟨ decrypt definition 1770 ⟩ +≡
  memset(buffer, 0, buffer_size);
  munlock(buffer, buffer_size);
#if DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "Exiting 'decrypt' successfully with return value 0." << endl;
    unlock_cerr_mutex();
  }      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */
  return 0;      /* End of decrypt definition */

```

1781. Set password. [LDF 2013.09.26.]

Log

[LDF 2013.09.26.] Added this function.

```
{ set_password declaration 1781 } ≡  
int set_password(string filename, char *&password, size_t password_length, string default_filename, char  
*passphrase = 0, string directory = "", string thread_str = "");
```

This code is used in sections 1786 and 1787.

1782.

```

⟨ set_password definition 1782 ⟩ ≡
int set_password(string filename, char *&password, size_t password_length, string default_filename, char
                 *passphrase, string directory, string thread_str){ bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status = 0;
    stringstream temp_strm;
    bool first_time = true;
    bool start_empty = (filename.empty()) ? true : false;
    CHECK_FILENAME:
    if (!filename.empty()) {
        if (directory == config_dir) {
#if DEBUG_COMPILE
            if (DEBUG) {
                lock_cerr_mutex();
                cerr << "'directory'" <= "config_dir" <= " " << directory << endl <<
                    "Setting 'first_time' to " <= "false" . " << endl;
                unlock_cerr_mutex();
            } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
            first_time = false;
        }
#if DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "filename is not empty." << endl << "filename" <= " " << filename << endl;
            unlock_cerr_mutex();
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        string temp_filename;
        size_t pos = filename.find("/");
        if (pos == string::npos) {
#if DEBUG_COMPILE
            if (DEBUG) {
                lock_cerr_mutex();
                cerr << "filename does not contain a slash: " << endl << "filename" <= " " << filename <<
                    endl << "Will prepend directory: " <= "directory" <= " " << directory << "." << endl;
                unlock_cerr_mutex();
            } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
            temp_filename += directory;
            temp_filename += "/";
            temp_filename += filename;
        }
        else {
#if DEBUG_COMPILE
            if (DEBUG) {
                lock_cerr_mutex();
                cerr << "filename contains a slash: " << endl << "filename" <= " " << filename << endl <<
                    "Not prepending 'directory' " <= " " << directory << '.' << endl;
                unlock_cerr_mutex();
            } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        }
    }
}

```

```

        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
temp_filename = filename;
} /* else */
errno = 0;
status = access(temp_filename.c_str(), F_OK);
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "access returned status == " << status << endl << strerror(errno) << endl;
    if (errno == ENOENT) cerr << "ENOENT" << endl;
    else cerr << "Not ENOENT" << endl;
    cerr << "start_empty == " << start_empty << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
if (status == 0) /* File exists */
{
    filename = temp_filename;
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << thread_str << "In 'set_password': " << "'filename'" << " exists." << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (status == 0) */
else if (start_empty == true & first_time == false) {
    lock_cerr_mutex();
    cerr << thread_str << "NOTICE: In 'set_password': Default password file " <<
        default_filename << "' doesn't exist." << endl << "Not using password." <<
        endl << "PLEASE NOTE: This is only for testing purposes. Do not do this" <<
        "unless you understand the risks (see documentation)." << endl;
    unlock_cerr_mutex();
    filename = "";
    goto END_DECRYPT_PASSWORD;
}
else if (status == -1 & errno != ENOENT) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'set_password': 'access' failed, " <<
        "returning -1: " << endl << strerror(errno) << endl <<
        "Failed to check for 'temp_filename' == " << temp_filename <<
        " ." << endl << "Can't set 'filename'." << endl <<
        "Exiting 'set_password' unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    return 1;
} /* else if (status == -1 & errno != ENOENT) */
else if (status == -1 & errno == ENOENT & directory != config_dir & pos == string::npos & first_time ==
        true) {
    lock_cerr_mutex();
}

```

```

cerr << thread_str << "In 'set_password': " << File << temp_filename << endl <<
    "doesn't exist in directory " << directory << endl << strerror(errno) <<
    endl << "Will check for it in 'config_dir' == " << config_dir << endl <<
    endl;
unlock_cerr_mutex();
temp_filename = filename;
filename = config_dir;
filename += "/";
filename += temp_filename;
first_time = false;
goto CHECK_FILENAME;
} /* else if */
else if (start_empty == false) {
    lock_cerr_mutex();
    cerr << thread_str << "ERROR! In 'set_password': " << File <<
        temp_filename << endl << "doesn't exist." << endl << strerror(errno) <<
        endl << "Failed to check for 'temp_filename' == " <<
        temp_filename << endl << "Can't set 'filename'." << endl <<
        "Exiting 'set_password' unsuccessfully with return value 1." << endl;
unlock_cerr_mutex();
return 1;
} /* else */
} /* if (!filename.empty()) */
if (filename.empty()) {
    filename = default_filename;
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "filename was empty." << endl << "Will check for default 'filename' == " <<
        endl << filename << endl;
unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
goto CHECK_FILENAME;
} /* if */

```

See also section 1783.

This code is used in section 1786.

1783. Decrypt password. [LDF 2013.09.25.]

```

⟨ set_password definition 1782 ⟩ +≡
  if (gpg_passphrase == 0) {
#if DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "gpg_passphrase==0" << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
}
else if (gpg_passphrase ≠ 0 ∧ strlen(gpg_passphrase) < 2) {
  cerr << thread_str << "ERROR! In 'set_password':" << endl << "'gpg_passphrase'!=0 and 'strlen(gpg_passphrase)'<2." << endl << "This isn't permitted." << endl << "Exiting 'set_password' unsuccessfully with return value 1." << endl;
  return 1;
} /* else if */
#endif /* DEBUG_COMPILE */
else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "gpg_passphrase!=0" << endl << "gpg_passphrase==" << gpg_passphrase << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
if (password ≡ 0) /* It should be 0 at this point. [LDF 2013.09.25.] */
{
  password = new char[password_length];
  memset(password, 0, password_length);
  mlock(password, password_length); /* Unlocked in finish. [LDF 2013.09.27.] */
}
/* if (password ≡ 0) */
status = decrypt(filename, true, password, password_length, gpg_passphrase, '\n', temp_strm.str());
if (status ≠ 0) {
  lock_cerr_mutex();
  cerr << "ERROR! In 'set_password':" << endl << "'decrypt' failed, returning" <<
    status << "." << endl << "Failed to decrypt password." << endl <<
    "Exiting 'set_password' unsuccessfully with return value 1." << endl;
  unlock_cerr_mutex();
  return 1;
} /* if (status ≠ 0) */
#endif /* DEBUG_COMPILE */
else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'set_password':" << endl << "'decrypt' succeeded, returning 0." << endl <<
      "Decrypted password successfully." << endl << "'password'==" << password << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
END_DECRYPT_PASSWORD:
#endif /* DEBUG_COMPILE */
  if (DEBUG) {

```

```
lock_cerr_mutex();
cerr << thread_str << "Exiting 'set_password' successfully with return value 0." << endl;
unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
return 0; } /* End of set_password definition */
```

1784. Write to FIFO (*write_to_fifo*). [LDF 2013.09.27.]

```
< write_to_fifo declaration 1784 > ≡
int write_to_fifo(char *buffer, size_t buffer_length, int fd, bool write_string_length = true, bool
append_newline = true, string thread_str = "");
```

This code is used in sections 1786 and 1787.

1785.

```

⟨ write_to_fifo definition 1785 ⟩ ≡
int write_to_fifo(char *buffer, size_t buffer_length, int fd, bool write_string_length, bool
                  append_newline, string thread_str)
{
    bool DEBUG = true;      /* false */
    set_debug_level(DEBUG, 0, 0);
    int status = 0;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Entering 'write_to_fifo'." << endl;
        cerr << "fd==" << fd << endl;
        cerr << "buffer==" << buffer << endl;
        cerr << "buffer_length==" << buffer_length << endl;
        unlock_cerr_mutex();
    }      /* if (DEBUG) */
#endif      /* DEBUG_COMPILE */
    errno = 0;
    if (write_string_length == true) {
        status = write(fd, buffer, strlen(buffer));
    }
    else {
        status = write(fd, buffer, buffer_length);
    }
    if (append_newline == true) write(fd, "\n", 1);      /* GPG will block until it reads a newline */
    if (status == -1) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'write_to_fifo': " << "'write' failed, returning -1:" <<
            endl << strerror(errno) << endl << "Failed to write to FIFO." << endl <<
            "Exiting 'write_to_fifo' unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        return 1;
    }      /* if (status == -1) */
    else if (status == 0) {
        lock_cerr_mutex();
        cerr << thread_str << "ERROR! In 'write_to_fifo': " << "'write' failed, returning 0:" <<
            endl << strerror(errno) << endl << "Failed to write to FIFO." << endl <<
            "Exiting 'write_to_fifo' unsuccessfully with return value 1." << endl;
        unlock_cerr_mutex();
        return 1;
    }      /* if (status == 0) */
#endif DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << thread_str << "In 'write_to_fifo': " << "'write' succeeded, returning " <<
                status << "." << endl << "Wrote" << status <<
                " bytes plus one newline character to FIFO successfully." << endl;
            unlock_cerr_mutex();
        }      /* else if (DEBUG) */

```

```
#endif /* DEBUG_COMPILE */
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << thread_str << "Exiting 'write_to_fifo' successfully with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0;
} /* End of write_to_fifo definition */

```

This code is used in section 1786.

1786. Putting utility functions together. [LDF 2008.12.05.]

```
< Include files 3>
using namespace std;
using namespace gwrdifpk;
< Mutex function declarations 1718>
< submit_mysql_query declaration 1723>
< submit_mysql_queries declaration 1728>
< get_datestamp declaration 1731>
< convert_seconds declaration 1734>
< get_seconds_since_epoch declaration 1736>
< convert_time_spec declaration 1738>
< hexl_encode declaration 1749>
< hexl_decode declaration 1752>
< initialize_signal_maps declaration 1756>
< set_debug_level declaration 1758>
< check_irods_server declaration 1760>
< decrypt declaration 1769>
< set_password declaration 1781>
< write_to_fifo declaration 1784>
< submit_mysql_query definition 1724>
< submit_mysql_queries definition 1729>
< get_datestamp definition 1732>
< convert_seconds definition 1735>
< get_seconds_since_epoch definition 1737>
< convert_time_spec definition 1739>
< hexl_encode definition 1750>
< hexl_decode definition 1753>
< initialize_signal_maps definition 1757>
< set_debug_level definition 1759>
< check_irods_server definition 1761>
< decrypt definition 1770>
< set_password definition 1782>
< write_to_fifo definition 1785>
```

1787. This is what's written to the header file `utilfncts.h`. [LDF 2008.12.05.]

```
<utilfncts.h 1787> ≡
#ifndef UTILFNCS_H
#define UTILFNCS_H 1
    using namespace std;
    using namespace gwrdifpk;
    < Mutex function declarations 1718 >
    < submit_mysql_query declaration 1723 >
    < submit_mysql_queries declaration 1728 >
    < get_timestamp declaration 1731 >
    < convert_seconds declaration 1734 >
    < get_seconds_since_epoch declaration 1736 >
    < convert_time_spec declaration 1738 >
    < hexl_encode declaration 1749 >
    < hexl_decode declaration 1752 >
    < initialize_signal_maps declaration 1756 >
    < set_debug_level declaration 1758 >
    < check_irods_server declaration 1760 >
    < decrypt declaration 1769 >
    < set_password declaration 1781 >
    < write_to_fifo declaration 1784 >
#endif
```

1788. Generate TANs (gentans.web).

Log

[LDF 2012.07.13.] Created this file.

[LDF 2012.07.16.] Rewrote this program. It now calls `generate_tans`, which is defined in `utilfncts.web`.

This program, `gentan`, is meant to run as a daemon process in the background or as a cron job. If it is run as a daemon process, then a cron job should check whether it's still running and restart it, if necessary. It checks the number of unassigned TANs in the database table `gwiridsif.TANs` and generates new ones, up to a target amount (default 1000). It generates them x at a time, where the default for x is 100.

`gentans` calls the function `generate_tans` to generate new TANs. `generate_tans` checks the number of TANs in the database table `gwiridsif.TANs` and generates new ones, up to a target amount passed to the function as an argument. The target value can be passed to this program as the first (optional) command-line argument. Otherwise, a default of 1000 is used. If the target value is less than or equal to 0, a warning is issued and this program exits unsuccessfully with an exit status of 2.

The number of TANs written to the database per iteration can be set using the second (optional) command-line argument. If no argument is used, or the argument is 0, a default of 100 is used. If the value passed is negative or positive and greater than the target value, a warning is issued and `tans_per_iteration` is set to the minimum of the target value or 100.

A *sleep value* can be passed to `gentans` as the third (optional) command-line argument. If this value is > 0 , it will call `generate_tans` in a loop, sleeping for this number of seconds after each call to `generate_tans` exits. If no argument is provided a sleep value of 0 is assumed and `gentans` simply exits after calling `generate_tans` once.

!! PLEASE NOTE: Depending on how many TANs are already in the `gwiridsif.TANs` database table, the values of *target* and *tans_per_iteration*, and the number of rows actually inserted into the table in `generate_tans`, the table may contain more TANs than *target* when this program exits. This is because `generate_tans` will always try to insert *tans_per_iteration* rows, even when the number of rows in the table is already greater than *target - tans_per_iteration*. However, after a run of this program, the number of rows will never exceed *target + tans_per_iteration - 1*.

The IGNORE keyword is used with the SQL INSERT command, so that duplicates are not written to the table.

Used or expired TANs are removed from the table, so it is possible for a TAN to be identical to one that existed formerly. However, this shouldn't cause any problems. [LDF 2012.07.13.] [LDF 2012.07.16.] [LDF 2012.07.19.]

1789. Include files.

```
<Include files 3> +≡
#ifndef _XOPEN_SOURCE
#define _XOPEN_SOURCE
#endif
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <sys/mman.h>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <iostream>
#include <string>
#include <sstream>
#include <map>
#include <vector>
#if HAVE_CONFIG_H
#include <config.h>
#endif
#include <mysql.h>
#include "glblcnst.h++"
#include "glblvrbl.h++"
#include "excptntp.h++"
#include "utilfncs.h++"
#include "tanfncs.h++"
```

1790. Finish (exit handler). [LDF 2012.07.17.]

Log

[LDF 2012.07.17.] Added this function.

[LDF 2013.09.19.] Added code for unlocking the memory for *gpg-key-id* and *gpg-passphrase* and deleting them.

```

⟨finish definition 1790⟩ ≡
void finish(void)
{
    bool DEBUG = false;      /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Entering 'finish'." << endl;
        unlock_cerr_mutex();
    }
    if (gpg_key_id ≠ 0) {
        memset(gpg_key_id, 0, 10);
        munlock(gpg_key_id, 10);
        delete[] gpg_key_id;
        gpg_key_id = 0;
    }
    if (gpg_passphrase ≠ 0) {
        memset(gpg_passphrase, 0, gpg_passphrase_length);
        munlock(gpg_passphrase, gpg_passphrase_length);
        delete[] gpg_passphrase;
        gpg_passphrase = 0;
    }
    if (mysql_password ≠ 0) {
        memset(mysql_password, 0, MYSQL_PASSWORD_LENGTH);
        munlock(mysql_password, MYSQL_PASSWORD_LENGTH);
        delete[] mysql_password;
        mysql_password = 0;
    }
    pthread_mutex_destroy(&cerr_mutex);
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "Exiting 'finish'." << endl;
        unlock_cerr_mutex();
    }
    return;
}      /* End of finish definition */

```

See also section 1802.

This code is used in sections 1798 and 1835.

1791. Main definition. [LDF 2012.07.13.]

```
( main definition 1791 ) ≡
int main(int argc, char *argv[]){ int status;
    bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    MYSQL *mysql_ptr;
    pthread_mutex_init(&cerr_mutex, 0); /* Needed in generate_tans. Otherwise, not needed, since
        this program doesn't use threads. [LDF 2012.07.17.] */
    int sleep_val = 0; /* 0 = don't sleep, but rather exit. 43200 seconds = 12 hours */
    int target = 1000; /* Target number of TANs */
    int tans_per_iteration = 100; /* Number of TANs written to the gwirdsif.TANs database table
        per iteration. (See TEX text, above). */
    if (argc > 1) target = atoi(argv[1]);
    if (target ≤ 0) {
        cerr << "[gentans] WARNING! In 'main': ('target' == " << target << ") ≤ 0." <<
            endl << "Not generating TANs. Exiting 'gentans' unsuccessfully with "
            "exit_status 2." << endl;
        exit(2);
    } /* if (target ≤ 0) */
    if (argc > 2) tans_per_iteration = atoi(argv[2]);
    if (tans_per_iteration ≡ 0) {
        if (DEBUG) cerr << "[gentans] In 'main': 'tans_per_iteration' == 0." << endl <<
            "Setting it to default value of 100." << endl;
        tans_per_iteration = 100;
    }
    else if (tans_per_iteration < 0) {
        cerr << "[gentans] WARNING! In 'main': ('tans_per_iteration' == " <<
            tans_per_iteration << ") < 0." << endl << "Setting 'tans_per_iteration' to the minimum
            of 'target' == " << target << " and 100." << endl << "Continuing." << endl;
        tans_per_iteration = (target < 100) ? target : 100;
    } /* if (target ≤ 0) */
    if (tans_per_iteration > target) {
        cerr << "[gentans] WARNING! In 'main': 'tans_per_iteration' == " <<
            tans_per_iteration << " > 'target' == " << target << "." << endl <<
            "Setting 'tans_per_iteration' to the minimum of 'target' == " << target << " and 100." <<
            endl << "Continuing." << endl;
        tans_per_iteration = (target < 100) ? target : 100;
    } /* if (tans_per_iteration > target) */
    if (argc > 3) sleep_val = atoi(argv[3]);
    if (DEBUG)
        cerr << "[gentans] In 'main': sleep_val == " << sleep_val << endl << "'target' == " <<
            target << endl << "'tans_per_iteration' == " << tans_per_iteration << endl;
```

See also sections 1792, 1793, 1794, 1795, 1796, 1797, 1803, 1804, 1805, 1806, 1807, 1809, 1810, 1811, 1812, 1813, 1814, 1815, 1816, 1817, 1818, 1819, 1820, 1821, 1822, 1823, 1824, 1825, 1826, 1827, 1828, 1829, 1830, 1831, 1832, 1833, and 1834.

This code is used in sections 1798 and 1835.

1792. Initialize the MySQL library. [LDF 2012.07.13.]

```
< main definition 1791 > +≡
  if (mysql_library_init(0, Λ, Λ)) {
    cerr ≪ "ERROR! In 'main': Failed to initialize the MySQL library." ≪ endl ≪
      "Exiting 'gentans' unsuccessfully with exit status 1." ≪ endl;
    exit(1);
  }
  else if (DEBUG) {
    cerr ≪ "mysql_library_init succeeded." ≪ endl;
  }
```

1793.

```
< main definition 1791 > +≡
  mysql_ptr = mysql_init(0);
  if (mysql_ptr) {
    if (DEBUG) {
      cerr ≪ "In 'main': mysql_init succeeded." ≪ endl;
    } /* if (DEBUG) */
    /* if (mysql_ptr) */
  else {
    cerr ≪ "ERROR! In 'main':" ≪ endl ≪ "'mysql_init' failed. Exiting 'gentans'\
      unsuccessfully with " ≪ "exit status 1." ≪ endl;
    mysql_library_end();
    exit(1);
  }
```

1794. Connect to the ‘gwirdsif’ database. [LDF 2012.07.13.]

```
< main definition 1791 > +≡
  if (!mysql_real_connect(mysql_ptr, 0, "root", 0, 0, 0, "/var/run/mysql/mysql.sock", 0)) {
    cerr ≪ "ERROR! In 'main':" ≪ endl ≪ "'mysql_real_connect' failed.\
      Error: " ≪ mysql_error(mysql_ptr) ≪ endl ≪
      "Exiting 'gentans' unsuccessfully with exit status 1." ≪ endl;
    mysql_close(mysql_ptr);
    mysql_library_end();
    exit(1);
  }
  else if (DEBUG) {
    cerr ≪ "In 'main': mysql_real_connect succeeded." ≪ endl;
  }
```

1795. Select “gwirdsif” database. [LDF 2012.07.13.]

```
< main definition 1791 > +≡
status = mysql_select_db(mysql_ptr, "gwirdsif");
if (status == 0) {
    if (DEBUG) {
        cerr << "In 'main': " << "'mysql_select_db' succeeded." << endl;
    } /* if (DEBUG) */
} /* if (status == 0) */
else /* status ≠ 0 */
{
    cerr << "In 'main': " << "'mysql_select_db' failed, returning " << status << endl;
    mysql_close(mysql_ptr);
    mysql_library_end();
    exit(1);
} /* else (status ≠ 0) */
```

1796.

```
< main definition 1791 > +≡
for ( ; ; ) {
    status = generate_tans(mysql_ptr, 1000, 100);
    if (status != 0) {
        cerr << "ERROR! In 'main': 'generate_tans' failed, returning " << status << "."
            << endl << "Exiting 'gentans' unsuccessfully with exit status 1." << endl;
        mysql_close(mysql_ptr);
        mysql_library_end();
        exit(1);
    } /* if (status != 0) */
    else if (DEBUG) {
        cerr << "'generate_tans' succeeded." << endl;
    } /* else if (DEBUG) */
    if (sleep_val > 0) {
        if (DEBUG)
            cerr << "[gentans] 'sleep_val==' " << sleep_val << " > 0. "
                << "Going to sleep for "
            sleep_val << " seconds." << endl;
        sleep(sleep_val);
    }
    else {
        if (DEBUG) cerr << "[gentans] 'sleep_val==' " << sleep_val << " ≤ 0. "
            << "Not sleeping. Will exit."
        break;
    }
} /* for */
```

1797.

```
< main definition 1791 > +≡
mysql_close(mysql_ptr);
mysql_library_end();
exit(0); } /* End of main definition */
```

1798. Putting Generate TANs together. [LDF 2012.07.13.]

```
⟨ Include files 3 ⟩  
using namespace std;  
using namespace gwrdifpk;  
⟨ finish definition 1790 ⟩  
⟨ main definition 1791 ⟩
```

1799.

```
⟨ gentans.h 1799 ⟩ ≡      /* Empty */
```

1800. Generate PIDs (genpids.web).

Log

[LDF 2012.07.17.] Created this file.

1801. Include files.

```
<Include files 3> +≡
#ifndef _XOPEN_SOURCE
#define _XOPEN_SOURCE
#endif
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include <errno.h>
#include <ctype.h>
#include <limits.h>
#include <sys/types.h>
#include <pwd.h>
#include <sys/mman.h>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <iterator>
#include <string>
#include <sstream>
#include <deque>
#include <map>
#include <set>
#include <vector>
#if HAVE_CONFIG_H
#include <config.h>
#endif
#include <mysql.h>
#include "gblcnst.h++"
#include "gblvrb1.h++"
#include "excptntp.h++"
#include "utilfncs.h++"
#include "hndlvltp.h++"
#include "rspnstp.h++"
#include "irdsavtp.h++"
#include "irdsobtp.h++"
#include "hndltype.h++"
#include "pidfncs.h++"
```

1802. Finish (exit handler). [LDF 2012.07.17.]**Log**

[LDF 2012.07.17.] Added this function.

[LDF 2013.09.19.] Added code for unlocking the memory for *gpg-key-id* and *gpg-passphrase* and deleting them.

```
<finish definition 1790> +≡
void finish(void)
{
    bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
#if DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[genpids] " "Entering " "finish" . " << endl;
        unlock_cerr_mutex();
    }
#endif /* DEBUG_COMPILE */
    if (gpg-key-id ≠ 0) {
        memset(gpg-key-id, 0, 10);
        munlock(gpg-key-id, 10);
        delete[] gpg-key-id;
        gpg-key-id = 0;
    }
    if (gpg-passphrase ≠ 0) {
        memset(gpg-passphrase, 0, gpg-passphrase_length);
        munlock(gpg-passphrase, gpg-passphrase_length);
        delete[] gpg-passphrase;
        gpg-passphrase = 0;
    }
    if (mysql_password ≠ 0) {
        memset(mysql_password, 0, MYSQL_PASSWORD_LENGTH);
        munlock(mysql_password, MYSQL_PASSWORD_LENGTH);
        delete[] mysql_password;
        mysql_password = 0;
    }
    pthread_mutex_destroy(&cerr_mutex);
#if DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[genpids] " "Exiting " "finish" . " << endl;
        unlock_cerr_mutex();
    }
#endif /* DEBUG_COMPILE */
    return;
} /* End of finish definition */
```

1803. Main definition. [LDF 2012.07.13.]

```
/* **** (4) Arguments */
```

Command-line arguments:

```
1:      PID
2:      Prefix
3:      Institute
4:      Suffix
5:      Number of PIDs
6:      FIFO pathname
7:      User ID
8:      Username
9 10 ...: Type-value pairs (must be even number!)
⟨ main definition 1791 ⟩ +≡
int main(int argc, char *argv[]){ int status;
    bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    MYSQL *mysql_ptr = 0;
    stringstream sql_strm;
    MYSQL_RES *result = 0;
    MYSQL_ROW curr_row;
    unsigned int row_ctr = 0;
    unsigned int field_ctr = 0;
    string prefix_str;
    string institute_str;
    string suffix_str;
    pthread_mutex_init(&cerr_mutex, 0);
    string fifo_pathname;
    long int user_id = -1;
    string username;
    string pid_str;
    vector<Handle_Value_Triple> hvt_vector;
    Handle_Value_Triple hvt;
```

1804.

Log

[LDF 2013.02.22.] Added this section.

⟨ main definition 1791 ⟩ +≡

```
Handle_Value_Type::initialize_maps();
```

1805. Initialize the MySQL library. [LDF 2012.07.13.]

```

⟨ main definition 1791 ⟩ +≡
  if (mysql_library_init(0, Λ, Λ)) {
    lock_cerr_mutex();
    cerr << "[genpids] In 'main': ERROR! In 'main': Failed to initialize the MySQL library\
y." << endl << "Exiting 'genpids' unsuccessfully with exit status 1." << endl;
    unlock_cerr_mutex();
    exit(1);
  }
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "[genpids] In 'main': mysql_library_init succeeded." << endl;
      unlock_cerr_mutex();
    }
#endif /* DEBUG_COMPILE */

```

1806.

```

⟨ main definition 1791 ⟩ +≡
  mysql_ptr = mysql_init(0);
  if (mysql_ptr) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "[genpids] In 'main': 'mysql_init' succeeded." << endl;
      unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    /* if (mysql_ptr) */
  else {
    lock_cerr_mutex();
    cerr << "[genpids] ERROR! In 'main':" << endl <<
      "'mysql_init' failed. Exiting 'genpids' unsuccessfully with " <<
      "exit status 1." << endl;
    unlock_cerr_mutex();
    mysql_library_end();
    exit(1);
  }

```

1807. Connect to the database server. [LDF 2012.07.13.]

```
{ main definition 1791 } +≡
if (!mysql_real_connect(mysql_ptr, "root", 0, 0, 0, "/var/run/mysql/mysql.sock", 0)) {
    lock_cerr_mutex();
    cerr << "[genpids] ERROR! In 'main':"
    << endl <<
        "'mysql_real_connect' failed." << "Error:" << mysql_error(mysql_ptr) <<
        endl << "Exiting 'genpids' unsuccessfully with exit status 1." << endl;
    unlock_cerr_mutex();
    mysql_close(mysql_ptr);
    mysql_library_end();
    exit(1);
}
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[genpids] In 'main': mysql_real_connect succeeded."
        << endl;
        unlock_cerr_mutex();
    }
#endif /* DEBUG_COMPILE */
```

1808. Process command-line arguments. [LDF 2012.10.05.]

1809. Handle PID string. [LDF 2013.02.22.]

Log

[LDF 2013.02.22.] Added this section.

```
{ main definition 1791 } +≡
if (argc > 1 & strlen(argv[1]) > 0) pid_str = argv[1];
```

1810. Handle prefix (i.e., “naming authority”) string argument. [LDF 2012.09.28.]

Log

[LDF 2012.09.28.] Added this section.

```
{ main definition 1791 } +≡
if (argc > 2 & strlen(argv[2]) > 0) prefix_str = argv[2];
else {
    char *temp_str = 0;
    temp_str = getenv("HOSTNAME");
    if (temp_str == 0 || strncmp(temp_str, "pcfinston", strlen("pcfinston")) == 0) prefix_str = "12345";
    else prefix_str = "00001";
} /* else */
```

1811. Handle institute string argument. [LDF 2012.09.28.]

Log

[LDF 2012.09.28.] Added this section.

```

⟨ main definition 1791 ⟩ +≡
if (argc > 3 ∧ strlen(argv[3]) > 0) {
    if (strlen(argv[3]) ≠ 4) {
        lock_cerr_mutex();
        cerr ≪ "[genpids] ERROR! In 'main': Institute_string_argument_has"
            ≪ "invalid_length: " ≪ strlen(argv[3]) ≪ " characters (exactly 4 required). "
            ≪ endl ≪ "Exiting 'genpids' unsuccessfully with exit status 1." ≪ endl;
        unlock_cerr_mutex();
        mysql_close(mysql_ptr);
        mysql_library_end();
        exit(1);
    }
    else {
        for (int i = 0; i < strlen(argv[3]); ++i) institute_str += toupper(argv[3][i]);
    #if DEBUG_COMPILE
        if (DEBUG) {
            lock_cerr_mutex();
            cerr ≪ "[genpids] In 'main': institute_str == " ≪ institute_str ≪ ". "
            unlock_cerr_mutex();
        } /* if (DEBUG) */
    #endif /* DEBUG_COMPILE */
    } /* else */
} /* if (argc > 3) */

```

1812. Handle suffix argument. [LDF 2012.09.28.]

Log

[LDF 2012.09.28.] Added this section.

```

⟨ main definition 1791 ⟩ +≡
if (argc > 4 ∧ strlen(argv[4]) > 0) {
    if (strlen(argv[4]) > 24) {
        lock_cerr_mutex();
        cerr << "[genpids] ERROR! In 'main': Suffix_string_argument_too_long: "
            << strlen(argv[4]) << " characters (maximum 24)." << endl <<
            "Exiting 'genpids' unsuccessfully with exit status 1." << endl;
        unlock_cerr_mutex();
        mysql_close(mysql_ptr);
        mysql_library_end();
        exit(1);
    }
    for (int i = 0; i < strlen(argv[4]); ++i) {
        if (¬(isalnum(argv[4][i]) ∨ argv[4][i] ≡ '-')) {
            lock_cerr_mutex();
            cerr << "[genpids] ERROR! In 'main': Suffix_string_argument_contains_invalid "
                << "character: " << argv[4][i] << endl << "Only alphabetical characters and '-' are "
                << "allowed." << endl << "Exiting 'genpids' unsuccessfully with exit status 1." << endl;
            unlock_cerr_mutex();
            mysql_close(mysql_ptr);
            mysql_library_end();
            exit(1);
        }
        else if (islower(argv[4][i])) {
            suffix_str += toupper(argv[4][i]);
        }
        else suffix_str += argv[4][i];
    } /* for */
#endif DEBUG_COMPILE
if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[genpids] In 'main': 'suffix_str' == " << suffix_str << endl;
    unlock_cerr_mutex();
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (argc > 4) */
unsigned int number_of_pids = 1;
if (argc > 5) number_of_pids = atoi(argv[5]);

```

1813. Handle FIFO pathname argument. [LDF 2013.02.15.]

Log

[LDF 2013.02.15.] Added this section.

```

⟨ main definition 1791 ⟩ +≡
  if (argc > 6 ∧ strlen(argv[6]) > 0) {
    fifo_pathname = argv[6];
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "[genpids] In 'main': 'fifo_pathname' == " ≪ fifo_pathname ≪ endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (argc > 6) */

```

1814. Handle user ID argument. [LDF 2013.02.22.]

Log

[LDF 2013.02.22.] Added this section.

```

⟨ main definition 1791 ⟩ +≡
  if (argc > 7 ∧ strlen(argv[7]) > 0) { errno = 0;
  user_id = strtol(argv[7], 0, 10);
  if (user_id ≡ LONG_MAX ∨ user_id ≡ LONG_MIN) {
    lock_cerr_mutex();
    cerr ≪ "[genpids] ERROR! In 'main': strtoul failed, returning";
    if (user_id ≡ LONG_MAX) cerr ≪ "'LONG_MAX': " ≪ endl;
    else cerr ≪ "'LONG_MIN': " ≪ endl;
    cerr ≪ strerror(errno) ≪ endl ≪ "Failed to set 'user_id'. Exiting genp\
ids' unsuccessfully with " ≪ "exit_status 1." ≪ endl;
    unlock_cerr_mutex();
    mysql_close(mysql_ptr);
    mysql_library_end();
    exit(1);
} /* if (user_id ≡ LONG_MAX ∨ user_id ≡ LONG_MIN) */

```

1815.

```

⟨ main definition 1791 ⟩ +≡
  if (user_id > INT_MAX) {
    lock_cerr_mutex();
    cerr ≪ "[genpids] ERROR! In 'main': 'user_id' == " ≪ user_id ≪ " (> 'INT_MAX') " ≪
      endl ≪ "This is not permitted. Exiting 'genpids' unsuccessfully with " ≪
      "exit_status 1." ≪ endl;
    unlock_cerr_mutex();
    mysql_close(mysql_ptr);
    mysql_library_end();
    exit(1);
} /* if (user_id > INT_MAX) */

```

1816.

```
{ main definition 1791 } +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[genpids] In 'main': user_id" << user_id << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (argc > 7) */
```

1817. Handle username argument. [LDF 2013.02.22.]

Log

[LDF 2013.02.22.] Added this section.

```
{ main definition 1791 } +≡
    if (argc > 8 & strlen(argv[8]) > 0) {
        username = argv[8];
#endif DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[genpids] In 'main': username" << username << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (argc > 8) */
```

1818. If *username.empty()*, get system username. [LDF 2013.02.27.]

Log

[LDF 2013.02.27.] Added this section.

```

⟨ main definition 1791 ⟩ +≡
  if (username.empty()) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[genpids] In 'main': 'username' is empty. Will call 'getpwuid_r'" <<
      "to get username." << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  struct passwd pwd;
  struct passwd *res;
  char *buf;
  size_t bufsize;
  int s;
  bufsize = sysconf(_SC_GETPW_R_SIZE_MAX);
  if (bufsize == -1) /* Value was indeterminate */
    bufsize = 16384; /* Should be more than enough */
  errno = 0;
  buf = static_cast<char*>(malloc(bufsize));
  if (buf == 0) {
    lock_cerr_mutex();
    cerr << "[genpids] ERROR! In 'main': malloc failed:" << endl << strerror(errno) <<
      endl << "Failed to set 'username'. Exiting 'genpids' unsuccessfully with" <<
      "exit status 1." << endl;
    unlock_cerr_mutex();
    mysql_close(mysql_ptr);
    mysql_library_end();
    exit(1);
  } /* if (buf == 0) */
  errno = 0;
  s = getpwuid_r(getuid(), &pwd, buf, bufsize, &res);
  if (res == 0) {
    if (s == 0) {
      lock_cerr_mutex();
      cerr << "[genpids] ERROR! In 'main': 'getpwuid_r' set 'res' to 0." << endl <<
        "Failed to set 'username'. Exiting 'genpids' unsuccessfully with" <<
        "exit status 1." << endl;
      unlock_cerr_mutex();
    }
  } else {
    lock_cerr_mutex();
    cerr << "[genpids] ERROR! In 'main': 'getpwuid_r' failed, returning" << s << ":" <<
      endl << strerror(errno) << endl << "Exiting 'genpids' unsuccessfully with" <<
      "exit status 1." << endl;
    unlock_cerr_mutex();
  }
}

```

```

    }
    mysql_close(mysql_ptr);
    mysql_library_end();
    exit(1);
} /* if (res == 0) */
if (pwd.pw_name == 0 || strlen(pwd.pw_name) == 0) {
    cerr << "[genpids] ERROR! In 'main': 'pwd.pw_name' is NULL or empty." << endl <<
        "Failed to set 'username'. " << "Exiting 'genpids' unsuccessfully with " <<
        "exit status 1." << endl;
unlock_cerr_mutex();
mysql_close(mysql_ptr);
mysql_library_end();
exit(1);
} /* if (pwd.pw_name == 0 || strlen(pwd.pw_name)) */

```

1819.

```

⟨ main definition 1791 ⟩ +≡
    username = pwd.pw_name;
    free(buf);
    buf = 0;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "'username' == " << username << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (username.empty()) */

```

1820. If *user_id* ≤ 0, look up *user_id* in ‘gwirdsif.Users’ database table. [LDF 2013.02.27.]

Log

[LDF 2013.02.27.] Added this section.

```

⟨ main definition 1791 ⟩ +≡
    if (user_id ≤ 0) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[genpids] In 'main': 'user_id' == " << user_id << "(≤ 0)" << endl <<
            "Will look up 'user_id' for 'username' == " << username << "." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1821.

```

⟨ main definition 1791 ⟩ +≡
  sql_strm << "select user_id from gwiridsif.Users where username='"
  << username << "'";
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "sql_strm.str() == "
    << sql_strm.str() << endl;
    unlock_cerr_mutex();
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  status = submit_mysql_query(sql_strm.str(), result, mysql_ptr, &row_ctr, &field_ctr);
  if (status != 0) {
    cerr << "[genpids] ERROR! In 'main': 'submit_mysql_query' failed, returning "
    << status << "."
    << endl << "Failed to retrieve 'user_id' for 'username' == "
    << username << "'"
    << " from 'gwiridsif.Users' database table."
    << endl <<
    "Exiting 'genpids' unsuccessfully with "
    << "exit status 1."
    << endl;
    unlock_cerr_mutex();
    if (result) {
      mysql_free_result(result);
    }
    mysql_close(mysql_ptr);
    mysql_library_end();
    exit(1);
  } /* if (status != 0) */
#endif DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr << "[genpids] In 'main': 'submit_mysql_query' succeeded, returning 0."
      << endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1822.

```

⟨ main definition 1791 ⟩ +≡
  if (row_ctr ≡ 0) {
    cerr << "[genpids] ERROR! In 'main': submit_mysql_query' returned 0 rows." <<
      endl << "Failed to retrieve 'user_id' for 'username' == '" <<
      username << "' from 'gwirdsif.Users' database table." << endl <<
      "Exiting 'genpids' unsuccessfully with " << exit_status_1." << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    mysql_close(mysql_ptr);
    mysql_library_end();
    exit(1);
  } /* if (row_ctr ≡ 0) */
#endif /* DEBUG_COMPILE
else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[genpids] In 'main': 'submit_mysql_query' returned " << row_ctr << " rows." <<
      endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1823.

```

⟨ main definition 1791 ⟩ +≡
  curr_row = mysql_fetch_row(result);
  if (curr_row ≡ 0) {
    lock_cerr_mutex();
    cerr << "[genpids] ERROR! In 'main':" << endl << "'mysql_fetch_row' failed,
      returning NULL:" << endl << "MySQL error: " << mysql_error(mysql_ptr) << endl <<
      "Exiting 'genpids' unsuccessfully with exit status 1." << endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    mysql_close(mysql_ptr);
    mysql_library_end();
    exit(1);
  } /* if (curr_row ≡ 0) */

```

1824.

```

⟨ main definition 1791 ⟩ +≡
#endif DEBUG_COMPILE
else
  if (DEBUG) {
    lock_cerr_mutex();
    cerr << "[genpids] In 'main':" << endl << "'mysql_fetch_row' succeeded." << endl <<
      "'curr_row[0]' == " << curr_row[0] << endl;
    unlock_cerr_mutex();
  } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1825.

```

⟨ main definition 1791 ⟩ +≡
  errno = 0;
  user_id = strtol(curr_row[0], 0, 10);
  if (user_id ≡ LONG_MIN ∨ user_id ≡ LONG_MAX) {
    lock_cerr_mutex();
    cerr ≪ "[genpids] ERROR! In 'main': " ≪ endl ≪ "'strtol' failed, returning";
    if (user_id ≡ LONG_MIN) cerr ≪ "'LONG_MIN': " ≪ endl;
    else cerr ≪ "'LONG_MAX': " ≪ endl;
    cerr ≪ strerror(errno) ≪ endl ≪ "Exiting 'genpids' unsuccessfully with exit status 1." ≪
      endl;
    unlock_cerr_mutex();
    mysql_free_result(result);
    mysql_close(mysql_ptr);
    mysql_library_end();
    exit(1);
  } /* if (user_id ≡ LONG_MIN ∨ user_id ≡ LONG_MAX) */
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr ≪ "[genpids] In 'main': " ≪ endl ≪ "'strtol' succeeded. 'user_id' == "
        user_id ≪ ". " ≪ endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  mysql_free_result(result);
  result = 0; } /* if (user_id ≤ 0) */

```

1826.

```

⟨ main definition 1791 ⟩ +≡
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      lock_cerr_mutex();
      cerr ≪ "[genpids] In 'main': 'user_id' == " ≪ user_id ≪ " (> 0)" ≪ endl ≪
        "Not looking up 'user_id' for 'username' == " ≪ username ≪ ". " ≪ endl;
      unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1827.

```

⟨ main definition 1791 ⟩ +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[genpids] In 'main': " << endl << "Variables set by means of command-line \
            arguments or otherwise: " << endl << "pid_str==" << pid_str << endl <<
            "prefix_str==" << prefix_str << endl << "institute_str==" << institute_str << endl <<
            "suffix_str==" << suffix_str << endl << "number_of_pids==" << number_of_pids << endl <<
            "fifo.pathname==" << fifo.pathname << endl << "user_id==" << user_id << endl <<
            "username==" << username << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1828. Handle additional type-value pairs.

Log

[LDF 2013.02.22.] Added this section.

```

⟨ main definition 1791 ⟩ +≡
    if (argc > 9) { int j;
        unsigned int curr_type_ctr = 0;
        unsigned int other_type_ctr = Handle_Value_Type::OTHER_HANDLE_VALUE_TYPE_INDEX;
        map<string, unsigned int>::const_iterator iter; for (int i = 9; i < argc; i += 2) { j = i + 1;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[genpids] In 'main': " << endl << "'argv[" << i << "]' == " << argv[i] << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    if (argc ≤ j) {
        cerr << "[genpids] WARNING! In 'genpids': " << endl <<
            "No value supplied for 'type'" << argv[i] << "." << endl <<
            "Can't pass type/value pair to 'generate_pids'." << endl << "Continuing." << endl;
        break;
    } /* if (argc ≤ j) */
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[genpids] In 'genpids': " << endl << "'type'" == " << argv[i] << ',' << endl <<
            "'value'" == " << argv[j] << ',' << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1829.

```

⟨ main definition 1791 ⟩ +≡
    iter = Handle_Value_Type::type_idx_map.find(string(argv[i]));
    if (iter != Handle_Value_Type::type_idx_map.end()) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[genpids] In 'genpids': " << endl << "'type'" <= " " << argv[i] << "' found in " <<
            "'Handle_Value_Type::type_idx_map'." << endl << "'idx'" <= " " << iter->second << "(" <<
            Handle_Value_Type::idx_type_map[iter->second] << ")" << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    hvt.idx = iter->second;
} /* if (iter != Handle_Value_Type::type_idx_map.end()) */
else {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[genpids] In 'genpids': " << endl << "'type'" <= " " << argv[i] <<
            "' " << "not found in 'Handle_Value_Type::type_idx_map'." << endl <<
            "'Setting 'hvt.idx' to " << other_type_ctr << "." << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    hvt.idx = other_type_ctr++;
} /* else */
hvt.type = argv[i];
hvt.data_str = argv[j];
hvt_vector.push_back(hvt); } /* for */
} /* if (argc > 9) */

```

1830.

```
< main definition 1791 > +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        if (hvt_vector.size() > 0) {
            lock_cerr_mutex();
            cerr << "[genpids] In 'genpids': " << endl << "'hvt_vector.size()' == " <<
                hvt_vector.size() << "(>0)" << endl << "'hvt_vector': " << endl;
            for (vector<Handle_Value_Triple>::const_iterator iter = hvt_vector.begin();
                    iter != hvt_vector.end(); ++iter) {
                cerr << "idx:" << setw(3) << iter->idx << " | " << "type:" << setw(16) << iter->type <<
                    " | data_str:" << iter->data_str << endl;
            }
            cerr << endl;
            unlock_cerr_mutex();
        } /* if (hvt_vector.size() > 0) */
    } else {
        lock_cerr_mutex();
        cerr << "[genpids] In 'genpids': " << endl << "'hvt_vector.size()' == 0." << endl;
        unlock_cerr_mutex();
    }
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

1831. Generate PIDs. [LDF 2012.09.28.]

Log

[LDF 2012.09.28.] Added this section.
 [LDF 2013.01.28.] Now writing *pid_str* to *cout*.
 [LDF 2013.02.15.] Now passing **string** *fifo.pathname* to *generate_pids* as an argument.
 [LDF 2013.02.22.] Now passing **long int** *user_id* and **string** *username* to *generate_pids* as arguments.

```

⟨ main definition 1791 ⟩ +≡
vector<string> pid_vector;
vector<Handle_Type> handle_vector;
bool standalone_hs = true;
/* false true for use with standalone handle server, otherwise false. [LDF 2012.09.28.] */
status = generate_pids(mysql_ptr, prefix_str, pid_str, &pid_vector, number_of_pids, 0, 0, standalone_hs,
                      institute_str, suffix_str, &handle_vector, fifo_pathname, user_id, username);
if (status ≠ 0) {
    lock_cerr_mutex();
    cerr ≪ "[genpids] ↴ERROR! ↴In ↴‘main’: ↴‘generate_pids’ ↴failed, ↴returning ↴" ≪ status ≪
        ". " ≪ endl ≪ "Exiting ↴‘genpids’ ↴unsuccessfully ↴with ↴exit ↴status ↴1." ≪ endl;
    unlock_cerr_mutex();
    mysql_close(mysql_ptr);
    mysql_library_end();
    exit(1);
} /* if (status ≠ 0) */
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        lock_cerr_mutex();
        cerr ≪ "[genpids] ↴In ↴‘main’: ↴‘generate_pids’ ↴succeeded, ↴returning ↴0." ≪ endl ≪
            "'pid_str' ↴== ↴" ≪ pid_str ≪ ". " ≪ endl ≪ "'pid_vector.size()' ↴== ↴" ≪
            pid_vector.size() ≪ endl;
        for (int i = 0; i < pid_vector.size(); ++i) {
            cerr ≪ "pid_vector[" ≪ i ≪ "]" ≪ pid_vector[i] ≪ endl;
        }
        cerr ≪ endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
if (pid_vector.size() > 0) {
    lock_cout_mutex();
    lock_cerr_mutex();
    for (int i = 0; i < pid_vector.size(); ++i) {
        cout ≪ pid_vector[i] ≪ endl;
    }
    unlock_cerr_mutex();
    unlock_cout_mutex();
}
else {
    lock_cout_mutex();
    lock_cerr_mutex();
    cout ≪ pid_str ≪ endl;
    unlock_cerr_mutex();
}

```

```
    unlock_cout_mutex( );
}
```

1832.

Log

[LDF 2013.02.22.] Added this section.

```

⟨ main definition 1791 ⟩ +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[genpids] In 'main':" << endl << "'handle_vector.size()' == " <<
            handle_vector.size() << endl;
        for (int i = 0; i < handle_vector.size(); ++i) {
            cerr << "'handle_vector['" << i << "]':" << endl;
            handle_vector[i].show();
        }
        cerr << endl;
        cerr << "'hvt_vector.size()' == " << hvt_vector.size() << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    if (handle_vector.size() > 0 & hvt_vector.size() > 0) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[genpids] In 'main':" << endl << "'handle_vector.size()' == " <<
            handle_vector.size() << "(>0) and " << endl <<
            "'hvt_vector.size()' == " << hvt_vector.size() << "(>0)" << endl <<
            "Will call 'Handle_Type::add_values' on the members of " << "'handle_vector'" <<
            endl << "'user_id' == " << user_id << endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        for (vector<Handle_Type>::iterator iter = handle_vector.begin(); iter != handle_vector.end();
            ++iter) {
            status = iter->add_values(mysql_ptr, hvt_vector, user_id);
            if (status != 0) {
                lock_cerr_mutex();
                cerr << "[genpids] ERROR! In 'main':" << endl <<
                    "'Handle_Type::add_values' failed, returning " << status << "." <<
                    endl << "Failed to add handle values to handle " << iter->handle << "." << endl <<
                    "Exiting 'genpids' unsuccessfully with exit status 1." << endl;
                unlock_cerr_mutex();
                mysql_close(mysql_ptr);
                mysql_library_end();
                exit(1);
            } /* if (status != 0) */
#endif /* DEBUG_COMPILE */
        else
            if (DEBUG) {
                lock_cerr_mutex();

```

```

    cerr << "[genpids] In 'main':" << endl <<
        "'Handle_Type::add_values' succeeded, returning 0." << endl <<
        "Added handle values to handle " << iter->handle << "' successfully." << endl;
        unlock_cerr_mutex();
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* for */
} /* if (handle_vector.size() > 0 & hvt_vector.size() > 0) */

```

1833.

```

⟨ main definition 1791 ⟩ +≡
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            lock_cerr_mutex();
            cerr << "[genpids] In 'main':" << endl << "'hvt_vector.size()' == " << hvt_vector.size() <<
                endl << "'hvt_vector.size()' == " << hvt_vector.size() << "." <<
                endl << "Either 'handle_vector' or 'hvt_vector' or both are empty." <<
                endl << "Not calling 'Handle_Type::add_values' on the "
                    "members (if any) of 'handle_vector'." << endl;
            unlock_cerr_mutex();
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

1834.

```

⟨ main definition 1791 ⟩ +≡
    mysql_close(mysql_ptr);
    mysql_library_end();
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        lock_cerr_mutex();
        cerr << "[genpids] In 'main': Exiting 'genpids' successfully with exit status 0." <<
            endl;
        unlock_cerr_mutex();
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    exit(0); } /* End of main definition */

```

1835. Putting Generate PIDs together. [LDF 2012.07.13.]

```

⟨ Include files 3 ⟩
using namespace std;
using namespace gwrdifpk;
⟨ finish definition 1790 ⟩
⟨ main definition 1791 ⟩

```

1836. Index.

bin: [1387](#).
_GNU_SOURCE: [31](#), [1458](#), [1675](#).
_MSC_VER: 1262.
_SC_GETPW_R_SIZE_MAX: 1818.

_XOPEN_SOURCE: [3](#), [308](#), [701](#), [751](#), [886](#), [995](#), [1037](#),
[1337](#), [1369](#), [1413](#), [1579](#), [1660](#), [1715](#), [1789](#), [1801](#).
aadmin_read: [789](#), [790](#), 791.
aadmin_write: [789](#), [790](#), 791.

accept: 1470, 1483, 1501.
access: 41, 157, 234, 239, 241, 1613, 1696, 1705, 1782.
activation_time: 1389, 1395.
add_avu: 474, 1140, 1141, 1148, 1151.
add_avu_cond: 1149, 1150, 1152, 1203.
add_handle_value: 887, 1191, 1210, 1221, 1223, 1224, 1228.
ADD_HANDLE_VALUE_TYPE: 9, 10, 57, 58.
add_metadata: 253, 289, 340, 384, 416, 1634.
add_value: 181, 182, 401, 403, 404, 405, 406, 463, 657, 899, 900, 932, 1169, 1170, 1223, 1228.
add_values: 171, 463, 465, 466, 900, 901, 902, 925, 1832.
ADD_VALUES_UNLOCK_TABLES: 905, 906, 907, 909, 910, 912, 913, 914, 916, 922, 924.
addrinfo: 1416.
admin_data: 791, 1607.
admin_data_length: 791, 1607.
admin_read: 588, 752, 759, 776, 791, 870, 872, 876.
admin_write: 588, 752, 759, 777, 791, 870, 872, 876.
AF_INET: 1422, 1482, 1483, 1500, 1501.
AF_UNIX: 1464.
AF_UNSPEC: 1416.
affected_rows: 107, 108, 116, 122, 127, 151, 798, 819, 820, 832, 846, 847, 874, 877, 878, 902, 922, 927, 937, 938, 966, 971, 972, 982, 987, 988, 1010, 1021, 1027, 1030, 1064, 1069, 1071, 1073, 1078, 1122, 1127, 1129, 1154, 1159, 1160, 1174, 1182, 1194, 1206, 1209, 1211, 1212, 1213, 1298, 1306, 1307, 1308, 1309, 1314, 1316, 1582, 1603, 1609, 1610, 1662, 1670, 1723, 1724, 1725.
affected_rows_vector: 109, 110, 111, 1232, 1240, 1242, 1257, 1728, 1729, 1730.
affected_rows_1: 1154, 1159, 1160.
AFTER_TESTS: 1288, 1289, 1290.
ai_addr: 1416, 1418.
ai_addrlen: 1416.
ai_family: 1416, 1418.
ai_flags: 1416.
ai_next: 1416.
AI_PASSIVE: 1416.
ai_protocol: 1416.
ai_socktype: 1416.
algo: 1389, 1395.
and_str: 1161.
ANON_AUTH_TYPE: 35, 36, 1483.
anoncred: 1482, 1489, 1494.
anonymous: 33, 39, 1508.
append: 825, 852.
append_newline: 1784, 1785.

archive: 550, 556, 1231, 1232, 1233, 1236, 1245, 1248.
arg: 1454, 1455.
argc: 752, 887, 1791, 1803, 1809, 1810, 1811, 1812, 1813, 1814, 1816, 1817, 1828, 1829.
args: 134, 144, 145.
argv: 752, 887, 1791, 1803, 1809, 1810, 1811, 1812, 1813, 1814, 1817, 1828, 1829.
ASCII control characters: 610.
assign: 787, 794.
atoi: 77, 85, 87, 98, 120, 767, 768, 769, 771, 772, 773, 775, 776, 777, 778, 779, 1623, 1666, 1791, 1812.
attr: 1286, 1287, 1290, 1464, 1473, 1483, 1486, 1500, 1508, 1509.
attrib: 999, 1000, 1005, 1006, 1229, 1230.
attribute: 381, 996, 1000, 1004, 1006, 1008, 1013, 1024, 1030, 1052, 1072, 1113, 1136, 1144, 1151, 1230.
attribute_map: 367, 1280, 1290, 1307, 1319, 1325, 1329, 1331.
attributes_found: 1298, 1307.
authentication_error: 1517, 1524, 1526.
avu: 474, 476, 1047, 1049, 1052, 1053, 1054, 1055, 1140, 1141, 1144, 1146, 1147.
avu_id: 1068.
avu_iter: 1174.
avu_vector: 186, 191, 331, 380, 381, 408, 467, 468, 1038, 1044, 1055, 1056, 1059, 1061, 1072, 1073, 1117, 1136, 1140, 1146, 1151, 1230, 1247.
avu_1: 474, 476.
b: 512, 1576, 1655.
back: 170, 171, 181, 182, 397, 398, 410, 411, 412, 413, 462, 463, 814, 815, 900, 921, 1707, 1709, 1710.
basename: 210, 444, 445.
begin: 25, 52, 106, 136, 138, 169, 181, 244, 274, 331, 337, 350, 368, 371, 374, 380, 381, 395, 396, 400, 401, 404, 425, 426, 463, 482, 535, 537, 539, 552, 574, 645, 646, 668, 763, 816, 821, 844, 848, 894, 898, 916, 917, 918, 919, 945, 949, 953, 970, 1056, 1061, 1070, 1072, 1117, 1128, 1136, 1158, 1161, 1164, 1168, 1188, 1189, 1190, 1201, 1203, 1209, 1210, 1220, 1230, 1236, 1247, 1248, 1257, 1273, 1275, 1277, 1280, 1307, 1325, 1331, 1449, 1450, 1528, 1615, 1704, 1730, 1830, 1832.
bin: 1386, 1387.
bin_size: 1386, 1387.
binary: 249, 250, 1520.
binary_data: 821, 823, 825, 848, 850, 852.
bind: 1465, 1482, 1500.
bin2hex: 1386, 1387, 1395, 1434, 1435.

bits: [1389](#), [1395](#).

bitset: [91](#), [498](#), [512](#), [1639](#), [1641](#), [1655](#).

blank_str: [379](#), [380](#), [381](#).

brief: [318](#), [319](#).

buf: [1818](#), [1819](#).

Buff: [1263](#).

buff_ptr: [233](#).

buff_size: [135](#), [136](#), [137](#), [141](#), [142](#), [144](#), [145](#), [148](#), [200](#), [201](#), [202](#), [255](#), [256](#), [257](#), [258](#), [259](#).

buffer: [95](#), [98](#), [99](#), [126](#), [137](#), [139](#), [141](#), [142](#), [145](#), [146](#), [148](#), [149](#), [158](#), [159](#), [160](#), [161](#), [211](#), [212](#), [213](#), [214](#), [223](#), [227](#), [228](#), [230](#), [233](#), [251](#), [263](#), [266](#), [267](#), [279](#), [282](#), [283](#), [296](#), [389](#), [425](#), [426](#), [444](#), [456](#), [478](#), [479](#), [480](#), [481](#), [485](#), [486](#), [487](#), [520](#), [526](#), [527](#), [565](#), [566](#), [568](#), [569](#), [573](#), [574](#), [589](#), [590](#), [592](#), [595](#), [598](#), [599](#), [601](#), [604](#), [607](#), [612](#), [614](#), [616](#), [618](#), [619](#), [620](#), [621](#), [624](#), [625](#), [626](#), [627](#), [631](#), [632](#), [634](#), [635](#), [637](#), [641](#), [644](#), [647](#), [649](#), [652](#), [653](#), [655](#), [656](#), [657](#), [660](#), [662](#), [663](#), [664](#), [665](#), [667](#), [668](#), [670](#), [671](#), [672](#), [673](#), [675](#), [677](#), [678](#), [679](#), [680](#), [681](#), [682](#), [684](#), [687](#), [689](#), [692](#), [694](#), [697](#), [792](#), [793](#), [1010](#), [1015](#), [1016](#), [1017](#), [1018](#), [1024](#), [1047](#), [1049](#), [1050](#), [1051](#), [1052](#), [1053](#), [1054](#), [1055](#), [1133](#), [1141](#), [1163](#), [1174](#), [1178](#), [1179](#), [1181](#), [1249](#), [1251](#), [1252](#), [1254](#), [1282](#), [1283](#), [1284](#), [1446](#), [1499](#), [1502](#), [1517](#), [1518](#), [1519](#), [1520](#), [1522](#), [1523](#), [1525](#), [1526](#), [1530](#), [1531](#), [1539](#), [1549](#), [1551](#), [1552](#), [1558](#), [1560](#), [1562](#), [1563](#), [1566](#), [1567](#), [1662](#), [1668](#), [1679](#), [1681](#), [1682](#), [1685](#), [1697](#), [1703](#), [1737](#), [1749](#), [1750](#), [1752](#), [1762](#), [1764](#), [1765](#), [1766](#), [1771](#), [1772](#), [1773](#), [1775](#), [1776](#), [1777](#), [1778](#), [1779](#), [1780](#), [1784](#), [1785](#).

buffer_contents_length: [1771](#), [1776](#), [1779](#).

buffer_length: [1784](#), [1785](#).

buffer_ptr: [135](#), [136](#), [138](#), [139](#), [140](#), [141](#), [142](#), [144](#), [145](#), [146](#), [147](#), [148](#), [149](#), [200](#), [201](#), [202](#), [207](#), [210](#), [214](#), [217](#), [218](#), [219](#), [220](#), [221](#), [223](#), [224](#), [225](#), [226](#), [253](#), [255](#), [256](#), [257](#), [259](#), [261](#), [264](#), [265](#), [268](#), [269](#), [271](#), [273](#), [274](#), [275](#), [277](#), [280](#), [281](#), [284](#), [285](#), [286](#), [299](#), [340](#), [341](#), [344](#), [345](#), [346](#), [347](#), [348](#), [350](#), [351](#), [352](#), [354](#), [355](#), [357](#), [360](#), [363](#), [364](#), [369](#), [370](#), [371](#), [372](#), [378](#), [381](#), [395](#), [530](#), [531](#), [532](#), [535](#), [565](#), [566](#), [568](#), [569](#), [573](#), [574](#), [589](#), [590](#), [592](#), [593](#), [595](#), [596](#), [598](#), [599](#), [601](#), [602](#), [604](#), [605](#), [607](#), [608](#), [609](#), [610](#), [611](#), [612](#), [614](#), [616](#), [618](#), [621](#), [624](#), [627](#), [631](#), [632](#), [634](#), [635](#), [637](#), [641](#), [644](#), [647](#), [649](#), [652](#), [653](#), [655](#), [657](#), [660](#), [662](#), [663](#), [664](#), [665](#), [667](#), [668](#), [670](#), [671](#), [672](#), [673](#), [675](#), [677](#), [678](#), [679](#), [680](#), [682](#), [684](#), [687](#), [689](#), [692](#), [694](#), [697](#), [796](#), [830](#), [1517](#), [1539](#).

BUFFER_SIZE: [33](#), [39](#), [137](#), [142](#), [223](#), [224](#), [225](#), [226](#), [227](#), [230](#), [232](#), [233](#), [251](#), [273](#), [520](#), [526](#), [528](#), [565](#), [568](#), [569](#), [573](#), [574](#), [589](#), [592](#), [593](#), [595](#), [596](#), [598](#), [599](#), [601](#), [602](#), [604](#), [605](#), [607](#), [608](#), [614](#), [618](#),

buffer_size: [253](#), [273](#), [274](#), [275](#), [277](#), [280](#), [281](#), [284](#), [285](#), [286](#), [296](#), [299](#), [340](#), [341](#), [344](#), [345](#), [346](#), [347](#), [348](#), [350](#), [351](#), [352](#), [354](#), [355](#), [357](#), [360](#), [363](#), [369](#), [370](#), [371](#), [372](#), [378](#), [381](#), [395](#), [425](#), [478](#), [481](#), [484](#), [485](#), [486](#), [487](#), [530](#), [796](#), [830](#), [1749](#), [1750](#), [1752](#), [1770](#), [1771](#), [1775](#), [1776](#), [1777](#), [1778](#), [1779](#), [1780](#).

BUFSIZE: [1262](#).

bufsize: [1818](#).

BUG FIX: [38](#), [183](#), [188](#), [190](#), [225](#), [232](#), [251](#), [255](#), [336](#), [342](#), [407](#), [474](#), [565](#), [639](#), [646](#), [764](#), [791](#), [869](#), [923](#), [935](#), [1015](#), [1083](#), [1165](#), [1177](#), [1240](#), [1345](#), [1415](#), [1417](#), [1501](#), [1518](#), [1520](#), [1522](#), [1530](#), [1531](#), [1552](#), [1554](#), [1565](#), [1567](#), [1598](#), [1599](#), [1604](#), [1608](#), [1621](#), [1723](#), [1772](#).

byte_ctr_vector: [423](#), [425](#).

c: [1750](#).

c_str: [46](#), [47](#), [52](#), [54](#), [97](#), [101](#), [126](#), [138](#), [142](#), [146](#), [149](#), [157](#), [158](#), [173](#), [184](#), [189](#), [207](#), [210](#), [211](#), [214](#), [227](#), [233](#), [239](#), [241](#), [245](#), [249](#), [260](#), [261](#), [264](#), [265](#), [267](#), [268](#), [269](#), [271](#), [275](#), [276](#), [277](#), [280](#), [281](#), [283](#), [284](#), [285](#), [286](#), [328](#), [344](#), [345](#), [346](#), [347](#), [348](#), [350](#), [351](#), [352](#), [354](#), [355](#), [357](#), [360](#), [363](#), [364](#), [369](#), [370](#), [371](#), [372](#), [378](#), [381](#), [408](#), [419](#), [424](#), [434](#), [444](#), [445](#), [452](#), [486](#), [487](#), [515](#), [527](#), [531](#), [532](#), [535](#), [566](#), [569](#), [599](#), [610](#), [616](#), [625](#), [626](#), [650](#), [653](#), [656](#), [657](#), [662](#), [663](#), [664](#), [665](#), [667](#), [668](#), [670](#), [671](#), [672](#), [677](#), [678](#), [679](#), [680](#), [681](#), [865](#), [903](#), [906](#), [909](#), [913](#), [922](#), [923](#), [953](#), [1014](#), [1024](#), [1048](#), [1055](#), [1069](#), [1071](#), [1073](#), [1127](#), [1129](#), [1134](#), [1137](#), [1144](#), [1162](#), [1178](#), [1226](#), [1248](#), [1416](#), [1463](#), [1464](#), [1466](#), [1500](#), [1549](#), [1552](#), [1553](#), [1559](#), [1560](#), [1561](#), [1562](#), [1563](#), [1564](#), [1565](#), [1566](#), [1613](#), [1614](#), [1615](#), [1616](#), [1668](#), [1682](#), [1696](#), [1699](#), [1705](#), [1708](#), [1724](#), [1735](#), [1737](#), [1743](#), [1754](#), [1763](#), [1766](#), [1775](#), [1777](#), [1782](#).

ca_cert: [34](#).

ca_list: [1500](#).

ca_list_size: [1500](#).

CACHE: [1371](#), [1372](#), [1374](#).

cache_db: [1372](#), [1374](#), [1376](#), [1378](#), [1380](#), [1382](#).

cache_db_ptr: [1372](#), [1378](#).

call_imeta: [1140](#), [1141](#), [1143](#), [1151](#).

calloc: [1374](#).

ccertificate_id: [709](#), [710](#), [711](#), [712](#).

ccommonName: [709](#), [710](#), [711](#), [712](#), [1343](#), [1344](#).

ccountryName: [709](#), [710](#), [711](#), [712](#), [1343](#), [1344](#).

ccreated: [789](#), [790](#), [791](#).

ccreated_by_user_id: 789, 790, 791.
ccreated_str: 789, 790, 791.
cd: 255, 256, 272, 599.
CD_TYPE: 9, 10, 57, 58.
cerr: 25, 39, 40, 41, 45, 46, 47, 49, 51, 52, 54, 55, 62, 63, 64, 65, 66, 67, 68, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 106, 110, 111, 112, 113, 115, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 136, 138, 139, 140, 141, 142, 145, 146, 147, 148, 149, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 193, 194, 195, 196, 197, 199, 201, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 235, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 256, 257, 258, 259, 260, 261, 262, 264, 265, 266, 267, 268, 269, 270, 271, 272, 274, 275, 276, 277, 278, 280, 281, 282, 283, 284, 285, 287, 288, 319, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 339, 341, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 381, 382, 384, 386, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 400, 401, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 429, 430, 431, 433, 434, 435, 436, 437, 438, 439, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 460, 461, 462, 463, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 484, 485, 486, 487, 488, 490, 491, 492, 493, 494, 495, 496, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 526, 527, 528, 530, 531, 532, 534, 535, 536, 537, 538, 540, 541, 550, 552, 553, 554, 555, 556, 557, 565, 566, 567, 568, 569, 570, 573, 574, 576, 577, 578, 579, 580, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 612, 613, 614, 615, 616, 617, 618, 620, 621, 622, 623, 624, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 662, 663, 664, 665,

667, 668, 670, 671, 673, 674, 675, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 735, 736, 737, 738, 739, 740, 741, 742, 744, 757, 759, 761, 763, 765, 771, 781, 782, 783, 784, 785, 786, 788, 790, 791, 792, 793, 795, 797, 798, 799, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 812, 813, 814, 815, 817, 818, 819, 820, 826, 827, 828, 829, 831, 833, 834, 835, 837, 838, 839, 840, 841, 845, 846, 847, 853, 854, 856, 857, 859, 860, 861, 862, 863, 864, 865, 866, 867, 868, 872, 874, 875, 876, 877, 878, 879, 880, 881, 882, 892, 894, 898, 900, 902, 903, 904, 905, 906, 907, 908, 909, 910, 911, 912, 913, 914, 915, 916, 917, 919, 920, 921, 922, 924, 925, 927, 928, 929, 930, 931, 932, 933, 934, 935, 937, 938, 939, 940, 941, 942, 944, 946, 949, 952, 953, 954, 955, 956, 957, 960, 961, 962, 963, 964, 966, 967, 968, 969, 970, 971, 972, 973, 974, 975, 976, 978, 979, 980, 982, 983, 984, 985, 986, 987, 988, 989, 990, 991, 1006, 1010, 1011, 1012, 1013, 1014, 1015, 1016, 1017, 1018, 1020, 1021, 1022, 1024, 1026, 1027, 1028, 1029, 1030, 1031, 1032, 1033, 1042, 1044, 1046, 1048, 1049, 1050, 1051, 1052, 1053, 1054, 1055, 1056, 1057, 1061, 1063, 1065, 1066, 1067, 1068, 1069, 1070, 1071, 1072, 1073, 1074, 1075, 1077, 1079, 1080, 1081, 1082, 1083, 1084, 1085, 1086, 1087, 1088, 1089, 1090, 1091, 1092, 1093, 1094, 1095, 1096, 1099, 1100, 1101, 1102, 1103, 1104, 1105, 1106, 1107, 1108, 1109, 1110, 1111, 1112, 1113, 1114, 1115, 1116, 1117, 1118, 1120, 1121, 1123, 1124, 1125, 1126, 1127, 1128, 1129, 1130, 1131, 1133, 1134, 1135, 1136, 1137, 1138, 1139, 1141, 1142, 1143, 1144, 1145, 1147, 1148, 1150, 1151, 1152, 1154, 1155, 1156, 1157, 1158, 1159, 1160, 1161, 1162, 1163, 1164, 1165, 1166, 1167, 1169, 1170, 1171, 1172, 1174, 1175, 1176, 1177, 1178, 1179, 1180, 1181, 1182, 1183, 1184, 1185, 1186, 1188, 1189, 1191, 1193, 1194, 1195, 1196, 1197, 1198, 1199, 1200, 1201, 1202, 1203, 1204, 1206, 1207, 1208, 1209, 1210, 1211, 1212, 1213, 1214, 1215, 1216, 1217, 1218, 1219, 1220, 1221, 1222, 1225, 1226, 1227, 1228, 1230, 1232, 1233, 1234, 1235, 1237, 1239, 1241, 1242, 1243, 1244, 1245, 1246, 1247, 1248, 1249, 1250, 1251, 1252, 1253, 1254, 1255, 1273, 1275, 1277, 1279, 1280, 1281, 1283, 1284, 1285, 1287, 1288, 1289, 1290, 1292, 1293, 1296, 1298, 1299, 1300, 1301, 1302, 1303, 1304, 1305, 1306, 1307, 1308, 1309, 1310, 1311, 1312, 1314, 1315, 1316, 1317, 1318, 1325, 1327, 1331, 1346, 1348, 1349,

1365, 1384, 1389, 1392, 1395, 1398, 1400, 1404, 1405, 1406, 1416, 1417, 1418, 1428, 1431, 1432, 1434, 1435, 1437, 1438, 1439, 1446, 1449, 1450, 1451, 1452, 1453, 1455, 1463, 1464, 1465, 1466, 1467, 1469, 1470, 1472, 1473, 1481, 1482, 1483, 1485, 1486, 1487, 1499, 1500, 1501, 1502, 1503, 1505, 1507, 1508, 1509, 1518, 1519, 1520, 1521, 1522, 1523, 1524, 1526, 1527, 1528, 1529, 1530, 1531, 1532, 1533, 1537, 1539, 1540, 1549, 1550, 1556, 1558, 1567, 1569, 1570, 1571, 1581, 1583, 1584, 1585, 1586, 1587, 1588, 1589, 1590, 1591, 1592, 1593, 1594, 1595, 1596, 1597, 1598, 1599, 1600, 1601, 1602, 1603, 1606, 1607, 1609, 1610, 1611, 1612, 1613, 1614, 1615, 1616, 1617, 1618, 1620, 1621, 1622, 1623, 1629, 1630, 1631, 1632, 1634, 1635, 1637, 1639, 1641, 1643, 1645, 1646, 1647, 1648, 1649, 1651, 1655, 1656, 1662, 1663, 1664, 1665, 1666, 1667, 1668, 1669, 1670, 1671, 1677, 1678, 1680, 1681, 1682, 1683, 1684, 1685, 1686, 1687, 1688, 1689, 1691, 1692, 1695, 1724, 1725, 1726, 1727, 1729, 1730, 1732, 1733, 1735, 1737, 1739, 1740, 1741, 1742, 1743, 1744, 1745, 1746, 1747, 1750, 1751, 1753, 1754, 1755, 1759, 1761, 1763, 1764, 1765, 1766, 1767, 1768, 1770, 1771, 1772, 1773, 1774, 1775, 1776, 1777, 1778, 1779, 1780, 1782, 1783, 1785, 1790, 1791, 1792, 1793, 1794, 1795, 1796, 1802, 1805, 1806, 1807, 1811, 1812, 1813, 1814, 1815, 1816, 1817, 1818, 1819, 1820, 1821, 1822, 1823, 1824, 1825, 1826, 1827, 1828, 1829, 1830, 1831, 1832, 1833, 1834.

cerr_mutex: 1689, 1695, 1716, 1790, 1791, 1802, 1803.

cert: 702, 707, 708, 730, 731, 1352, 1389, 1392, 1393, 1394, 1395, 1403, 1404, 1428, 1430, 1431, 1432, 1433, 1434, 1435, 1437, 1438, 1439, 1440.

cert_cred: 1401, 1402, 1500, 1501.

cert_list: 1389, 1392, 1428, 1430.

cert_list_size: 1389, 1392, 1395, 1428, 1430.

cert_ptr: 34, 39, 51, 106.

cert_vector: 481, 482.

certificate: 92, 309, 317, 319, 511, 1653.

certificate_id: 63, 92, 702, 706, 708, 710, 712, 715, 731, 733, 744, 746, 1653.

char_ctr: 217, 218, 219, 220, 221, 222, 223, 224, 225, 227, 228, 520, 526, 1549, 1552, 1560.

check_attrib: 1149, 1150, 1151.

CHECK_FILENAME: 1782.

check_irods_server: 1760, 1761, 1768.

check_prefix: 1597, 1619, 1620, 1623.

check_val: 1149, 1150, 1151.

chmod: 1466.

cin: 522, 526, 528.

cleanup_handler: 1447, 1449, 1451, 1454, 1455.

clear: 17, 19, 22, 23, 51, 52, 53, 183, 186, 188, 191, 274, 316, 317, 337, 349, 359, 374, 377, 389, 395, 408, 426, 462, 464, 481, 528, 535, 552, 646, 671, 732, 733, 765, 807, 841, 869, 870, 882, 894, 895, 896, 923, 956, 1007, 1008, 1022, 1044, 1049, 1058, 1059, 1117, 1166, 1201, 1236, 1240, 1242, 1257, 1270, 1271, 1287, 1296, 1328, 1329, 1349, 1360, 1361, 1527, 1605, 1606.

client_action_command_only: 59, 302.

client_action_map: 34, 36, 59, 1555, 1556, 1557, 1558.

client_action_name: 1555, 1557, 1558.

client_action_name_map: 34, 36, 60, 1556, 1558.

client_action_send_file: 59, 302.

client_action_unknown: 59, 302, 1557.

client_connect_auth: 1338, 1427.

client_finished: 33, 39, 106, 1521, 1524, 1530, 1531, 1532, 1550, 1551, 1552, 1565.

client_func: 33.

client_len: 1482, 1483, 1500, 1501.

client_sending_file_rule_func: 9, 33, 1628, 1629, 1635.

client_side_filename: 200, 201, 210, 211, 213, 214.

close: 51, 96, 100, 101, 126, 204, 223, 225, 229, 230, 250, 251, 369, 372, 425, 426, 486, 527, 528, 589, 590, 1416, 1420, 1465, 1466, 1467, 1469, 1472, 1473, 1482, 1483, 1485, 1487, 1500, 1501, 1502, 1507, 1520, 1521, 1522, 1524, 1532, 1539, 1558, 1560, 1566, 1615, 1616.

cmd_str: 341, 342, 344, 345, 346, 347, 348, 350, 351, 352, 354, 355, 357, 360, 363, 364, 369, 370, 371, 377, 378.

cmd_strm: 1662, 1666, 1668.

cmnd_strm: 1010, 1013, 1014, 1161, 1162, 1174, 1177, 1178.

column: 295, 513, 514, 515, 516, 517, 518.

comma_str: 816, 818, 844, 845, 953, 970, 1070, 1072, 1128, 1158, 1190, 1191, 1206, 1209, 1232, 1236, 1298, 1307, 1582, 1606, 1662, 1668.

comma_str_1: 1298, 1307.

command: 9, 19, 23, 25, 33, 93, 153, 157, 158, 160, 161, 163, 168, 169, 171, 172, 173, 175, 176, 179, 181, 182, 184, 185, 186, 189, 190, 191, 194, 195, 196, 197, 211, 213, 233, 328, 329, 334, 335, 336, 337, 372, 388, 390, 391, 393, 394, 395, 396, 397, 400, 403, 404, 405, 406, 408, 409, 410, 411, 413, 414, 415, 462, 535, 539, 551, 552, 553, 555, 556, 565, 566, 568, 569, 616, 645, 646, 685, 690, 798, 799, 800, 802, 804, 805, 807, 809, 810, 812, 813, 814, 815, 817, 819, 825, 833, 834, 838, 839, 841, 842, 845, 846, 852, 1045.

1046, 1048, 1236, 1528, 1537, 1554, 1556, 1641, 1647, 1648, 1649, 1651, 1654, 1655.
COMMAND_ONLY_TYPE: 9, 10, 57, 58, 59, 60, 93, 151, 201, 327, 341, 377, 384, 462, 535, 539, 551, 552, 553, 555, 556, 637, 645, 646, 685, 690, 798, 832, 1236, 1637, 1645.
commonName: 63, 319, 702, 708, 710, 712, 724, 731, 733, 744, 746, 1338, 1344, 1349, 1352, 1355, 1361, 1363, 1365, 1406, 1653.
config_dir: 1177, 1500, 1782.
conn: 50.
connect: 1416.
connect_func: 33, 1445, 1446, 1451, 1453, 1473, 1486, 1508.
CONNECT_H: 1458.
connection_type: 33, 39, 1469, 1483, 1501, 1526.
const_iterator: 25, 52, 106, 169, 244, 274, 337, 368, 371, 374, 380, 381, 395, 400, 425, 426, 535, 574, 763, 894, 898, 916, 917, 953, 1056, 1061, 1070, 1072, 1128, 1188, 1189, 1201, 1273, 1275, 1277, 1280, 1287, 1307, 1325, 1331, 1615, 1646, 1828, 1830.
content: 1295, 1296.
continue_on_error: 109, 110, 111, 1728, 1729, 1730.
control characters, ASCII : 610.
convert_seconds: 872, 929, 930, 932, 935, 936, 937, 1061, 1072, 1117, 1161, 1169, 1170, 1734, 1735, 1746.
convert_time_spec: 1738, 1739, 1747.
countryName: 63, 702, 708, 710, 712, 725, 731, 733, 744, 746, 1338, 1344, 1349, 1352, 1355, 1361, 1363, 1365, 1406, 1653.
cout: 525, 1453, 1468, 1470, 1483, 1500, 1501, 1551, 1688, 1689, 1831.
cout_mutex: 1688, 1689.
create_databases: 33.
CREATE_HANDLE_TYPE: 9, 10, 57, 58.
created: 588, 752, 755, 759, 764, 781, 791, 792, 870, 872, 876, 1038, 1040, 1042, 1044, 1059, 1061, 1069, 1093, 1126.
created_by_user_id: 588, 660, 664, 665, 675, 678, 752, 755, 759, 784, 791, 809, 842, 870, 872, 876, 948, 949, 1169, 1170, 1228.
created_by_user_name: 588, 752, 759.
created_str: 752, 759, 781, 791, 792, 870, 872, 1038.
CREATOR_INDEX: 171, 752, 753, 762, 763.
cred: 1384.
ctime: 1395.
ctr: 33, 35, 253, 329, 330, 341, 378, 395, 635, 1045, 1046, 1056.
curr_attribute: 363, 365, 367.

curr_ava: 1404, 1405, 1406.
curr_avu: 151, 183, 186, 188, 191, 387, 408, 467, 1107, 1112, 1113, 1114, 1115, 1117, 1151, 1174.
curr_byte_ctr: 423, 425.
curr_ca_filename: 1500.
curr_cert: 481.
curr_cert_filename: 1500.
curr_crl_filename: 1500.
curr_dc_metadata_sub_id: 363, 364, 367.
curr_dn: 1404, 1405.
curr_element_name: 352, 356.
curr_handle: 336, 337, 338, 952, 956, 1077, 1582, 1605, 1607, 1608.
curr_handle_id: 1077, 1105.
curr_handle_value: 902, 923, 952, 956, 1582, 1605, 1607, 1608.
curr_handle_value_id: 902, 912, 922, 923.
curr_id: 1646.
curr_idx: 902, 916.
curr_idx_type_map: 902, 913, 916, 919, 920.
curr_irods_object: 151, 163, 167, 169, 186, 191, 192, 193, 194, 195, 387, 388, 392, 394, 400, 405, 407, 408, 446, 450, 451, 462, 467, 468, 469, 470, 535, 552, 553, 1174, 1201.
curr_key_filename: 1500.
curr_metadata_handle_id: 151, 176.
curr_pid: 396, 397, 461, 462, 463, 464, 467, 474.
curr_row: 63, 67, 73, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 116, 119, 120, 124, 125, 151, 175, 176, 336, 337, 344, 345, 346, 347, 348, 349, 352, 354, 355, 356, 357, 358, 359, 363, 364, 365, 366, 434, 437, 478, 481, 492, 496, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 513, 516, 517, 518, 735, 740, 741, 764, 765, 767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786, 798, 807, 832, 841, 902, 912, 916, 952, 956, 966, 982, 1010, 1027, 1064, 1078, 1084, 1085, 1086, 1088, 1089, 1090, 1091, 1092, 1093, 1094, 1095, 1096, 1097, 1103, 1104, 1105, 1110, 1111, 1112, 1113, 1114, 1115, 1116, 1117, 1122, 1154, 1174, 1184, 1185, 1198, 1199, 1206, 1216, 1217, 1232, 1243, 1244, 1298, 1303, 1304, 1305, 1314, 1582, 1585, 1586, 1587, 1591, 1592, 1601, 1602, 1620, 1623, 1662, 1665, 1666, 1803, 1823, 1824, 1825.
curr_row_1: 344.
curr_str: 1754.
curr_tag: 423.
curr_term_name: 352, 358.
curr_time: 798, 799, 800, 804, 805, 807, 809, 810, 812, 813, 814, 815, 816, 817, 819, 825, 1151, 1157, 1158.

curr_type_ctr: 1828.
curr_user_id: 69, 70, 72, 73, 77, 91, 104, 490, 491, 492, 493, 494, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509.
curr_username: 69, 70, 72, 73, 1644, 1645, 1646.
curr_val: 1754.
curr_value: 363, 366, 367.
current_dir: 33.
d: 1272, 1273, 1324, 1325, 1326, 1327, 1354, 1355, 1357, 1359, 1641.
data: 336, 575, 576, 578, 579, 580, 752, 755, 757, 759, 761, 771, 787, 791, 794, 825, 852, 870, 872, 876, 956, 1226, 1227, 1286, 1287, 1291, 1292, 1295, 1296, 1377, 1378, 1380, 1382, 1390, 1406, 1608.
data_buffer: 33, 39, 106, 524, 526, 527, 528, 1549, 1550, 1551, 1552, 1561.
data_filename: 33, 106, 523, 526, 527, 1549, 1550, 1551, 1552, 1559, 1560.
data_length: 574, 575, 576, 578, 579, 580, 752, 755, 759, 771, 787, 791, 794, 823, 824, 825, 850, 851, 852, 869, 870, 872, 876, 956, 1226, 1227.
data_str: 463, 464, 656, 657, 882, 887, 900, 917, 921, 922, 923, 1223, 1224, 1228, 1829, 1830.
database: 550, 556, 873, 874, 1140, 1141, 1142, 1147, 1149, 1150, 1151, 1231, 1232, 1233, 1236, 1245, 1248.
database_name: 328.
date_ext_str: 1697, 1705.
date_str: 1678, 1682, 1683, 1700.
dbf: 1377, 1378, 1379, 1380, 1381, 1382.
dc_metadata_handle: 290, 444, 463, 466, 467.
dc_metadata_id: 1298, 1304, 1306, 1307, 1312.
dc_metadata_id_ptr: 1312.
DC_METADATA_INDEX: 401, 752, 753, 763.
DC_METADATA_IRODS_OBJECT_DELETED_FROM_ARCHIVE_INDEX: 37752, 240, 241, 242, 244, 245, 247, 248, 249, 250, 251, 252, 256, 260, 262, 266, 269, 270, 271, 272, 276, 278, 282, 285, 287, 288, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 339, 341, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 379, 381, 382, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 400, 401, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 429, 430, 431, 433, 434, 435, 436, 437, 438, 439, 441, 442, 443, 444, 445, 446, 448, 449, 451, 452, 454, 455, 456, 457, 458, 460, 462, 463, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 484, 485, 486, 487, 488, 490, 492, 493,

DC_METADATA_IRODS_OBJECT_REF_DELETED_FROM_GWIRDSIF_DB_INDEX: 753, 763, 1221.
DC_METADATA_IRODS_OBJECT_REF_INDEX: 464, 752, 753, 762, 763.
DC_METADATA_PID_INDEX: 181, 406, 463, 752, 753, 763.
DC_METADATA_REF_INDEX: 752, 753, 763.
dc_metadata_sub_id: 1298, 1305, 1307.
dc_metadata_sub_iter: 341, 367.
dc_metadata_type_vector: 341, 349, 350, 368, 369, 371, 374.
dc_metadata_type_vector_ptr: 253, 340, 341, 374.
dc_metadata_vector: 291, 394, 395, 396, 418, 426.
dcm: 341, 346, 347, 348, 349, 425, 426, 1287, 1290, 1292.
dcm_sub: 341, 352, 354, 355, 357, 359, 1287, 1290, 1292.
DCMTDTTP_H: 1335.
ddata: 789, 790, 791.
ddata_length: 789, 790, 791.
ddata_str: 882, 899, 900.
ddeleted_from_archive: 999, 1000, 1005, 1006.
ddeleted_from_gwirdsif_db: 999, 1000, 1005, 1006.
DEBUG: 39, 40, 41, 45, 46, 47, 49, 51, 52, 54, 55, 62, 63, 64, 67, 68, 70, 71, 73, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 110, 112, 113, 115, 117, 118, 119, 120, 121, 122, 123, 124, 126, 127, 128, 129, 136, 138, 141, 145, 146, 149, 151, 152, 154, 155, 156, 157, 158, 159, 160, 162, 163, 164, 165, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 186, 187, 188, 191, 193, 194, 195, 199, 201, 203, 204, 205, 206, 208, 209, 210, 211, 212, 213, 215, 218, 220, 221, 222, 223, 224, 225, 227, 228, 229, 230, 231, 235, 236, 237, 238, 239, 240, 241, 242, 244, 245, 247, 248, 249, 250, 251, 252, 256, 260, 262, 266, 269, 270, 271, 272, 276, 278, 282, 285, 287, 288, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 339, 341, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 379, 381, 382, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 400, 401, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 429, 430, 431, 433, 434, 435, 436, 437, 438, 439, 441, 442, 443, 444, 445, 446, 448, 449, 451, 452, 454, 455, 456, 457, 458, 460, 462, 463, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 484, 485, 486, 487, 488, 490, 492, 493,

494, 496, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 513, 514, 515, 516, 517, 518, 519, 520, 522, 526, 527, 528, 530, 531, 534, 535, 537, 538, 540, 541, 550, 552, 553, 554, 556, 557, 565, 566, 567, 568, 569, 570, 573, 574, 576, 577, 580, 582, 584, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 612, 613, 614, 615, 616, 617, 618, 621, 623, 624, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 657, 658, 659, 660, 664, 665, 667, 671, 673, 674, 675, 678, 681, 682, 683, 684, 686, 687, 688, 689, 691, 692, 693, 694, 695, 696, 697, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 735, 736, 737, 739, 740, 741, 742, 757, 759, 761, 763, 765, 771, 781, 782, 783, 784, 785, 786, 788, 790, 791, 792, 793, 795, 797, 801, 803, 806, 807, 808, 815, 816, 818, 820, 826, 827, 828, 829, 831, 835, 837, 840, 841, 844, 845, 847, 853, 854, 856, 860, 862, 864, 865, 866, 867, 868, 874, 875, 876, 878, 879, 881, 892, 894, 900, 902, 903, 904, 906, 908, 909, 911, 912, 913, 915, 916, 917, 919, 920, 921, 922, 924, 925, 927, 929, 930, 931, 933, 934, 937, 938, 939, 940, 942, 944, 946, 949, 952, 953, 954, 955, 956, 957, 960, 961, 962, 963, 964, 966, 968, 969, 970, 972, 973, 974, 976, 978, 979, 980, 982, 985, 986, 987, 988, 989, 991, 1006, 1010, 1012, 1013, 1014, 1015, 1016, 1020, 1021, 1022, 1026, 1027, 1028, 1029, 1030, 1031, 1033, 1042, 1044, 1046, 1048, 1051, 1052, 1053, 1054, 1055, 1056, 1057, 1063, 1065, 1066, 1067, 1068, 1069, 1070, 1071, 1072, 1073, 1074, 1075, 1077, 1079, 1080, 1082, 1083, 1084, 1085, 1086, 1087, 1088, 1089, 1090, 1091, 1092, 1093, 1094, 1095, 1096, 1098, 1099, 1101, 1102, 1103, 1104, 1105, 1106, 1107, 1108, 1109, 1110, 1112, 1117, 1118, 1120, 1121, 1123, 1124, 1126, 1127, 1128, 1129, 1130, 1131, 1133, 1134, 1135, 1136, 1138, 1139, 1141, 1143, 1144, 1145, 1147, 1148, 1150, 1151, 1152, 1154, 1156, 1158, 1160, 1161, 1162, 1163, 1164, 1165, 1167, 1168, 1170, 1171, 1172, 1174, 1176, 1177, 1178, 1180, 1181, 1182, 1183, 1184, 1186, 1188, 1189, 1191, 1193, 1194, 1195, 1196, 1197, 1198, 1199, 1200, 1201, 1202, 1203, 1204, 1206, 1208, 1209, 1210, 1211, 1212, 1213, 1214, 1215, 1216, 1217, 1218, 1219, 1220, 1221, 1222, 1224, 1225, 1226, 1227, 1228, 1230, 1232, 1235, 1236, 1237, 1239, 1242, 1243, 1244, 1246, 1247, 1248, 1249, 1251, 1253, 1255, 1273, 1277, 1279, 1280, 1281, 1283, 1284, 1285, 1287, 1288, 1289, 1290, 1292,

1293, 1296, 1298, 1299, 1300, 1301, 1303, 1304, 1305, 1306, 1307, 1308, 1309, 1310, 1312, 1314, 1315, 1316, 1317, 1318, 1325, 1327, 1346, 1348, 1349, 1389, 1392, 1395, 1398, 1400, 1404, 1405, 1406, 1416, 1418, 1428, 1429, 1430, 1431, 1432, 1434, 1435, 1437, 1438, 1439, 1440, 1446, 1447, 1449, 1450, 1451, 1452, 1453, 1455, 1462, 1463, 1464, 1465, 1467, 1469, 1470, 1473, 1481, 1482, 1483, 1487, 1499, 1500, 1501, 1502, 1503, 1505, 1508, 1509, 1517, 1518, 1519, 1520, 1521, 1522, 1523, 1524, 1527, 1528, 1529, 1530, 1531, 1533, 1534, 1537, 1539, 1540, 1549, 1550, 1552, 1553, 1556, 1558, 1559, 1560, 1561, 1562, 1563, 1565, 1566, 1567, 1569, 1571, 1581, 1583, 1584, 1586, 1589, 1590, 1591, 1592, 1593, 1594, 1596, 1597, 1598, 1599, 1600, 1601, 1602, 1603, 1606, 1607, 1609, 1610, 1611, 1612, 1613, 1614, 1615, 1616, 1617, 1618, 1620, 1621, 1623, 1629, 1630, 1631, 1632, 1634, 1635, 1637, 1639, 1641, 1643, 1645, 1646, 1648, 1649, 1655, 1656, 1662, 1663, 1664, 1665, 1666, 1667, 1668, 1669, 1670, 1671, 1677, 1678, 1680, 1681, 1682, 1683, 1684, 1685, 1686, 1687, 1688, 1689, 1691, 1692, 1695, 1696, 1697, 1698, 1699, 1700, 1701, 1702, 1703, 1704, 1705, 1706, 1707, 1709, 1710, 1711, 1724, 1725, 1726, 1727, 1729, 1730, 1732, 1733, 1735, 1739, 1741, 1742, 1743, 1744, 1746, 1747, 1750, 1751, 1753, 1754, 1755, 1758, 1759, 1761, 1764, 1765, 1766, 1768, 1770, 1771, 1774, 1775, 1776, 1777, 1778, 1779, 1780, 1782, 1783, 1785, 1790, 1791, 1792, 1793, 1794, 1795, 1796, 1802, 1803, 1805, 1806, 1807, 1811, 1812, 1813, 1816, 1817, 1818, 1819, 1820, 1821, 1822, 1824, 1825, 1826, 1827, 1828, 1829, 1830, 1831, 1832, 1833, 1834.

DEBUG_COMPILE: 41, 45, 46, 52, 54, 55, 62, 63, 64, 67, 68, 70, 71, 73, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 112, 115, 117, 118, 119, 120, 121, 122, 123, 124, 126, 127, 128, 129, 136, 138, 141, 145, 146, 149, 151, 152, 154, 155, 156, 157, 158, 159, 160, 162, 163, 164, 165, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 186, 187, 188, 191, 193, 194, 195, 199, 201, 203, 204, 205, 206, 208, 209, 210, 211, 212, 213, 215, 218, 220, 221, 222, 223, 224, 227, 228, 230, 231, 235, 237, 239, 240, 241, 242, 244, 245, 247, 248, 249, 250, 251, 252, 260, 262, 266, 269, 270, 271, 272, 275, 276, 278, 282, 285, 287, 288, 328, 329, 330, 331, 332, 333, 334, 335, 337, 341, 343, 344, 345, 346, 347, 348, 349, 350, 351, 354, 355, 356, 357, 358, 359, 360, 361, 362, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374,

375, 376, 377, 379, 381, 382, 384, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 400, 401, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 429, 430, 431, 433, 434, 435, 436, 437, 438, 439, 441, 442, 443, 444, 445, 446, 448, 449, 451, 452, 454, 455, 456, 457, 458, 460, 462, 463, 465, 466, 467, 468, 469, 470, 473, 475, 476, 477, 478, 479, 480, 481, 482, 484, 485, 486, 487, 488, 490, 492, 493, 494, 496, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 513, 514, 515, 516, 517, 518, 519, 520, 522, 526, 527, 528, 530, 531, 534, 535, 537, 538, 540, 541, 550, 552, 553, 554, 556, 557, 565, 566, 567, 568, 569, 570, 573, 574, 576, 577, 580, 582, 584, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 612, 613, 614, 615, 616, 617, 618, 621, 623, 624, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 657, 658, 659, 660, 664, 665, 667, 671, 673, 674, 675, 678, 681, 682, 683, 684, 686, 687, 688, 689, 691, 692, 693, 694, 695, 696, 697, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 735, 736, 737, 739, 740, 741, 742, 757, 759, 761, 763, 765, 771, 781, 782, 783, 784, 785, 786, 788, 790, 791, 792, 793, 795, 797, 801, 803, 806, 807, 808, 815, 816, 818, 820, 826, 827, 828, 829, 831, 835, 837, 840, 841, 844, 845, 847, 853, 854, 856, 860, 862, 864, 865, 866, 867, 868, 874, 875, 876, 878, 879, 881, 892, 894, 900, 902, 903, 904, 906, 908, 909, 911, 912, 913, 915, 916, 917, 919, 920, 921, 922, 924, 925, 927, 929, 930, 931, 933, 934, 937, 938, 939, 940, 942, 944, 946, 949, 952, 953, 954, 955, 956, 957, 960, 961, 962, 963, 964, 966, 968, 969, 970, 972, 973, 974, 976, 978, 979, 980, 982, 985, 986, 987, 988, 989, 991, 1006, 1010, 1012, 1013, 1014, 1015, 1016, 1020, 1021, 1022, 1026, 1027, 1028, 1029, 1030, 1031, 1033, 1042, 1044, 1046, 1048, 1051, 1052, 1053, 1054, 1055, 1056, 1057, 1063, 1065, 1066, 1067, 1068, 1069, 1070, 1071, 1072, 1073, 1074, 1075, 1077, 1079, 1080, 1082, 1083, 1084, 1085, 1086, 1087, 1088, 1089, 1090, 1091, 1092, 1093, 1094, 1095, 1096, 1098, 1099, 1101, 1102, 1103, 1104, 1105, 1106, 1107, 1108, 1109, 1110, 1112, 1117, 1118, 1120, 1121, 1123, 1124, 1126, 1127, 1128, 1129, 1130, 1131, 1133, 1134, 1135, 1136, 1138, 1139, 1141, 1143, 1144, 1145, 1147, 1148, 1150, 1151, 1152, 1154, 1156, 1158, 1160, 1161, 1162, 1163, 1164, 1165, 1167, 1168, 1170, 1171, 1172, 1174, 1176, 1177, 1178, 1180, 1181, 1182, 1183, 1184, 1186, 1188, 1189, 1191, 1193, 1194, 1195, 1196, 1197, 1198, 1199, 1200, 1201, 1202, 1203, 1204, 1206, 1208, 1209, 1210, 1211, 1212, 1213, 1214, 1215, 1216, 1217, 1218, 1219, 1220, 1221, 1222, 1225, 1226, 1227, 1228, 1230, 1232, 1235, 1236, 1237, 1239, 1242, 1243, 1244, 1246, 1247, 1248, 1249, 1251, 1253, 1255, 1273, 1277, 1279, 1280, 1281, 1283, 1284, 1285, 1287, 1288, 1289, 1290, 1292, 1293, 1296, 1298, 1299, 1300, 1301, 1303, 1304, 1305, 1306, 1307, 1308, 1309, 1310, 1312, 1314, 1315, 1316, 1317, 1318, 1325, 1327, 1428, 1429, 1430, 1431, 1432, 1434, 1435, 1437, 1438, 1439, 1440, 1449, 1450, 1451, 1452, 1453, 1455, 1463, 1464, 1465, 1467, 1469, 1470, 1473, 1481, 1482, 1483, 1487, 1499, 1500, 1501, 1502, 1503, 1505, 1508, 1509, 1518, 1519, 1520, 1521, 1522, 1523, 1527, 1528, 1529, 1530, 1531, 1533, 1534, 1537, 1539, 1540, 1550, 1552, 1553, 1555, 1556, 1558, 1559, 1560, 1561, 1562, 1563, 1565, 1566, 1567, 1569, 1571, 1581, 1583, 1584, 1586, 1589, 1590, 1591, 1592, 1593, 1594, 1596, 1597, 1598, 1599, 1600, 1601, 1602, 1603, 1606, 1607, 1609, 1610, 1611, 1612, 1613, 1614, 1615, 1616, 1617, 1618, 1620, 1621, 1623, 1629, 1630, 1631, 1632, 1634, 1635, 1637, 1639, 1641, 1643, 1645, 1646, 1648, 1649, 1655, 1656, 1662, 1663, 1664, 1665, 1666, 1667, 1668, 1669, 1670, 1671, 1677, 1678, 1680, 1681, 1682, 1683, 1684, 1685, 1686, 1687, 1688, 1689, 1691, 1692, 1695, 1696, 1697, 1698, 1699, 1700, 1701, 1702, 1703, 1704, 1705, 1706, 1707, 1709, 1710, 1711, 1724, 1727, 1730, 1732, 1733, 1735, 1739, 1741, 1742, 1743, 1744, 1746, 1747, 1750, 1751, 1753, 1754, 1755, 1761, 1764, 1765, 1766, 1768, 1770, 1771, 1774, 1775, 1776, 1777, 1778, 1779, 1780, 1782, 1783, 1785, 1802, 1805, 1806, 1807, 1811, 1812, 1813, 1816, 1817, 1818, 1819, 1820, 1821, 1822, 1824, 1825, 1826, 1827, 1828, 1829, 1830, 1831, 1832, 1833, 1834.

debug_level: 1759.

dec: 25, 164, 341, 512, 744, 797, 927, 1365, 1435, 1603.

decrypt: 1769, 1770, 1780, 1783.

DEFAULT_CA_FILENAME: 1500.

DEFAULT_CERT_FILENAME: 1500.

DEFAULT_CRL_FILENAME: 1500.

default_filename: 1781, 1782.

default_handle_prefix: 33, 86, 106, 166, 309, 317, 319, 396, 461, 511, 1654.

default_handle_prefix_id: 33, 39, 85, 106, 309, 315, 317, 319, 511, 1654.

default_institute_id: 33, 39, 69, 87, 106, 309, 315, 317, 319, 511, 1654.
default_institute_name: 33, 69, 88, 106, 309, 317, 319, 511, 1654.
DEFAULT_KEY_FILENAME: 1500.
default_prefix: 69.
default_prefix_id: 69.
default_resource: 33.
delay: 538, 539.
delay_time: 929, 930, 936.
delay_val: 9.
delay_value: 9, 13, 19, 23, 25, 33, 39, 106, 538, 669, 685, 686, 796, 797, 798, 799, 800, 804, 805, 807, 809, 810, 812, 813, 814, 815, 816, 817, 819, 825, 926, 927, 928, 930, 936, 1154, 1157, 1158, 1161, 1171.
delayed_response_deque: 33, 51, 106, 645, 646, 1527.
DELEGATE_PRIVILEGE: 35, 36, 106, 319, 499, 512.
DELETE_DATABASE_ENTRIES_AND_EXIT: 1306, 1307, 1308, 1309.
delete_from_archive: 530, 887, 1173, 1174, 1204, 1223.
delete_from_archive_timestamp: 535, 1038, 1040, 1042, 1044, 1059, 1061, 1095, 1151.
delete_from_database: 530, 669, 926, 927, 942, 1009, 1010, 1020.
delete_from_database_timestamp: 588, 752, 755, 759, 786, 870, 872, 876, 935.
delete_from_database_timestamp_str: 752, 759, 786, 870, 872.
delete_from_gwirdsif_db: 887, 1205, 1206, 1222, 1223.
delete_from_gwirdsif_db_timestamp: 535, 1038, 1040, 1042, 1044, 1059, 1061, 1096.
DELETE_HANDLE_TYPE: 9, 10, 57, 58.
delete_handle_value: 686, 796, 797, 829.
DELETE_HANDLE_VALUE_TYPE: 9, 10, 57, 58.
delete_handle_values: 501, 965, 966, 976, 977, 978, 980, 1245.
DELETE_HANDLE_VALUES_PRIVILEGE: 35, 36, 106, 501, 512, 685.
delete_handles: 500.
DELETE_HANDLES_PRIVILEGE: 35, 36, 106, 319, 500, 512, 662, 665.
delete_hs_admin_handle_values: 502.
DELETE_HS_ADMIN_HANDLE_VALUES_PRIVILEGE: 35, 36, 106, 502, 512, 802, 810.
delete_irods_avu: 1009, 1010, 1022.
delete_last_hs_admin_handle_value: 503.
DELETE_LAST_HS_ADMIN_HANDLE_VALUE_PRIVILEGE: 36, 106, 503, 512, 814, 815.

delete_temp_files_str: 1678, 1682.
deleted: 752, 1038.
deleted_from_archive: 996, 998, 1000, 1004, 1006, 1008, 1024, 1038, 1040, 1042, 1044, 1059, 1061, 1092, 1151, 1161, 1236.
deleted_from_gwirdsif_db: 996, 998, 1000, 1004, 1006, 1008, 1024, 1038, 1040, 1042, 1044, 1059, 1061, 1236.
delim_0: 1731, 1732, 1733.
delim_1: 1731, 1732, 1733.
delimiter: 1749, 1750.
delimiter_1: 1749, 1750.
Depth: 1263, 1287, 1290, 1292, 1293.
deque: 33, 235, 244, 645, 646, 796, 797, 830, 831, 1528.
dest: 1752, 1753, 1754.
dest_length: 1752, 1753, 1754.
destroy_java_vm: 50.
DH_BITS: 1370, 1402, 1489, 1491, 1494, 1512.
dh_params: 1397, 1398, 1482, 1491, 1494, 1500.
dir: 255, 256, 259, 260, 261, 264, 265, 268, 269, 270, 271.
dir_list: 235, 243, 244.
directory: 1781, 1782.
directory_list: 274, 275, 276, 277, 280, 281, 284, 285, 286.
dirname: 9, 19, 23, 25, 599.
discard: 235, 239, 245, 246, 249, 250, 251.
Distinguished_Name: 33.
distinguished_name: 33, 63, 67, 79, 106, 171, 309, 317, 319, 511, 1502, 1641.
distinguished_name_rule_func: 9, 33, 309, 1636, 1637, 1643.
distinguished_name_str: 1345, 1636, 1637, 1639, 1640, 1641.
Distinguished_Name_Type: 33, 63, 309, 702, 1338, 1341, 1342, 1343, 1344, 1346, 1349, 1352, 1354, 1355, 1356, 1357, 1358, 1359, 1361, 1363, 1365, 1366, 1502, 1641.
dn: 69, 70, 72, 73, 1389, 1395.
dn_fields: 1406.
do_irods_user_metadata: 253, 340, 341, 375, 395.
do_output: 253, 340, 341, 369, 372, 395, 635.
do_response: 134, 135, 136, 138, 139, 140, 141, 142.
dparams: 1398.
DSTNGNMT_H: 1367.
DUBLIN_CORE_ABSTRACT_TERM: 1264, 1265.
DUBLIN_CORE_ACCESSRIGHTS_TERM: 1264, 1265.
DUBLIN_CORE_ACCRUALMETHOD_TERM: 1264, 1265.
DUBLIN_CORE_ACCRUALPERIODICITY_TERM: 1264, 1265.
DUBLIN_CORE_ACCRUALPOLICY_TERM: 1264, 1265.

DUBLIN_CORE_ALTERNATIVE_TERM: 1264, 1265.
 DUBLIN_CORE_ATTRIBUTE_TYPE: 1264.
 DUBLIN_CORE_AUDIENCE_TERM: 1264, 1265.
 DUBLIN_CORE_AVAILABLE_TERM: 1264, 1265.
 DUBLIN_CORE_BIBLIOGRAPHIC_CITATION_TERM: 1264, 1265.
 DUBLIN_CORE_CONFORMSTO_TERM: 1264, 1265.
 DUBLIN_CORE_CONTRIBUTOR_ELEMENT: 1264, 1265.
 DUBLIN_CORE_CONTRIBUTOR_TERM: 1264, 1265.
 DUBLIN_CORE_COVERAGE_ELEMENT: 1264, 1265.
 DUBLIN_CORE_COVERAGE_TERM: 1264, 1265.
 DUBLIN_CORE_CREATED_TERM: 1264, 1265.
 DUBLIN_CORE_CREATOR_ELEMENT: 1264, 1265.
 DUBLIN_CORE_CREATOR_TERM: 1264, 1265.
 DUBLIN_CORE_DATE_ELEMENT: 1264, 1265.
 DUBLIN_CORE_DATE_TERM: 1264, 1265.
 DUBLIN_CORE_DATEACCEPTED_TERM: 1264, 1265.
 DUBLIN_CORE_DATECOPYRIGHTED_TERM: 1264, 1265.
 DUBLIN_CORE_DATESUBMITTED_TERM: 1264, 1265.
 DUBLIN_CORE_DESCRIPTION_ELEMENT: 1264, 1265.
 DUBLIN_CORE_DESCRIPTION_TERM: 1264, 1265.
 DUBLIN_CORE_EDUCATIONLEVEL_TERM: 1264, 1265.
 DUBLIN_CORE_ELEMENT_TYPE: 1264, 1265.
 DUBLIN_CORE_EXTENT_TERM: 1264, 1265.
 DUBLIN_CORE_FORMAT_ELEMENT: 1264, 1265.
 DUBLIN_CORE_FORMAT_TERM: 1264, 1265.
 DUBLIN_CORE_HASFORMAT_TERM: 1264, 1265.
 DUBLIN_CORE_HASPART_TERM: 1264, 1265.
 DUBLIN_CORE_HASVERSION_TERM: 1264, 1265.
 DUBLIN_CORE_IDENTIFIER_ELEMENT: 1264, 1265.
 DUBLIN_CORE_IDENTIFIER_TERM: 1264, 1265.
 DUBLIN_CORE_INSTRUCTIONALMETHOD_TERM: 1264, 1265.
 DUBLIN_CORE_ISFORMATOF_TERM: 1264, 1265.
 DUBLIN_CORE_ISPARTOF_TERM: 1264, 1265.
 DUBLIN_CORE_ISREFERENCEDBY_TERM: 1264, 1265.
 DUBLIN_CORE_ISREPLACEDBY_TERM: 1264, 1265.
 DUBLIN_CORE_ISREQUIREDBY_TERM: 1264, 1265.
 DUBLIN_CORE_ISSUED_TERM: 1264, 1265.
 DUBLIN_CORE_ISVERSIONOF_TERM: 1264, 1265.
 DUBLIN_CORE_LANGUAGE_ELEMENT: 1264, 1265.
 DUBLIN_CORE_LANGUAGE_TERM: 1264, 1265.
 DUBLIN_CORE_LICENSE_TERM: 1264, 1265.
 DUBLIN_CORE_MEDIATOR_TERM: 1264, 1265.
 DUBLIN_CORE_MEDIUM_TERM: 1264, 1265.
Dublin_Core_Metadata_Sub_Type: 341, 1264, 1273, 1275, 1280, 1287, 1292, 1307, 1319, 1320, 1321, 1322, 1323, 1324, 1325, 1326, 1327, 1329, 1331.
Dublin_Core_Metadata_Type: 253, 291, 340, 341, 350, 368, 371, 374, 394, 395, 396, 399, 418, 425, 426, 1264, 1265, 1266, 1267, 1268, 1269, 1271, 1272, 1273, 1275, 1277, 1279, 1281, 1283, 1285, 1287, 1290, 1292, 1294, 1296, 1297, 1298, 1312, 1314, 1318, 1319, 1331, 1333, 1335.
 DUBLIN_CORE_MODIFIED_TERM: 1264, 1265.
 DUBLIN_CORE_NULL_ELEMENT: 1264, 1265.
 DUBLIN_CORE_NULL_TERM: 1264, 1265.
 DUBLIN_CORE_NULL_TYPE: 1264, 1265.
 DUBLIN_CORE_PROVENANCE_TERM: 1264, 1265.
 DUBLIN_CORE_PUBLISHER_ELEMENT: 1264, 1265.
 DUBLIN_CORE_PUBLISHER_TERM: 1264, 1265.
Dublin_Core_Qualifier_Type: 1319.
 DUBLIN_CORE_QUALIFIER_TYPE: 1264, 1265.
 DUBLIN_CORE_REFERENCES_TERM: 1264, 1265.
 DUBLIN_CORE_RELATION_ELEMENT: 1264, 1265.
 DUBLIN_CORE_RELATION_TERM: 1264, 1265.
 DUBLIN_CORE_REPLACESTERM: 1264, 1265.
 DUBLIN_CORE_REQUIRESTERM: 1264, 1265.
 DUBLIN_CORE_RIGHTS_ELEMENT: 1264, 1265.
 DUBLIN_CORE_RIGHTS_TERM: 1264, 1265.
 DUBLIN_CORE_RIGHTSHOLDER_TERM: 1264, 1265.
 DUBLIN_CORE_SOURCE_ELEMENT: 1264, 1265.
 DUBLIN_CORE_SOURCE_TERM: 1264, 1265.
 DUBLIN_CORE_SPATIAL_TERM: 1264, 1265.
 DUBLIN_CORE SUBJECT ELEMENT: 1264, 1265.
 DUBLIN_CORE SUBJECT TERM: 1264, 1265.
 DUBLIN_CORE_TABLEOFCONTENTS_TERM: 1264, 1265.
 DUBLIN_CORE_TEMPORAL_TERM: 1264, 1265.
 DUBLIN_CORE_TITLE_ELEMENT: 1264, 1265.
 DUBLIN_CORE_TITLE_TERM: 1264, 1265.
 DUBLIN_CORE_TYPE_ELEMENT: 1264, 1265.
 DUBLIN_CORE_TYPE_TERM: 1264, 1265.
 DUBLIN_CORE_VALID_TERM: 1264, 1265.
EEXIST: 245.
el: 1286, 1287, 1288, 1291, 1292.
element: 1287, 1288, 1289.
element_ctr: 1287, 1288, 1289, 1290.
element_map: 1264, 1265, 1277, 1288, 1289.
element_id: 355, 1280, 1290, 1292, 1307, 1319, 1321, 1325, 1329, 1331.
element_map: 1264, 1265, 1277, 1280, 1288, 1331.
empty: 45, 72, 73, 78, 138, 165, 210, 211, 213, 214, 219, 220, 222, 223, 227, 230, 233, 236, 237, 243, 248, 259, 269, 271, 285, 337, 410, 445, 523, 552, 565, 568, 569, 574, 610, 656, 663, 677, 794, 800, 803, 805, 807, 812, 813, 817, 819, 834, 837, 839, 841, 845, 846, 882, 953, 1024, 1061, 1136, 1151, 1228, 1238, 1528, 1537, 1549, 1550, 1551, 1552, 1556, 1559, 1560, 1584, 1589, 1590, 1593, 1594, 1595, 1599, 1604, 1606, 1613, 1616, 1668, 1730, 1733, 1775, 1782, 1818, 1819.

ENABLE_SRP: 1384.
enabled: [1623](#).
encrypted_text: [1769](#), [1770](#), 1775, 1776, 1778.
end: 25, 52, 106, 136, 138, 169, 181, 244, 274, 331, 337, 350, 367, 368, 371, 374, 380, 381, 395, 396, 400, 401, 404, 425, 426, 463, 482, 535, 537, 539, 552, 574, 645, 646, 763, 816, 821, 844, 848, 894, 898, 916, 917, 918, 919, 920, 945, 946, 953, 970, 984, 1012, 1056, 1061, 1070, 1072, 1117, 1128, 1136, 1151, 1158, 1161, 1164, 1168, 1169, 1170, 1176, 1188, 1189, 1190, 1201, 1203, 1209, 1210, 1220, 1225, 1228, 1230, 1236, 1247, 1248, 1257, 1273, 1275, 1277, 1280, 1284, 1288, 1289, 1290, 1291, [1292](#), 1294, 1307, 1325, 1331, 1449, 1450, 1472, 1485, 1507, 1528, 1537, 1538, 1539, 1556, 1557, 1558, 1615, 1631, 1642, 1646, 1704, 1730, 1829, 1830, 1832.
END_ASSIGN: 498, [510](#).
END_AVUS_MARK_FOR_DELETION: 1162, [1163](#).
END_CALL_IMETA: 1143, [1145](#).
end_char: [1769](#), [1770](#), 1779.
END_CREATE_PID_STR: 1599, [1608](#).
END_DECRYPT_PASSWORD: 1782, [1783](#).
END_DELETE_FROM_ARCHIVE: 1197, 1200, 1202, [1204](#).
END_DELETE_FROM_GWIRDSIF_DB: 1215, 1218, 1219, [1222](#).
END_DELETE_IRODS_AVU: 1020, [1022](#).
END_GET_PID_COUNTER: [1603](#).
end_server_enabled: 619, 620, 622.
END_SERVER_TYPE: [9](#), [10](#), 57, 58, 1539.
end_val: [425](#).
endl: 25, 39, 40, 41, 45, 46, 47, 49, 51, 52, 54, 55, 62, 63, 64, 65, 66, 67, 68, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 106, 110, 111, 112, 113, 115, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 136, 138, 139, 140, 141, 142, 145, 146, 147, 148, 149, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 193, 194, 195, 196, 197, 199, 201, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 235, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 256, 257, 258, 259, 260, 261, 262, 264, 265, 266, 267, 268, 269, 270, 271, 272, 274, 275, 276, 277, 278, 280, 281, 282, 283, 284, 285, 287, 288, 319, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 339, 341, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 381, 382, 384, 386, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 400, 401, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 429, 430, 431, 433, 434, 435, 436, 437, 438, 439, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 460, 461, 462, 463, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 484, 485, 486, 487, 488, 490, 491, 492, 493, 494, 495, 496, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 540, 541, 550, 552, 553, 554, 555, 556, 557, 565, 566, 567, 568, 569, 570, 573, 574, 576, 577, 578, 579, 580, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 612, 613, 614, 615, 616, 617, 618, 620, 621, 622, 623, 624, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 662, 663, 664, 665, 667, 668, 670, 671, 673, 674, 675, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741, 742, 744, 757, 759, 761, 763, 765, 771, 781, 782, 783, 784, 785, 786, 788, 790, 791, 792, 793, 795, 797, 798, 799, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 812, 813, 814, 815, 817, 818, 819, 820, 826, 827, 828, 829, 831, 833, 834, 835, 837, 838, 839, 840, 841, 842, 845, 846, 847, 853, 854, 856, 857, 859, 860, 861, 862, 863, 864, 865, 866, 867, 868, 872, 874, 875, 876, 877, 878, 879, 880, 881, 882, 892, 894, 898, 900, 902, 903, 904, 905, 906, 907, 908, 909, 910, 911, 912, 913, 914, 915, 916, 917, 919, 920, 921, 922, 924, 925, 927, 928, 929, 930, 931, 932, 933, 934, 935, 937, 938, 939, 940, 941, 942, 944, 946, 949, 952, 953, 954, 955, 956, 957, 960, 961, 962, 963, 964, 966, 967, 968, 969, 970, 971, 972, 973, 974, 975, 976, 978, 979, 980, 982, 983, 984, 984, 985, 986, 987, 988, 989, 990, 991, 1006, 1010, 1011, 1012, 1013, 1014, 1015, 1016, 1017, 1018, 1020, 1021, 1022, 1024, 1026, 1027, 1028, 1029, 1030, 1031,

1032, 1033, 1042, 1044, 1046, 1048, 1049, 1050, 1051, 1052, 1053, 1054, 1055, 1056, 1057, 1061, 1063, 1065, 1066, 1067, 1068, 1069, 1070, 1071, 1072, 1073, 1074, 1075, 1077, 1079, 1080, 1081, 1082, 1083, 1084, 1085, 1086, 1087, 1088, 1089, 1090, 1091, 1092, 1093, 1094, 1095, 1096, 1099, 1100, 1101, 1102, 1103, 1104, 1105, 1106, 1107, 1108, 1109, 1110, 1111, 1112, 1113, 1114, 1115, 1116, 1117, 1118, 1120, 1121, 1123, 1124, 1125, 1126, 1127, 1128, 1129, 1130, 1131, 1133, 1134, 1135, 1136, 1137, 1138, 1139, 1141, 1142, 1143, 1144, 1145, 1147, 1148, 1150, 1151, 1152, 1154, 1155, 1156, 1157, 1158, 1159, 1160, 1161, 1162, 1163, 1164, 1165, 1166, 1167, 1169, 1170, 1171, 1172, 1174, 1175, 1176, 1177, 1178, 1179, 1180, 1181, 1182, 1183, 1184, 1185, 1186, 1188, 1189, 1191, 1193, 1194, 1195, 1196, 1197, 1198, 1199, 1200, 1201, 1202, 1203, 1204, 1206, 1207, 1208, 1209, 1210, 1211, 1212, 1213, 1214, 1215, 1216, 1217, 1218, 1219, 1220, 1221, 1222, 1225, 1226, 1227, 1228, 1230, 1232, 1233, 1234, 1235, 1237, 1239, 1241, 1242, 1243, 1244, 1245, 1246, 1247, 1248, 1249, 1250, 1251, 1252, 1253, 1254, 1255, 1273, 1275, 1277, 1279, 1280, 1281, 1283, 1284, 1285, 1287, 1288, 1289, 1290, 1292, 1293, 1296, 1298, 1299, 1300, 1301, 1302, 1303, 1304, 1305, 1306, 1307, 1308, 1309, 1310, 1311, 1312, 1314, 1315, 1316, 1317, 1318, 1325, 1327, 1331, 1346, 1348, 1349, 1365, 1384, 1389, 1392, 1395, 1398, 1400, 1404, 1405, 1406, 1416, 1417, 1418, 1428, 1431, 1432, 1434, 1435, 1437, 1438, 1439, 1446, 1448, 1449, 1450, 1451, 1452, 1453, 1455, 1463, 1464, 1465, 1466, 1467, 1468, 1469, 1470, 1472, 1473, 1481, 1482, 1483, 1485, 1486, 1487, 1499, 1500, 1501, 1502, 1503, 1505, 1507, 1508, 1509, 1518, 1519, 1520, 1521, 1522, 1523, 1524, 1526, 1527, 1528, 1529, 1530, 1531, 1532, 1533, 1537, 1539, 1540, 1549, 1550, 1551, 1552, 1553, 1556, 1558, 1559, 1560, 1561, 1562, 1563, 1564, 1565, 1566, 1567, 1569, 1570, 1571, 1581, 1583, 1584, 1585, 1586, 1587, 1588, 1589, 1590, 1591, 1592, 1593, 1594, 1595, 1596, 1597, 1598, 1599, 1600, 1601, 1602, 1603, 1606, 1607, 1609, 1610, 1611, 1612, 1613, 1614, 1615, 1616, 1617, 1618, 1620, 1621, 1622, 1623, 1629, 1630, 1631, 1632, 1634, 1635, 1637, 1639, 1641, 1643, 1645, 1646, 1647, 1648, 1649, 1651, 1655, 1656, 1662, 1663, 1664, 1665, 1666, 1667, 1668, 1669, 1670, 1671, 1677, 1678, 1680, 1681, 1682, 1683, 1684, 1685, 1686, 1687, 1688, 1689, 1691, 1692, 1695, 1696, 1697, 1698, 1699, 1700, 1701, 1702, 1703, 1704, 1705, 1706, 1707, 1708, 1709, 1710, 1711, 1724, 1725, 1726, 1727, 1729, 1730, 1732, 1733, 1735, 1737, 1739, 1740, 1741, 1742, 1743, 1744, 1745, 1746, 1747, 1750, 1751, 1753, 1754, 1755, 1759, 1761, 1763, 1764, 1765, 1766, 1767, 1768, 1770, 1771, 1772, 1773, 1774, 1775, 1776, 1777, 1778, 1779, 1780, 1782, 1783, 1785, 1790, 1791, 1792, 1793, 1794, 1795, 1796, 1802, 1805, 1806, 1807, 1811, 1812, 1813, 1814, 1815, 1816, 1817, 1818, 1819, 1820, 1821, 1822, 1823, 1824, 1825, 1826, 1827, 1828, 1829, 1830, 1831, 1832, 1833, 1834.

ENOENT: 239, 242, 245, 1463, 1696, 1705, 1782.

env_strm: 101.

eof: 522, 526, 528, 1560.

EOF: 522, 1252, 1700.

erase: 154, 240, 243, 259, 267, 274, 283, 860, 864, 1052, 1053, 1054, 1055, 1296, 1346, 1347, 1349, 1450, 1452, 1453, 1594, 1598, 1634, 1742, 1743, 1766.

err: 1416, 1482, 1494, 1500, 1512.

err_log_filename: 1687.

err_log_strm_mutex: 1687.

err_out_str: 1678.

err_ret_val: 235, 239, 245, 246, 249, 251.

errno: 41, 52, 95, 96, 99, 100, 138, 146, 157, 158, 176, 211, 223, 225, 227, 229, 230, 239, 241, 242, 245, 246, 250, 267, 268, 270, 283, 284, 346, 347, 348, 354, 355, 357, 364, 369, 389, 437, 452, 453, 485, 486, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 518, 527, 589, 697, 715, 716, 718, 719, 720, 721, 781, 782, 784, 785, 786, 798, 865, 912, 916, 929, 1014, 1042, 1044, 1049, 1086, 1089, 1090, 1091, 1092, 1093, 1094, 1095, 1096, 1105, 1112, 1125, 1157, 1162, 1178, 1185, 1199, 1217, 1252, 1304, 1305, 1435, 1437, 1438, 1463, 1464, 1465, 1466, 1467, 1470, 1482, 1500, 1520, 1531, 1532, 1560, 1561, 1562, 1563, 1564, 1566, 1587, 1602, 1613, 1614, 1615, 1616, 1668, 1681, 1682, 1683, 1684, 1696, 1699, 1700, 1705, 1735, 1737, 1754, 1766, 1771, 1772, 1775, 1782, 1785, 1814, 1818, 1825.

error_warning_str: 1446, 1448, 1449, 1452, 1453.

errors_occurred: 34, 38, 39, 64, 65, 66, 67, 70, 72, 73, 74, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 91, 92, 95, 96, 97, 98, 99, 100, 101, 106, 115, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 138, 139, 140, 142, 146, 147, 148, 153, 157, 158, 161, 163, 168, 171, 173, 175, 176, 179, 181, 182, 184, 185, 189, 190, 194, 195, 196, 197, 204, 207, 210, 211, 214, 219, 223, 225, 226, 227, 229, 230, 245, 249, 257, 258, 261, 264, 265, 268, 269, 275, 277, 280, 281, 284, 285, 287, 328, 329, 334, 336, 337, 344, 345, 346, 347, 348,

350, 351, 352, 354, 355, 357, 360, 363, 364, 369, 370, 371, 378, 388, 390, 393, 394, 395, 396, 398, 400, 401, 403, 404, 405, 406, 408, 419, 421, 422, 423, 424, 425, 426, 434, 435, 437, 441, 445, 446, 447, 450, 452, 453, 456, 461, 463, 465, 466, 468, 469, 470, 471, 474, 476, 479, 480, 481, 485, 486, 491, 493, 496, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 521, 527, 531, 535, 538, 552, 553, 555, 556, 567, 569, 593, 596, 599, 638, 653, 656, 657, 662, 663, 664, 665, 667, 671, 673, 678, 680, 682, 687, 692, 695, 696, 697, 1448, 1631, 1634, 1647, 1648, 1649, 1651.

ETX (ASCII control character) : 610.

EX_RFC2818_H: [1442](#).

exchange_data_with_client: [9](#), [33](#), 200, 255, 273, 565, 568, 592, [752](#), [887](#), [1264](#), [1319](#), 1445, 1448, 1514, [1516](#), [1517](#), 1540, 1565.

exchange_data_with_server: [9](#), [33](#), [1264](#), [1319](#), 1522, 1546, [1548](#), [1549](#), 1572.

EXCHNCLI_H: [1545](#).

EXCHNSRV_H: [1577](#).

EXCPTNTP_H: [6](#).

existing_metadata_vector: [395](#), 396.

exit: 1451, 1791, 1792, 1793, 1794, 1795, 1796, 1797, 1805, 1806, 1807, 1811, 1812, 1814, 1815, 1818, 1821, 1822, 1823, 1825, 1831, 1832, 1834.

expiration_time: [1389](#), 1395.

expire: 1731.

expires: [33](#), 39, 130, 131, 132, 133.

extract: 799, 833, [855](#), [856](#), 868.

extract_dn_fields: [702](#), 1393, 1394, [1403](#), [1404](#), 1406, 1439.

F_OK: 41, 157, 239, 241, 1613, 1696, 1705, 1782.

failed: [39](#), 44, 46, 47, [1582](#), 1584, 1585, 1587, 1588, 1591, 1592, 1594, 1595, 1597, 1600, 1601, 1602, 1603, 1607, 1609, 1610, 1612, 1613, 1614, 1615, 1616, 1617.

false: 13, 23, 39, 44, 46, 47, 51, 52, 62, 69, 70, 94, 106, 107, 109, 110, 114, 115, 136, 145, 151, 201, 203, 218, 234, 235, 239, 256, 274, 295, 297, 318, 319, 327, 341, 384, 388, 389, 390, 393, 395, 396, 399, 418, 423, 429, 433, 441, 444, 447, 460, 474, 478, 490, 491, 513, 520, 522, 530, 535, 538, 550, 552, 553, 565, 568, 573, 574, 589, 590, 592, 595, 598, 601, 604, 607, 614, 618, 620, 624, 626, 629, 631, 634, 635, 637, 641, 644, 649, 652, 655, 660, 675, 684, 689, 694, 697, 706, 709, 711, 714, 719, 720, 733, 735, 743, 744, 752, 755, 757, 759, 761, 763, 765, 790, 797, 798, 807, 816, 821, 823, 831, 844, 845, 848, 849, 850, 856, 870, 873, 874, 892, 894, 900, 902, 919, 923, 927, 929, 944, 949, 952, 960, 966, 978, 982, 998, 999, 1005, 1006, 1008,

1010, 1026, 1040, 1042, 1044, 1046, 1051, 1059, 1063, 1076, 1077, 1090, 1091, 1092, 1120, 1132, 1133, 1141, 1142, 1143, 1150, 1151, 1153, 1154, 1161, 1162, 1174, 1201, 1206, 1224, 1230, 1232, 1240, 1273, 1277, 1279, 1283, 1287, 1292, 1296, 1298, 1307, 1314, 1325, 1327, 1346, 1389, 1394, 1398, 1400, 1403, 1404, 1416, 1428, 1446, 1447, 1455, 1462, 1471, 1481, 1484, 1499, 1506, 1508, 1517, 1518, 1520, 1523, 1524, 1528, 1549, 1550, 1551, 1552, 1553, 1559, 1560, 1561, 1563, 1566, 1567, 1576, 1581, 1582, 1607, 1620, 1629, 1632, 1637, 1645, 1662, 1677, 1695, 1724, 1728, 1729, 1732, 1734, 1735, 1739, 1750, 1753, 1759, 1761, 1770, 1782, 1785, 1790, 1791, 1802, 1803, 1831.

fclose: 229, 1524, 1569, 1570, 1571.

fd: [96](#), 100, [204](#), [235](#), 250, [369](#), [485](#), 486, [520](#), 527, [573](#), 589, [1517](#), 1518, 1520, [1549](#), 1566, [1582](#), 1614, 1615, 1616, [1784](#), [1785](#).

feof: 264, 280, 420, 1055, 1668, 1684, 1685, 1702, 1703.

ferror: 264, 280, 1055, 1668, 1684, 1685, 1702, 1703.

fetch_handle_from_database: [292](#), [293](#), [429](#), 431, [433](#), 438, 439, 657, 664, 678, 958, [959](#), [960](#), 964.

fetch_handles_from_database: 179, [294](#), 400, 430, 440, [441](#), 443, [951](#), [952](#), 957, 961, 1166, 1189, 1208, 1219.

ffilename: [789](#), [790](#), 794.

fgets: 98, 420, 1049, 1668.

field_ctr: [63](#), 64, [73](#), [107](#), [108](#), [116](#), 118, 123, [151](#), 173, [327](#), 334, 337, [344](#), 345, 350, 351, 360, 362, [434](#), [444](#), [478](#), 480, [492](#), 493, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, [513](#), 515, [714](#), 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, [735](#), 736, 739, [764](#), [765](#), 766, 783, 784, 785, 786, [798](#), 804, 806, 807, 815, [832](#), 838, 840, 841, [902](#), 906, 909, 913, [952](#), 953, 956, [966](#), [982](#), [1010](#), [1027](#), [1064](#), [1078](#), 1080, 1099, 1107, 1110, [1122](#), [1154](#), [1174](#), 1183, 1196, [1206](#), 1214, [1298](#), 1301, [1314](#), [1582](#), 1584, 1591, 1592, 1600, [1620](#), 1621, [1662](#), 1664, [1723](#), [1724](#), [1803](#), 1821.

field_ctr_vector: [109](#), [110](#), 111, [1232](#), 1240, 1242, 1257, [1728](#), [1729](#), 1730.

field_ctr_1: [344](#).

fifo.pathname: 167, 396, 461, [752](#), [887](#), [1580](#), [1581](#), 1613, 1614, 1615, 1616, [1803](#), 1813, 1827, [1831](#).

filename: [9](#), [33](#), [135](#), [136](#), 137, [200](#), [201](#), 205, 206, 207, 208, 209, 210, 211, 213, 214, [217](#), [218](#), 219, 220, 222, 223, 227, 230, [253](#), [296](#), [327](#), 329, 331, 337, [341](#), 343, 344, 345, 346, 347, 348, 350, 351, 352, 354, 355, 357, 360, 363, 364, 369, 370, 371,

372, 377, 378, 380, 381, 395, 478, 486, 574, 752, 759, 787, 794, 870, 872, 923, 956, 1045, 1046, 1132, 1133, 1134, 1628, 1629, 1630, 1631, 1678, 1683, 1684, 1694, 1695, 1696, 1698, 1705, 1706, 1707, 1709, 1710, 1781, 1782, 1783.
filename_str: 550, 552, 555, 556.
filename_vector: 33, 53.
filename_1: 134, 135, 136, 138, 254, 327, 337.
find: 166, 203, 259, 276, 343, 367, 386, 534, 535, 552, 857, 861, 919, 943, 944, 946, 984, 1012, 1052, 1053, 1054, 1055, 1169, 1170, 1176, 1225, 1288, 1289, 1290, 1349, 1472, 1485, 1507, 1536, 1555, 1593, 1598, 1631, 1642, 1646, 1740, 1742, 1766, 1782, 1829.
find_avu: 1151, 1229, 1230.
find_last_of: 152, 206, 238, 240, 243, 267, 283, 1347.
FINISH: 656, 657, 662, 663, 664, 665, 667, 668, 670, 671, 673, 677, 678, 679, 680, 682, 685, 686, 687, 690, 692.
finish: 1450, 1783, 1790, 1802.
first: 106, 763, 894, 916, 919, 1277, 1280, 1307, 1325, 1331, 1449, 1450.
first_time: 1047, 1049, 1051, 1549, 1550, 1552, 1553, 1559, 1561, 1782.
flags: 9, 19, 23, 25, 138, 158, 200, 201, 203, 211, 253, 254, 273, 276, 327, 329, 332, 341, 342, 377, 380, 395, 534, 602, 605, 608.
flush: 1446.
fopen: 157, 227, 1522, 1567.
force: 290, 444, 447, 448, 457, 458, 460, 470, 1132, 1133, 1134.
force_add: 384, 385, 393.
force_store: 384, 385, 413.
force_write: 574, 589.
format_str: 1735, 1736, 1737.
found: 816, 818, 819, 820, 821, 825, 827, 828, 844, 845, 846, 847, 848, 852, 919, 1161, 1162.
fp: 97, 98, 126, 138, 139, 140, 141, 142, 146, 147, 148, 149, 158, 160, 161, 211, 213, 214, 227, 229, 260, 261, 263, 264, 265, 268, 269, 276, 277, 279, 280, 281, 284, 285, 419, 420, 444, 452, 453, 455, 456, 1010, 1014, 1015, 1016, 1018, 1019, 1045, 1048, 1049, 1050, 1055, 1056, 1154, 1162, 1163, 1174, 1178, 1179, 1248, 1249, 1250, 1251, 1517, 1518, 1522, 1524, 1549, 1567, 1569, 1570, 1571, 1662, 1668, 1682, 1683, 1684, 1685, 1695, 1699, 1700, 1702, 1703, 1762, 1763, 1764, 1765, 1775, 1776.
fprintf: 1284, 1384, 1428, 1429, 1430, 1433, 1440.
fread: 126, 139, 146, 158, 161, 211, 214, 227, 263, 279, 456, 1015, 1163, 1177, 1178, 1249, 1685, 1703, 1764, 1765, 1775, 1776.
free: 1376, 1819.
freeaddrinfo: 1418.
front: 430, 963, 1534, 1554.
fscanf: 453, 1015, 1162, 1177, 1683, 1700.
fstream: 33.
func_ptr: 34, 36, 1536, 1555.
fwrite: 251, 576, 582, 771, 791, 871, 872, 1226, 1227, 1296, 1519, 1750, 1754.
gai_strerror: 1416.
gcount: 425, 526, 1560.
gcry_control: 31, 308, 323, 547, 563, 701, 751, 886, 995, 1037, 1337, 1369, 1444, 1460, 1479, 1497, 1515, 1547, 1627.
gen_temp_file: 234, 235, 236, 249, 250, 251.
generate_dh_params: 1397, 1398, 1482, 1490, 1491, 1500.
generate_pids: 166, 167, 168, 396, 461, 653, 752, 887, 1580, 1581, 1618, 1831.
GENERATE_PIDS_POST_FIFO: 1613, 1614, 1615, 1616.
generate_rsa_params: 1399, 1400, 1500.
generate_tans: 121, 1661, 1662, 1671, 1788, 1791, 1796.
get: 200, 201, 215, 608, 612.
get_avus_from_irods_system: 33, 329, 378, 1045, 1046, 1057.
get_database_username: 61, 62, 68, 77, 78, 702, 1503.
get_datestamp: 64, 65, 66, 67, 1731, 1732, 1733.
get_deleted_from_archive: 996.
get_deleted_from_gwirdsif_db: 996.
get_expires: 132, 133.
get_from_database: 92, 192, 193, 194, 195, 388, 402, 446, 471, 535, 553, 734, 735, 742, 1076, 1077, 1118, 1201.
get_handle: 254, 326, 327, 331, 339, 637, 638, 752, 764.
GET_HANDLE_TYPE: 9, 10, 57, 58.
get_highest_value: 295, 513, 519, 1029, 1067, 1068.
get_id: 996.
get_in_addr: 1418, 1421, 1422.
get_input: 298, 520, 528, 1551.
get_irods_auth_filename: 309.
get_irods_env_filename: 309, 1013, 1177, 1203.
get_metadata: 35, 253, 340, 341, 382, 395, 635, 1045.
GET_METADATA_IRODS_USER_METADATA: 345, 375.
GET_METADATA_TYPE: 9, 10, 57, 58, 634.
get_pids: 35.
get_privileges: 91, 104, 489, 490, 510.

get_seconds_since_epoch: 728, 729, 1117, [1736](#), [1737](#).
get_thread_ctr: [35](#), 1404.
GET_TYPE: [9](#), [10](#), 57, 58.
get_user: 33, [69](#), [70](#), 102, 1012, 1176, 1503, 1639, 1640, 1641, 1649, 1678.
get_user_id: [309](#), 1504.
get_user_info_func: [9](#), [33](#), [309](#), 649, 650, [702](#), [1644](#), [1645](#), 1656.
GET_USER_INFO_TYPE: [9](#), [10](#), 57, 58.
getaddrinfo: 1415, 1416.
getenv: 239, 1810.
getline: 389, 526.
getpwuid_r: 40, 1818.
getuid: 1818.
global_debug_level: [1758](#), 1759.
global_thread_ctr: 1471, 1472, 1484, 1485, 1506, 1507.
global_thread_ctr_wrapped_around: 1471, 1484, 1506.
global_user_info_map: [310](#), [311](#), 1012, 1176, 1504, 1643, 1650.
global_user_info_map_mutex: [310](#), [311](#), 1012, 1176, 1504, 1643, 1650.
gmtime_r: 744, 792, 793, 1042, 1044, 1125, 1437, 1438, 1697, 1732, 1734, 1735.
GNTLSFNC_H: [1410](#).
gnutls_anon_allocate_server_credentials: 1482.
gnutls_anon_server_credentials_t: [1494](#).
gnutls_anon_set_server_dh_params: 1482.
gnutls_auth_get_type: 1384.
gnutls_bye: 51.
GNUTLS_CERT_INVALID: 1429.
GNUTLS_CERT_REQUEST: 1402.
GNUTLS_CERT_REVOKED: 1429.
GNUTLS_CERT_SIGNER_NOT_FOUND: 1429.
gnutls_certificate_allocate_credentials: 1500.
gnutls_certificate_credentials_t: 1401, 1402, 1500.
gnutls_certificate_get_peers: 1389, 1430.
gnutls_certificate_get_x509_cas: 1500.
gnutls_certificate_server_set_request: 1402.
gnutls_certificate_set_dh_params: 1500.
gnutls_certificate_set_rsa_export_params: 1500.
gnutls_certificate_set_x509_crl_file: 1500.
gnutls_certificate_set_x509_key_file: 1428, 1500.
gnutls_certificate_set_x509_trust_file: 1428, 1500.
gnutls_certificate_type_get: 1384, 1389, 1430.
gnutls_certificate_type_get_name: 1384.
gnutls_certificate_verify_peers2: 1428.
gnutls_check_version: 1384.
gnutls_cipher_get: 1384.
gnutls_cipher_get_name: 1384.

gnutls_compression_get: 1384.
gnutls_compression_get_name: 1384.
GNUTLS_CRD_ANON: 1384, 1489.
GNUTLS_CRD_CERTIFICATE: 1384, 1402.
GNUTLS_CRD_IA: 1384.
GNUTLS_CRD_PSK: 1384.
GNUTLS_CRD_SRP: 1384.
gnutls_credentials_set: 1402, 1489.
gnutls_credentials_type_t: 1384.
GNUTLS_CRT_PRINT_FULL: 1390.
GNUTLS_CRT_PRINT_ONELINE: 1390.
GNUTLS_CRT_X509: 1389, 1430.
gnutls_datum_t: 1377, 1378, 1379, [1380](#), 1381, 1382, [1389](#), [1390](#), 1391, 1398, 1428.
gnutls_db_set_ptr: 1402.
gnutls_db_set_remove_function: 1402.
gnutls_db_set_retrieve_function: 1402.
gnutls_db_set_store_function: 1402.
gnutls_deinit: 51, 1501, 1502, 1503.
gnutls_dh_get_prime_bits: 1384.
gnutls_dh_params_generate2: 1491.
gnutls_dh_params_import_pkcs3: 1398.
gnutls_dh_params_init: 1398, 1491.
gnutls_dh_params_t: 1397, 1398, [1494](#), 1500.
gnutls_dh_set_prime_bits: 1402, 1489.
GNUTLS_E_SUCCESS: 1434, 1500.
gnutls_handshake: 1483, 1501.
gnutls_init: 1402, 1489.
gnutls_kx_algorithm_t: 1384.
GNUTLS_KX_DHE_DSS: 1384.
GNUTLS_KX_DHE_RSA: 1384.
gnutls_kx_get: 1384.
gnutls_kx_get_name: 1384.
gnutls_mac_get: 1384.
gnutls_mac_get_name: 1384.
gnutls_malloc: 1380.
gnutls_perror: 1500.
gnutls_pk_algorithm_get_name: 1395.
gnutls_pk_algorithm_t: 1395.
gnutls_priority_set_direct: 1402, 1489.
gnutls_protocol_get_name: 1384.
gnutls_protocol_get_version: 1384.
gnutls_psk_client_get_hint: 1384.
gnutls_psk_server_get_username: 1384.
gnutls_record_recv: 232, 251, 1518, 1563.
gnutls_record_send: 223, 225, 227, 230, 232, 697, 1502, 1526, 1530, 1531, 1532, 1552, 1560, 1561, 1562.
gnutls_rsa_params_generate2: 1400.
gnutls_rsa_params_init: 1400.
gnutls_rsa_params_t: 1399, 1400, 1500.
GNUTLS_SERVER: 1402, 1489.

gnutls_session_t: 33, 702, 1383, 1384, 1388, 1389, 1401, [1402](#), 1427, 1428, 1488, [1489](#).

GNUTLS_SHUT_RDWR: 51.

gnutls_srp_server_get_username: 1384.

gnutls_strerror: 223, 225, 229, 230, 1434, 1483, 1501, 1532.

gnutls_transport_ptr_t: 1483, 1501.

gnutls_transport_set_ptr: 1483, 1501.

gnutls_x509_ava_st: [1390](#), 1404.

gnutls_x509_crt_check_hostname: 1433.

gnutls_x509_crt_deinit: 1395, 1430, 1431, 1432, 1433, 1434, 1435, 1437, 1438, 1439, 1440.

gnutls_x509_crt_get_activation_time: 1395, 1432, 1437.

gnutls_x509_crt_get_dn: 1395.

gnutls_x509_crt_get_expiration_time: 1395, 1431, 1438.

gnutls_x509_crt_get_issuer: 1404.

gnutls_x509_crt_get_issuer_dn: 1395.

gnutls_x509_crt_get_pk_algorithm: 1395.

gnutls_x509_crt_get_serial: 1395, 1434.

gnutls_x509_crt_get_subject: 1404.

gnutls_x509_crt_get_version: 1395.

gnutls_x509_crt_import: 1392, 1430.

gnutls_x509_crt_init: 1392, 1430.

gnutls_x509_crt_print: 1390.

gnutls_x509_crt_t: 702, 1389, 1403, 1404, 1428, 1500.

gnutls_x509_dn_get_rdn_ava: 1405.

gnutls_x509_dn_t: [1390](#), 1404.

GNUTLS_X509_FMT_DER: 1392, 1430.

GNUTLS_X509_FMT_PEM: 1398, 1500.

good: 370, 425, 527.

gpg_homedir: 126.

gpg_key_id: 126, 1790, 1802.

gpg_passphrase: 97, 1772, 1774, 1783, 1790, 1802.

gpg_passphrase_fifo_fd: 97, 1772.

gpg_passphrase_fifo_name: 97, 1772, 1773, 1774, 1775.

gpg_passphrase_length: 97, 1790, 1802.

group: 666.

Group_Type: 33, 106.

group_vector: [33](#), 51, 106.

GW_ERROR: 531, 532, 798, 799.

GW_HANDLE_NOT_FOUND: 678.

GW_HANDLE_NOT_MARKED_FOR_DELETION: 679, 984.

GW_HANDLE_VALUE_ALREADY_MARKED_FOR_DELETION:

GW_HANDLE_VALUE_ERROR: 807, 813, 833, 841.

GW_HANDLE_VALUE_NOT_FOUND: 805, 812, 839.

GW_HANDLE_VALUE_NOT_MARKED_FOR_DELETION: 845.

GW_INVALID_HANDLE_VALUE_SPECIFIER: 800, 834.

GW_IRODS_OBJECT_ALREADY_MARKED_FOR_DELETION:

GW_NO_PRIVILEGE_ERROR: 685, 690, 802, 809, 810, 814, 815, 842.

GW_SERVER_SIDE_DATABASE_ERROR: 535, 804, 807, 815, 819, 838, 841, 846.

GW_SUCCESS: 823, 850.

gwirdcli: 40.

gwirdsif_hostname: 101.

gwrdifpk: [5](#), [6](#), [27](#), [305](#), [306](#), [320](#), [544](#), [560](#), [698](#), [748](#), [883](#), [992](#), [1034](#), [1035](#), [1258](#), [1259](#), [1333](#), [1335](#), [1366](#), [1409](#), [1441](#), [1457](#), [1476](#), [1477](#), [1494](#), [1495](#), [1512](#), [1513](#), [1544](#), [1545](#), [1576](#), [1577](#), [1624](#), [1625](#), [1657](#), [1658](#), [1672](#), [1673](#), [1712](#), [1786](#), [1787](#), [1798](#), [1835](#).

gwrdbap: 40.

gwstrerror: 680.

h: [756](#), [757](#), [758](#), [759](#), [891](#), [892](#), [893](#), [894](#).

handle: [9](#), [13](#), [17](#), [19](#), [23](#), [25](#), [33](#), [181](#), [182](#), [183](#), [184](#), [185](#), [186](#), [292](#), [293](#), [337](#), [338](#), [401](#), [403](#), [404](#), [405](#), [406](#), [407](#), [408](#), [410](#), [412](#), [428](#), [429](#), [430](#), [433](#), [438](#), [444](#), [462](#), [463](#), [465](#), [466](#), [467](#), [574](#), [590](#), [655](#), [657](#), [660](#), [664](#), [667](#), [668](#), [669](#), [671](#), [675](#), [678](#), [679](#), [752](#), [759](#), [766](#), [783](#), [791](#), [798](#), [799](#), [801](#), [807](#), [815](#), [832](#), [833](#), [835](#), [841](#), [855](#), [856](#), [867](#), [870](#), [872](#), [876](#), [887](#), [894](#), [896](#), [898](#), [922](#), [923](#), [927](#), [956](#), [959](#), [960](#), [963](#), [984](#), [985](#), [1223](#), [1224](#), [1225](#), [1228](#), [1608](#), [1832](#).

Handle System: 34.

handle_data: [1284](#), [1295](#), [1296](#).

handle_database: [660](#), [671](#), [672](#), [675](#), [803](#), [805](#), [815](#), [816](#), [837](#), [839](#), [844](#), [902](#), [903](#), [905](#), [922](#), [934](#), [935](#), [937](#), [938](#), [952](#), [953](#), [961](#), [963](#), [1183](#), [1190](#), [1208](#), [1209](#), [1214](#), [1219](#).

handle_id: [292](#), [337](#), [348](#), [398](#), [411](#), [429](#), [430](#), [437](#), [438](#), [444](#), [462](#), [463](#), [588](#), [752](#), [755](#), [759](#), [767](#), [791](#), [815](#), [870](#), [872](#), [876](#), [887](#), [890](#), [894](#), [896](#), [898](#), [906](#), [913](#), [914](#), [915](#), [922](#), [923](#), [937](#), [956](#), [959](#), [960](#), [961](#), [978](#), [988](#), [1191](#), [1209](#), [1264](#), [1267](#), [1271](#), [1275](#), [1316](#), [1580](#), [1582](#), [1591](#), [1599](#), [1606](#), [1607](#), [1608](#).

handle_id_set: [965](#), [966](#), [967](#), [970](#), [978](#), [1232](#), [1243](#), [1244](#), [1245](#).

handle_id_vector: [167](#), [169](#), [294](#), [388](#), [400](#), [430](#), [441](#), [462](#), [951](#), [952](#), [953](#), [961](#), [1038](#), [1044](#), [1059](#), [1061](#), [1070](#), [1071](#), [1105](#), [1106](#), [1128](#), [1129](#), [1164](#), [1166](#), [1208](#).

handle_id_vector_ptr: [752](#), [887](#), [1580](#), [1581](#), [1599](#), [1606](#).

8HANDLE_MARKED_FOR_DELETION_INDEX: [752](#), [753](#), [763](#), [932](#).

handle_name_string_vector: [1038](#), [1044](#), [1059](#), [1061](#), [1105](#), [1106](#), [1164](#).

handle_obj: [336](#), [337](#).

5HANDLE_rows: [660](#), [669](#), [671](#), [672](#), [675](#), [926](#).

927, 938.
handle_str: 293, 327, 332, 337, 433, 434.
Handle_Type: 9, 19, 29, 151, 181, 290, 292, 293, 294, 336, 384, 400, 401, 404, 428, 429, 430, 433, 440, 441, 444, 461, 463, 655, 660, 675, 752, 883, 884, 887, 888, 889, 890, 891, 892, 893, 894, 896, 898, 900, 902, 925, 927, 935, 942, 944, 946, 949, 951, 952, 956, 957, 959, 960, 961, 964, 966, 976, 978, 980, 982, 991, 992, 993, 1038, 1061, 1166, 1168, 1174, 1189, 1190, 1206, 1208, 1209, 1210, 1219, 1220, 1223, 1224, 1245, 1580, 1581, 1582, 1831, 1832.
handle_value: 33, 894.
handle_value_ctr: 1190, 1191, 1206, 1210, 1220, 1221.
handle_value_id: 588, 752, 755, 759, 768, 791, 818, 845, 870, 872, 876, 909, 987, 1580, 1582, 1591, 1592, 1599, 1606, 1607, 1608.
handle_value_id_vector: 167, 169, 1038, 1044, 1059, 1061.
handle_value_id_vector_ptr: 752, 887, 1580, 1581, 1599, 1606.
handle_value_map: 337, 574, 667, 668, 887, 894, 896, 898, 923, 945, 946, 949, 956, 983, 984, 1169, 1170, 1225, 1228, 1607, 1608.
Handle_Value_Triple: 9, 33, 171, 444, 463, 655, 882, 899, 900, 901, 902, 917, 918, 921, 932, 1803, 1830.
Handle_Value_Type: 9, 33, 171, 181, 182, 292, 294, 336, 337, 401, 403, 404, 405, 406, 428, 440, 463, 464, 574, 686, 691, 750, 752, 753, 754, 755, 756, 757, 758, 759, 760, 761, 763, 765, 788, 790, 795, 797, 807, 815, 816, 821, 829, 831, 841, 844, 848, 854, 856, 868, 870, 872, 874, 881, 887, 894, 898, 901, 902, 919, 932, 943, 944, 945, 947, 952, 982, 992, 993, 1038, 1077, 1164, 1169, 1170, 1191, 1206, 1210, 1221, 1225, 1582, 1608, 1804, 1828, 1829.
handle_value_type_vector: 965, 966, 967, 970, 977, 978, 1232, 1245.
handle_value_vector: 1038, 1608.
handle_vector: 151, 167, 169, 170, 171, 179, 180, 181, 182, 294, 384, 396, 397, 398, 401, 409, 410, 411, 412, 413, 430, 440, 441, 442, 461, 462, 463, 752, 887, 951, 952, 956, 957, 961, 963, 1038, 1044, 1059, 1061, 1164, 1165, 1166, 1167, 1168, 1169, 1170, 1208, 1209, 1210, 1580, 1581, 1607, 1608, 1831, 1832.
handles_database: 966, 968, 969, 970, 971, 972, 973, 974, 986, 987, 988.
handles_table_name: 1582, 1599, 1604.
HAVE_CONFIG_H: 3, 8, 31, 308, 323, 547, 563, 701, 751, 886, 995, 1037, 1261, 1337, 1369, 1412, 1426, 1444, 1460, 1479, 1497, 1515, 1547, 1579, 1627, 1660, 1675, 1715, 1789, 1801.
hex: 341, 512, 744, 927, 1435, 1603, 1606, 1750.
hex_decode: 1748, 1752, 1753, 1755.
hex_encode: 576, 577, 583, 584, 876, 1748, 1749, 1750, 1751.
hexl_str: 574, 576, 577, 583, 584.
hhandle: 764, 765, 766, 783, 789, 790, 791.
hhandle_id: 789, 790, 791, 1313, 1314, 1316.
hhandle_value_id: 789, 790, 791.
hints: 1416.
HNDLTYPE_H: 993.
HNDLVLTP_H: 884.
homedir: 33.
hostname: 702, 1427, 1428, 1433.
hour_offset: 1731, 1732.
hs_admin_found: 807.
hton: 1482, 1500.
hv: 807, 808, 809, 810, 811, 841, 842, 843.
hv_iter: 982, 984, 987, 1164, 1169, 1170, 1206, 1225, 1226, 1227, 1228.
hv_str: 796, 797, 798, 799, 800, 802, 804, 805, 807, 809, 810, 812, 813, 814, 815, 817, 819, 823, 830, 831, 833, 834, 838, 839, 841, 842, 845, 846, 850, 855, 856, 857, 859, 860, 861, 863, 864, 867.
hv_vector: 807, 811, 812, 813, 814, 815, 816, 821, 841, 843, 844, 848.
hvt: 9, 19, 33, 444, 463, 464, 655, 656, 657, 882, 900, 932, 933, 1803, 1829.
hvt_vector: 171, 444, 463, 464, 465, 466, 900, 901, 902, 905, 917, 918, 1803, 1829, 1830, 1832, 1833.
i: 46, 47, 124, 175, 336, 345, 352, 363, 420, 468, 481, 576, 807, 825, 841, 852, 898, 916, 956, 1001, 1002, 1003, 1004, 1049, 1103, 1110, 1184, 1198, 1216, 1239, 1241, 1242, 1243, 1244, 1287, 1296, 1380, 1382, 1387, 1392, 1416, 1448, 1472, 1482, 1485, 1500, 1507, 1528, 1605, 1705, 1730, 1740, 1741, 1744, 1745, 1750, 1754, 1763, 1777, 1811, 1812, 1828, 1831, 1832.
icommands: 273, 376, 381.
icommands_flag_str: 33, 273.
id: 192, 346, 350, 352, 354, 359, 360, 408, 467, 474, 535, 996, 998, 1000, 1004, 1006, 1008, 1020, 1021, 1024, 1029, 1030, 1038, 1040, 1042, 1044, 1059, 1061, 1067, 1069, 1070, 1072, 1076, 1080, 1086, 1089, 1090, 1091, 1092, 1093, 1094, 1095, 1096, 1099, 1100, 1107, 1111, 1112, 1117, 1121, 1126, 1128, 1158, 1173, 1175, 1182, 1201, 1207, 1211, 1212, 1213, 1236, 1264, 1267, 1271, 1275, 1306, 1307, 1316, 1319, 1321, 1329, 1331, 1468, 1469, 1473, 1483, 1486, 1501, 1508.

id_only: 192, 388, 1076, 1077, 1087.
idx: 337, 463, 464, 574, 656, 657, 752, 755, 759, 769, 791, 809, 810, 814, 817, 823, 842, 845, 850, 870, 872, 876, 882, 894, 900, 917, 919, 920, 921, 922, 923, 956, 1608, 1829, 1830.
idx_type_map: 463, 464, 752, 753, 762, 763, 919, 1829.
ifstream: 424, 1517, 1549.
iid: 1005, 1006.
idx: 789, 790, 791, 882, 899, 900.
irods_object_id: 999, 1000, 1005, 1006.
irods_object_path: 1005, 1006.
iis_ca: 709, 710, 711, 712.
iis_proxy: 709, 710, 711, 712.
iissuer_cert: 709, 710, 711, 712.
iissuer_cert_id: 709, 710, 711, 712.
immediate: 661, 662, 663, 670, 671, 672, 798, 799, 800, 802, 804, 805, 807, 809, 810, 812, 813, 814, 815, 816, 817, 819, 825, 827, 828.
in_strm: 424, 425, 426, 1517, 1549, 1559, 1560.
INADDR_ANY: 1482, 1500.
incr: 295, 513, 518.
index: 798, 799, 800, 801, 802, 803, 805, 807, 812, 813, 814, 815, 819, 832, 833, 834, 835, 837, 839, 841, 846, 855, 856, 865, 866, 887, 1223, 1224, 1228.
inet_ntop: 1418, 1483, 1501.
INET6_ADDRSTRLEN: 1418.
Initialize_Exception_Type: 4, 1042, 1469, 1483, 1501.
initialize_idx_type_map: 762.
initialize_maps: 20, 21, 56, 57, 60, 301, 762, 763, 1276, 1277, 1804.
initialize_signal_maps: 1756, 1757.
initialize_tls_session: 1401, 1402, 1483, 1488, 1489, 1501.
input_commands: 33, 106.
insert: 156, 203, 209, 239, 241, 243, 248, 343, 359, 367, 386, 445, 552, 894, 978, 1243, 1244, 1290, 1292, 1598, 1757.
insert_str: 1662, 1668.
insert_str_len: 1662, 1668, 1669.
institute_str: 167, 752, 887, 1580, 1581, 1606, 1803, 1811, 1827, 1831.
int: 34, 36, 1536, 1555.
INT_MAX: 865, 1588, 1815.
int_val: 9, 13, 19, 23, 25, 479, 480, 625, 626, 628, 635, 638.
ios_base: 249, 250, 1520.
ip_address: 1415, 1416.
IRDSAVTP_H: 1035.
IRDSOBTP_H: 1259.
irod_object_path: 290.
IRODS BUGS: 101.
irods_auth_filename: 33, 96, 99, 101, 106, 309, 317, 319.
irods_avu: 1009, 1010, 1011, 1012, 1013, 1020, 1021, 1022.
Irods_AVU_TYPE: 474.
Irods_AVU_Type: 151, 331, 380, 381, 387, 467, 474, 994, 996, 997, 998, 999, 1000, 1001, 1002, 1003, 1004, 1006, 1008, 1009, 1010, 1022, 1024, 1026, 1033, 1038, 1047, 1056, 1061, 1072, 1107, 1117, 1136, 1140, 1141, 1151, 1174, 1229, 1230, 1247.
irods_current_dir: 33, 82, 106, 150, 151, 156, 205, 208, 209, 261, 264, 265, 268, 269, 270, 271, 309, 317, 319, 343, 344, 386, 444, 445, 511, 535, 552, 599, 1654.
irods_default_resource: 33, 84, 101, 106, 309, 317, 319, 511, 1639, 1653.
irods_env_filename: 33, 100, 101, 106, 138, 146, 158, 183, 188, 211, 260, 276, 309, 317, 319, 329, 377, 407, 452, 470, 474, 530, 538, 556, 887, 1132, 1133, 1134, 1136, 1140, 1141, 1144, 1149, 1150, 1151, 1153, 1154, 1161, 1231, 1232, 1248.
irods_homedir: 33, 82, 101, 106, 162, 259, 309, 317, 319, 511, 1639, 1653.
Irods_Object: 474.
irods_object: 34, 39, 51, 106, 329, 330, 331, 378, 379, 380, 381.
IRODS_OBJECT_DELETED_FROM_ARCHIVE_INDEX: 752, 753, 763, 1191.
IRODS_OBJECT_DELETED_FROM_GWIRDSIF_DB_INDEX: 752, 753, 763, 1210.
IRODS_OBJECT_DELETED_INDEX: 752.
irods_object_exists: 387, 389, 390, 391, 392, 393, 444, 448, 450, 456, 457, 460.
irods_object_filename: 162, 1080.
irods_object_handle: 290, 444, 463, 465, 467.
irods_object_handle_vector: 384, 400, 403, 404, 409, 410, 411, 412, 413.
irods_object_id: 408, 474, 588, 752, 755, 759, 785, 870, 872, 876, 996, 998, 1000, 1004, 1006, 1008, 1024, 1027, 1028, 1029, 1030, 1031, 1085, 1117.
irods_object_id_vector: 1174, 1199, 1200, 1201.
IRODS_OBJECT_INDEX: 171, 752, 753, 763.
IRODS_OBJECT_MARKED_FOR_DELETION_FROM_ARCHIVE_INDEX: 753, 763, 1169.
IRODS_OBJECT_MARKED_FOR_DELETION_FROM_GWIRDSIF_DB_INDEX: 753, 763, 1169.
irods_object_path: 344, 349, 384, 413, 426, 996, 1004, 1006, 1008, 1013, 1024, 1264, 1271, 1273, 1275, 1306.

IRODS_OBJECT_PID_INDEX: 182, 404, 463, 752, 753, 763.
IRODS_OBJECT_REF_DELETED_FROM_ARCHIVE_INDEX: 753, 763, 1191.
IRODS_OBJECT_REF_DELETED_FROM_GWIRDSIF_DB_INDEX: 753, 763, 1221.
IRODS_OBJECT_REF_INDEX: 403, 405, 463, 752, 753, 763.
Irods_Object_Type: 9, 33, 34, 151, 192, 329, 378, 379, 387, 388, 400, 402, 446, 470, 471, 535, 537, 538, 539, 552, 556, 752, 883, 884, 887, 992, 993, 996, 1034, 1035, 1038, 1039, 1040, 1041, 1042, 1044, 1046, 1057, 1059, 1061, 1063, 1075, 1076, 1077, 1118, 1120, 1131, 1133, 1139, 1141, 1148, 1150, 1152, 1153, 1154, 1158, 1161, 1164, 1172, 1174, 1203, 1204, 1206, 1222, 1223, 1224, 1228, 1230, 1231, 1232, 1236, 1247, 1248, 1256.
irods_object_vector: 34, 535, 536, 537, 538, 539, 552, 553, 554, 555, 556, 887, 1153, 1154, 1155, 1156, 1158, 1161, 1164, 1174, 1201, 1202, 1203, 1231, 1232, 1234, 1235, 1236, 1247, 1248.
Irods_Objects_Handles: 195.
irods_password_encrypted: 33, 80, 97, 309, 317, 319, 511, 1639.
irods_password_encrypted_timestamp: 33, 81, 97, 309, 317, 319, 511.
irods_server_dir: 1766.
irods_server_id: 1038, 1040, 1042, 1044, 1059, 1061, 1089.
irods_zone: 33, 83, 101, 106, 309, 317, 319, 511, 1639, 1653.
IrodsAccess: 38.
is_ca: 702, 706, 708, 710, 712, 719, 731, 733, 744, 746, 1653.
is_file: 1769, 1770, 1775, 1776, 1778.
is_gwirdcli: 235, 521, 1549.
is_gwirdsif: 40, 45, 47, 50, 51, 54, 69, 70, 106, 107, 114, 115, 490.
is_open: 101, 249, 250, 251, 370, 424, 527, 1520, 1559, 1566.
IS_OPTINUM_SRV: 1400.
is_proxy: 702, 706, 708, 710, 712, 720, 731, 733, 744, 746, 1653.
isalnum: 1812.
isdigit: 159, 212, 865, 1743, 1777.
islower: 1812.
isprint: 576, 825, 852.
isspace: 1296, 1777.
issuer_cert: 702, 706, 708, 710, 712, 731, 733, 744.
issuer_cert_id: 702, 706, 708, 710, 712, 718, 731, 733, 744, 746, 1653.
item: 1287, 1288, 1289, 1290.
items_written: 384, 396, 409, 413, 414, 415.
iter: 25, 52, 106, 136, 138, 169, 181, 182, 183, 184, 752, 185, 186, 244, 274, 331, 337, 350, 352, 359, 360, 367, 368, 371, 374, 380, 381, 395, 396, 398, 400, 403, 404, 405, 406, 407, 408, 425, 426, 463, 482, 535, 537, 539, 552, 553, 574, 575, 576, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 645, 646, 763, 816, 817, 818, 821, 822, 823, 824, 825, 844, 845, 848, 849, 850, 851, 852, 894, 898, 916, 917, 918, 919, 920, 921, 922, 923, 945, 953, 970, 1012, 1056, 1061, 1070, 1072, 1117, 1128, 1136, 1151, 1158, 1161, 1164, 1165, 1166, 1167, 1168, 1170, 1176, 1188, 1189, 1190, 1191, 1201, 1203, 1209, 1210, 1220, 1221, 1230, 1236, 1247, 1248, 1257, 1273, 1275, 1277, 1280, 1287, 1288, 1289, 1290, 1307, 1325, 1331, 1449, 1450, 1472, 1485, 1507, 1528, 1536, 1537, 1538, 1539, 1555, 1556, 1557, 1558, 1615, 1631, 1632, 1634, 1646, 1704, 1730, 1828, 1829, 1830, 1832.
iter_1: 396, 646, 1168, 1169, 1170, 1247, 1273, 1280, 1307, 1325, 1646.
iter_2: 404, 406.
iterator: 33, 136, 138, 181, 331, 341, 350, 396, 401, 404, 463, 482, 537, 539, 552, 645, 646, 816, 821, 844, 848, 902, 918, 919, 943, 944, 945, 970, 982, 1012, 1117, 1136, 1151, 1158, 1161, 1164, 1168, 1174, 1176, 1190, 1203, 1206, 1209, 1210, 1220, 1225, 1229, 1230, 1236, 1247, 1248, 1257, 1307, 1449, 1450, 1472, 1485, 1507, 1528, 1536, 1555, 1631, 1646, 1704, 1730, 1832.
j: 423, 1110, 1405, 1740, 1779, 1828.
k: 1406.
key: 1377, 1378, 1379, 1380, 1381, 1382.
keyword: 1349.
Kludge: 228.
kx: 1384.
last_modified: 195, 588, 752, 755, 759, 764, 782, 791, 793, 870, 872, 876, 1038, 1040, 1042, 1044, 1059, 1061, 1069, 1094, 1125, 1126.
last_modified_str: 752, 759, 782, 791, 793, 870, 872, 1038.
last_value: 1263, 1292, 1296.
left: 346, 347, 348, 349, 354, 355, 356, 357, 358, 359, 364, 365, 366, 744, 1277, 1365.
len: 1464, 1465.
length: 274, 275, 277, 280, 281, 284, 285, 286, 484, 486, 487, 876, 1052, 1053, 1054, 1055, 1175, 1295, 1296, 1668, 1669, 1741, 1754.
limit: 1678, 1681, 1683, 1684, 1686, 1687, 1688, 1689, 1694, 1695, 1701, 1702.
line_buffer: 420, 421, 423.
line_num: 1678, 1695, 1700, 1702.

line_num_vector: [1697](#), [1702](#), [1704](#), [1707](#), [1709](#), [1710](#).
listen: [1467](#), [1482](#), [1500](#).
listen_local: [33](#), [1461](#), [1462](#), [1473](#), [1484](#).
listen_local_thread_ctr: [1462](#), [1463](#), [1464](#), [1465](#), [1466](#), [1467](#), [1468](#), [1469](#), [1470](#), [1472](#), [1473](#).
listen_remote_anon: [33](#), [1480](#), [1481](#), [1487](#).
listen_remote_anon_thread_ctr: [1481](#), [1482](#), [1483](#), [1485](#), [1486](#), [1487](#).
listen_remote_X_509: [33](#), [1484](#), [1498](#), [1499](#), [1509](#).
listen_remote_X_509_thread_ctr: [1499](#), [1500](#), [1501](#), [1505](#), [1507](#), [1508](#), [1509](#).
listen_sd: [1482](#), [1483](#), [1485](#), [1487](#), [1500](#), [1501](#), [1507](#).
LISTNLCL_H: [1477](#).
llast_modified: [789](#), [790](#), [791](#).
llast_modified_str: [789](#), [790](#), [791](#).
localityName: [709](#), [710](#), [711](#), [712](#), [1343](#), [1344](#).
local: [1464](#), [1465](#).
local_buffer_size: [1662](#), [1668](#).
local_filename: [9](#), [19](#), [23](#), [25](#), [150](#), [151](#), [153](#), [155](#), [156](#), [157](#), [158](#), [160](#), [161](#), [162](#), [163](#), [168](#), [169](#), [170](#), [171](#), [172](#), [173](#), [175](#), [176](#), [179](#), [181](#), [182](#), [183](#), [184](#), [185](#), [186](#), [188](#), [189](#), [190](#), [191](#), [193](#), [194](#), [195](#), [196](#), [197](#), [204](#), [205](#), [211](#), [233](#), [234](#), [235](#), [236](#), [237](#), [238](#), [239](#), [240](#), [248](#), [249](#), [372](#), [386](#), [387](#), [388](#), [389](#), [390](#), [391](#), [393](#), [394](#), [395](#), [396](#), [397](#), [400](#), [401](#), [403](#), [404](#), [405](#), [406](#), [408](#), [409](#), [410](#), [411](#), [413](#), [414](#), [415](#), [426](#), [462](#), [463](#), [467](#), [471](#), [472](#), [568](#), [569](#), [608](#), [610](#), [616](#), [635](#), [638](#), [1632](#).
LOCAL_HOST: [31](#), [323](#), [547](#), [563](#), [751](#), [886](#), [995](#), [1037](#), [1444](#), [1460](#), [1479](#), [1497](#), [1515](#), [1547](#), [1627](#).
LOCAL_NULL_AUTH_TYPE: [35](#), [36](#), [1469](#).
local_path: [235](#), [240](#), [241](#), [242](#), [247](#), [248](#).
localityName: [702](#), [708](#), [710](#), [712](#), [726](#), [731](#), [733](#), [744](#), [746](#), [1338](#), [1344](#), [1349](#), [1352](#), [1355](#), [1361](#), [1363](#), [1365](#), [1406](#), [1653](#).
localtime_r: [133](#), [1734](#), [1735](#).
lock_cerr_mutex: [39](#), [40](#), [41](#), [45](#), [46](#), [47](#), [49](#), [51](#), [52](#), [54](#), [55](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [70](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [85](#), [86](#), [87](#), [88](#), [89](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#), [100](#), [101](#), [102](#), [110](#), [111](#), [112](#), [113](#), [115](#), [117](#), [118](#), [119](#), [120](#), [121](#), [122](#), [123](#), [124](#), [125](#), [126](#), [127](#), [128](#), [129](#), [136](#), [138](#), [139](#), [140](#), [141](#), [142](#), [145](#), [146](#), [147](#), [148](#), [149](#), [151](#), [152](#), [153](#), [154](#), [155](#), [156](#), [157](#), [158](#), [159](#), [160](#), [161](#), [162](#), [163](#), [164](#), [165](#), [169](#), [170](#), [171](#), [172](#), [173](#), [174](#), [175](#), [176](#), [177](#), [178](#), [179](#), [180](#), [181](#), [182](#), [183](#), [184](#), [185](#), [186](#), [187](#), [188](#), [189](#), [190](#), [191](#), [193](#), [194](#), [195](#), [196](#), [197](#), [199](#), [201](#), [203](#), [204](#), [205](#), [206](#), [207](#), [208](#), [209](#), [210](#), [211](#), [212](#), [213](#), [214](#), [215](#), [218](#), [219](#), [220](#), [221](#), [222](#), [223](#), [224](#), [225](#), [226](#), [227](#), [228](#), [229](#), [230](#), [231](#), [235](#), [237](#), [239](#), [240](#), [241](#), [242](#), [244](#), [245](#), [246](#), [247](#), [248](#), [249](#), [250](#), [251](#), [252](#), [256](#), [257](#), [258](#), [260](#), [261](#), [262](#), [264](#), [265](#), [266](#), [268](#), [269](#), [270](#), [271](#), [272](#), [274](#), [275](#), [276](#), [277](#), [278](#), [280](#), [281](#), [282](#), [284](#), [285](#), [287](#), [288](#), [327](#), [328](#), [329](#), [330](#), [331](#), [332](#), [333](#), [334](#), [335](#), [336](#), [337](#), [339](#), [341](#), [343](#), [344](#), [345](#), [346](#), [347](#), [348](#), [349](#), [350](#), [351](#), [352](#), [354](#), [355](#), [356](#), [357](#), [358](#), [359](#), [360](#), [361](#), [362](#), [363](#), [364](#), [365](#), [366](#), [367](#), [368](#), [369](#), [370](#), [371](#), [372](#), [373](#), [374](#), [375](#), [376](#), [377](#), [379](#), [381](#), [382](#), [384](#), [386](#), [387](#), [388](#), [389](#), [390](#), [391](#), [392](#), [393](#), [394](#), [395](#), [396](#), [397](#), [398](#), [400](#), [401](#), [403](#), [404](#), [405](#), [406](#), [407](#), [408](#), [409](#), [410](#), [411](#), [413](#), [414](#), [415](#), [426](#), [460](#), [461](#), [462](#), [463](#), [465](#), [466](#), [467](#), [468](#), [469](#), [470](#), [471](#), [472](#), [473](#), [474](#), [475](#), [476](#), [477](#), [478](#), [479](#), [480](#), [481](#), [482](#), [484](#), [485](#), [486](#), [487](#), [488](#), [490](#), [491](#), [492](#), [493](#), [494](#), [495](#), [496](#), [498](#), [499](#), [500](#), [501](#), [502](#), [503](#), [504](#), [505](#), [506](#), [507](#), [508](#), [509](#), [510](#), [513](#), [514](#), [515](#), [516](#), [517](#), [518](#), [519](#), [520](#), [521](#), [522](#), [523](#), [524](#), [525](#), [526](#), [527](#), [528](#), [530](#), [531](#), [532](#), [534](#), [535](#), [536](#), [537](#), [538](#), [540](#), [541](#), [550](#), [552](#), [553](#), [554](#), [555](#), [556](#), [557](#), [565](#), [566](#), [567](#), [568](#), [569](#), [570](#), [573](#), [574](#), [576](#), [577](#), [578](#), [579](#), [580](#), [582](#), [583](#), [584](#), [585](#), [586](#), [587](#), [588](#), [589](#), [590](#), [591](#), [592](#), [593](#), [594](#), [595](#), [596](#), [597](#), [598](#), [599](#), [600](#), [601](#), [602](#), [603](#), [604](#), [605](#), [606](#), [607](#), [608](#), [609](#), [612](#), [613](#), [614](#), [615](#), [616](#), [617](#), [618](#), [620](#), [621](#), [622](#), [623](#), [624](#), [626](#), [627](#), [628](#), [629](#), [630](#), [631](#), [632](#), [633](#), [634](#), [635](#), [636](#), [637](#), [638](#), [639](#), [640](#), [641](#), [642](#), [643](#), [644](#), [645](#), [646](#), [647](#), [648](#), [649](#), [650](#), [651](#), [652](#), [653](#), [654](#), [655](#), [656](#), [657](#), [658](#), [659](#), [660](#), [662](#), [663](#), [664](#), [665](#), [667](#), [668](#), [670](#), [671](#), [673](#), [674](#), [675](#), [677](#), [678](#), [679](#), [680](#), [681](#), [682](#), [683](#), [684](#), [685](#), [686](#), [687](#), [688](#), [689](#), [690](#), [691](#), [692](#), [693](#), [694](#), [695](#), [696](#), [697](#), [714](#), [715](#), [716](#), [717](#), [718](#), [719](#), [720](#), [721](#), [722](#), [723](#), [724](#), [725](#), [726](#), [727](#), [728](#), [729](#), [735](#), [736](#), [737](#), [738](#), [739](#), [740](#), [741](#), [742](#), [757](#), [759](#), [761](#), [763](#), [765](#), [771](#), [781](#), [782](#), [783](#), [784](#), [785](#), [786](#), [788](#), [790](#), [791](#), [792](#), [793](#), [795](#), [797](#), [798](#), [799](#), [800](#), [801](#), [802](#), [803](#), [804](#), [805](#), [806](#), [807](#), [808](#), [809](#), [810](#), [812](#), [813](#), [814](#), [815](#), [816](#), [817](#), [818](#), [819](#), [820](#), [826](#), [827](#), [828](#), [829](#), [831](#), [833](#), [834](#), [835](#), [837](#), [838](#), [839](#), [840](#), [841](#), [844](#), [845](#), [846](#), [847](#), [853](#), [854](#), [856](#), [857](#), [860](#), [861](#), [862](#), [864](#), [865](#), [866](#), [867](#), [868](#), [874](#), [875](#), [876](#), [877](#), [878](#), [879](#), [880](#), [881](#), [892](#), [894](#), [900](#), [902](#), [903](#), [904](#), [905](#), [906](#), [907](#), [908](#), [909](#), [910](#), [911](#), [912](#), [913](#), [914](#), [915](#), [916](#), [917](#), [919](#), [920](#), [921](#), [922](#), [924](#), [925](#), [927](#), [928](#), [929](#), [930](#), [931](#), [932](#), [933](#), [934](#), [935](#), [937](#), [938](#), [939](#), [940](#), [941](#), [942](#), [944](#), [946](#), [949](#), [952](#), [953](#), [954](#), [955](#), [956](#), [957](#), [960](#), [961](#), [962](#), [963](#), [964](#), [966](#), [967](#), [968](#), [969](#), [970](#), [971](#), [972](#), [973](#),

974, 975, 976, 978, 979, 980, 982, 983, 984,
 985, 986, 987, 988, 989, 990, 991, 1006, 1010,
 1011, 1012, 1013, 1014, 1015, 1016, 1017, 1018,
 1020, 1021, 1022, 1026, 1027, 1028, 1029, 1030,
 1031, 1032, 1033, 1042, 1044, 1046, 1048, 1049,
 1050, 1051, 1052, 1053, 1054, 1055, 1056, 1057,
 1063, 1065, 1066, 1067, 1068, 1069, 1070, 1071,
 1072, 1073, 1074, 1075, 1077, 1079, 1080, 1081,
 1082, 1083, 1084, 1085, 1086, 1087, 1088, 1089,
 1090, 1091, 1092, 1093, 1094, 1095, 1096, 1098,
 1099, 1100, 1101, 1102, 1103, 1104, 1105, 1106,
 1107, 1108, 1109, 1110, 1111, 1112, 1113, 1114,
 1115, 1116, 1117, 1118, 1120, 1121, 1123, 1124,
 1125, 1126, 1127, 1128, 1129, 1130, 1131, 1133,
 1134, 1135, 1136, 1137, 1138, 1139, 1141, 1142,
 1143, 1144, 1145, 1147, 1148, 1150, 1151, 1152,
 1154, 1155, 1156, 1157, 1158, 1160, 1161, 1162,
 1163, 1164, 1165, 1166, 1167, 1168, 1169, 1170,
 1171, 1172, 1174, 1175, 1176, 1177, 1178, 1179,
 1180, 1181, 1182, 1183, 1184, 1185, 1186, 1188,
 1189, 1191, 1193, 1194, 1195, 1196, 1197, 1198,
 1199, 1200, 1201, 1202, 1203, 1204, 1206, 1207,
 1208, 1209, 1210, 1211, 1212, 1213, 1214, 1215,
 1216, 1217, 1218, 1219, 1220, 1221, 1222, 1225,
 1226, 1227, 1228, 1230, 1232, 1233, 1234, 1235,
 1236, 1237, 1239, 1241, 1242, 1243, 1244, 1245,
 1246, 1247, 1248, 1249, 1250, 1251, 1252, 1253,
 1254, 1255, 1273, 1277, 1279, 1280, 1281, 1283,
 1284, 1285, 1287, 1288, 1289, 1290, 1292, 1293,
 1296, 1298, 1299, 1300, 1301, 1302, 1303, 1304,
 1305, 1306, 1307, 1308, 1309, 1310, 1311, 1312,
 1314, 1315, 1316, 1317, 1318, 1325, 1327, 1346,
 1348, 1349, 1398, 1400, 1404, 1409, 1428, 1429,
 1430, 1431, 1432, 1433, 1434, 1435, 1437, 1438,
 1439, 1440, 1446, 1447, 1449, 1450, 1451, 1452,
 1453, 1455, 1463, 1464, 1465, 1466, 1467, 1468,
 1469, 1470, 1472, 1473, 1481, 1482, 1483, 1485,
 1486, 1487, 1499, 1500, 1501, 1502, 1503, 1505,
 1507, 1508, 1509, 1518, 1519, 1520, 1521, 1522,
 1523, 1524, 1526, 1527, 1528, 1529, 1530, 1531,
 1532, 1533, 1534, 1537, 1539, 1540, 1549, 1550,
 1551, 1552, 1553, 1556, 1558, 1559, 1560, 1561,
 1562, 1563, 1564, 1566, 1567, 1569, 1570, 1571,
 1581, 1583, 1584, 1585, 1586, 1587, 1588, 1589,
 1590, 1591, 1592, 1593, 1594, 1595, 1596, 1597,
 1598, 1599, 1600, 1601, 1602, 1603, 1606, 1607,
 1609, 1610, 1611, 1612, 1613, 1614, 1615, 1616,
 1617, 1618, 1620, 1621, 1622, 1623, 1629, 1630,
 1631, 1632, 1634, 1635, 1637, 1639, 1641, 1643,
 1645, 1646, 1647, 1648, 1649, 1651, 1655, 1656,
 1662, 1663, 1664, 1665, 1666, 1667, 1668, 1669,
 1670, 1671, 1677, 1678, 1680, 1681, 1682, 1683,
 1684, 1685, 1686, 1687, 1688, 1689, 1691, 1692,
 1716, 1718, 1724, 1725, 1726, 1727, 1729, 1730,
 1731, 1732, 1733, 1735, 1737, 1739, 1740, 1741,
 1742, 1743, 1744, 1745, 1746, 1747, 1750, 1751,
 1753, 1754, 1755, 1759, 1761, 1763, 1764, 1765,
 1766, 1767, 1768, 1770, 1771, 1772, 1773, 1774,
 1775, 1776, 1777, 1778, 1779, 1780, 1782, 1783,
 1785, 1790, 1802, 1805, 1806, 1807, 1811, 1812,
 1813, 1814, 1815, 1816, 1817, 1818, 1819, 1820,
 1821, 1822, 1823, 1824, 1825, 1826, 1827, 1828,
 1829, 1830, 1831, 1832, 1833, 1834.
lock_cout_mutex: 525, 1453, 1468, 1470, 1483,
 1500, 1501, 1551, 1721, 1831.
lock_table: 965, 966, 968, 973, 977, 978.
log_dir: 1688, 1689, 1707.
log_filename: 1686.
log_strm_mutex: 1686.
LONG_MAX: 268, 284, 390, 784, 916, 1587, 1766,
 1814, 1825.
LONG_MIN: 268, 284, 390, 784, 916, 1587, 1766,
 1814, 1825.
ls: 135, 136, 142, 389, 593.
LS_TYPE: 9, 10, 57, 58.
LSTNRMTA_H: 1495.
LSTNRMTX_H: 1513.
lsUtil: 137.
main: 33, 752, 887, 1461, 1769, 1791, 1797,
 1803, 1834.
make_pair: 359, 367, 894, 1290, 1292, 1757.
malloc: 1818.
map: 9, 10, 33, 34, 36, 106, 310, 311, 574, 752,
 753, 763, 887, 894, 898, 902, 916, 919, 943, 944,
 945, 982, 1012, 1164, 1176, 1206, 1225, 1264,
 1265, 1277, 1287, 1449, 1450, 1452, 1453, 1472,
 1485, 1507, 1536, 1555, 1631, 1646, 1828.
map_iter: 902, 919, 920.
mark_for_deletion: 538, 887, 1153, 1154, 1172.
mark_irods_objects_for_deletion: 299, 530, 541,
 605.
MARK_IRODS_OBJECTS_FOR_DELETION_TYPE: 9,
 10, 57, 58.
marked_for_deletion: 588, 668, 752, 755, 759, 780,
 791, 817, 822, 845, 849, 870, 872, 876, 887.
marked_for_deletion_from_archive: 535, 1038,
 1040, 1042, 1044, 1059, 1061, 1090, 1151,
 1161, 1236.
marked_for_deletion_from_gwirdsif_database: 1161.
marked_for_deletion_from_gwirdsif_db: 535, 1038,
 1040, 1042, 1044, 1059, 1061, 1091, 1236.
matched: 396, 399, 415.
MAX_RESPONSE_TYPE: 9, 10, 57, 58, 59, 60.
MAX_SESSION_DATA_SIZE: 1370, 1371, 1378.

MAX_SESSION_ID_SIZE: [1370](#), 1371, 1378.
memcmp: 1380, 1382.
memcpy: 575, 759, 771, 775, 791, 1296, 1378, 1380.
memory leaks (fixed): 1240.
memset: 39, 45, 95, 99, 126, 137, 139, 145, 146, 158, 202, 211, 223, 227, 230, 233, 251, 259, 263, 279, 389, 420, 423, 425, 444, 456, 478, 481, 485, 486, 520, 526, 528, 565, 568, 569, 573, 574, 589, 592, 595, 598, 601, 604, 607, 614, 618, 624, 631, 634, 637, 641, 644, 647, 649, 652, 655, 660, 675, 684, 689, 694, 697, 765, 771, 775, 781, 782, 786, 792, 793, 952, 956, 1010, 1015, 1024, 1047, 1049, 1133, 1141, 1163, 1174, 1178, 1249, 1296, 1416, 1446, 1482, 1499, 1500, 1502, 1517, 1518, 1520, 1525, 1526, 1530, 1531, 1539, 1549, 1551, 1558, 1560, 1561, 1562, 1563, 1668, 1681, 1682, 1683, 1685, 1697, 1700, 1703, 1732, 1735, 1737, 1764, 1770, 1771, 1775, 1776, 1777, 1778, 1779, 1780, 1783, 1790, 1802.
message_str: [1389](#).
metadata_handle_id_vector: [151](#), 176, 177, 178, 179, 181.
metadata_handle_vector: [151](#), 179, 181, 186.
metadata_id: 352, 1307, [1319](#), 1321, 1329, 1331.
metadata_options: [9](#), 13, 19, 23, 25, 385, 393.
metadata_sub_map: 359, 367, [1264](#), 1271, 1273, 1275, 1279, 1280, 1292, 1307.
metadata_sub_stack: 1264, 1290, 1292.
min_offset: [1731](#), [1732](#).
minus_spec: [1746](#).
mkdir: 234, 245, [273](#), [274](#), 288, 602.
MKDIR_TYPE: [9](#), [10](#), 57, 58.
mkstemp: 96, 100, 204, 250, 369, 485, 527, 589, 1520, 1566.
mktime: 1055.
mlock: 95, 1771, 1772, 1783.
mmarked_for_deletion: [789](#), [790](#), 791.
multimap: 341, 1264, 1273, 1275, 1280, 1307, 1319, 1325, 1331.
munlock: 99, 1775, 1776, 1777, 1778, 1779, 1780, 1790, 1802.
mutex: [1694](#), [1695](#), 1697, 1699, 1705, 1706, 1708, 1709, 1710, 1711.
mutex_1_ptr: [1694](#), [1695](#), 1696, 1697, 1698, 1699, 1700, 1701, 1702, 1703, 1704, 1705, 1706, 1707, 1708, 1709, 1710, 1711.
my_bool: 44.
MYSQL: 33, 107, 295, 513, 734, 735, 752, 796, 797, 830, 831, 873, 874, 887, 899, 900, 901, 902, 926, 927, 951, 952, 959, 960, [965](#), [966](#), 977, 978, 981, 982, 1009, 1010, 1025, 1026, 1062, 1063, 1076, 1077, 1119, 1120, 1140, 1141, 1149, 1150, 1153, 1154, 1173, 1174, 1205, 1206, 1223, 1224, 1231, 1232, 1297, 1298, 1313, 1314, 1580, 1581, 1619, 1620, 1661, 1662, 1723, 1724, 1728, 1729, 1791, 1803.
mysql_affected_rows: 1725.
mysql_close: 41, 46, 47, 50, 51, 1794, 1795, 1796, 1797, 1807, 1811, 1812, 1814, 1815, 1818, 1821, 1822, 1823, 1825, 1831, 1832, 1834.
mysql_errno: 46, 64, 1724.
mysql_error: 46, 64, 67, 73, 76, 111, 119, 124, 173, 175, 328, 334, 336, 344, 345, 350, 352, 360, 363, 434, 437, 480, 481, 493, 496, 516, 740, 804, 807, 815, 819, 838, 841, 846, 875, 877, 879, 903, 912, 916, 953, 956, 1065, 1079, 1081, 1084, 1100, 1103, 1107, 1110, 1123, 1184, 1198, 1216, 1300, 1301, 1303, 1306, 1307, 1308, 1309, 1315, 1316, 1317, 1583, 1584, 1585, 1589, 1591, 1592, 1601, 1609, 1610, 1611, 1623, 1663, 1665, 1671, 1724, 1794, 1807, 1823.
mysql_fetch_row: 67, 76, 119, 124, 175, 336, 345, 352, 363, 437, 481, 496, 516, 740, 807, 841, 912, 916, 956, 1083, 1084, 1103, 1110, 1184, 1198, 1216, 1243, 1244, 1303, 1585, 1591, 1592, 1601, 1623, 1665, 1823.
mysql_free_result: 64, 65, 66, 67, 73, 74, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 102, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 173, 175, 176, 177, 334, 335, 336, 337, 344, 345, 346, 347, 348, 349, 350, 351, 352, 354, 355, 357, 359, 360, 361, 363, 364, 367, 434, 435, 437, 438, 455, 480, 481, 483, 485, 486, 493, 496, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 515, 516, 517, 518, 519, 736, 738, 740, 741, 742, 804, 805, 807, 809, 810, 811, 812, 813, 815, 819, 820, 838, 839, 841, 843, 846, 847, 875, 877, 878, 879, 904, 906, 907, 908, 909, 910, 912, 913, 914, 916, 922, 924, 934, 937, 938, 939, 953, 954, 956, 968, 971, 972, 973, 986, 987, 988, 989, 1021, 1028, 1029, 1030, 1031, 1032, 1066, 1067, 1068, 1069, 1071, 1073, 1074, 1081, 1083, 1084, 1085, 1086, 1087, 1089, 1090, 1091, 1092, 1093, 1094, 1095, 1096, 1098, 1099, 1100, 1102, 1103, 1104, 1105, 1107, 1109, 1110, 1111, 1112, 1113, 1114, 1115, 1116, 1117, 1124, 1127, 1129, 1130, 1159, 1160, 1182, 1183, 1184, 1185, 1187, 1194, 1196, 1197, 1198, 1199, 1204, 1209, 1211, 1212, 1213, 1214, 1215, 1216, 1217, 1222, 1241, 1242, 1243, 1244, 1300, 1301, 1302, 1303, 1304, 1305, 1306, 1307, 1308, 1309, 1310, 1311, 1315, 1316, 1317, 1583, 1585, 1587, 1588, 1589, 1591, 1592, 1601, 1602, 1603, 1609, 1610, 1611, 1622, 1623, 1663, 1664, 1665, 1666, 1668, 1669,

1670, 1671, 1730, 1821, 1822, 1823, 1825.
mysql_init: 40, 1793, 1806.
MYSQL_INIT_COMMAND: 44.
mysql_library_end: 1793, 1794, 1795, 1796, 1797, 1806, 1807, 1811, 1812, 1814, 1815, 1818, 1821, 1822, 1823, 1825, 1831, 1832, 1834.
mysql_library_init: 1792, 1805.
mysql_num_fields: 1724.
mysql_num_rows: 1724.
MYSQL_OPT_CONNECT_TIMEOUT: 44.
MYSQL_OPT_RECONNECT: 44.
mysql_options: 44.
mysql_password: 45, 46, 1790, 1802.
MYSQL_PASSWORD_LENGTH: 45, 1790, 1802.
mysql_ptr: 33, 40, 41, 44, 46, 47, 50, 51, 64, 67, 73, 76, 92, 107, 108, 111, 119, 121, 124, 167, 171, 173, 175, 181, 182, 192, 194, 195, 295, 328, 334, 336, 344, 345, 350, 352, 360, 363, 388, 396, 398, 401, 403, 404, 405, 406, 408, 434, 437, 441, 446, 450, 461, 463, 465, 466, 468, 469, 471, 476, 480, 481, 493, 496, 513, 515, 516, 535, 538, 553, 556, 653, 657, 669, 679, 686, 691, 734, 735, 736, 740, 752, 796, 797, 804, 807, 815, 819, 830, 831, 838, 841, 846, 873, 874, 875, 877, 879, 887, 899, 900, 901, 902, 903, 904, 906, 909, 912, 913, 916, 922, 924, 926, 927, 932, 934, 937, 939, 951, 952, 953, 956, 959, 960, 961, 965, 966, 968, 971, 973, 977, 978, 981, 982, 986, 987, 988, 989, 1009, 1010, 1021, 1025, 1026, 1028, 1029, 1030, 1031, 1062, 1063, 1065, 1066, 1067, 1068, 1069, 1071, 1073, 1074, 1076, 1077, 1079, 1080, 1081, 1084, 1099, 1100, 1103, 1107, 1110, 1119, 1120, 1123, 1124, 1127, 1129, 1130, 1140, 1141, 1142, 1147, 1149, 1150, 1151, 1153, 1154, 1159, 1166, 1169, 1170, 1173, 1174, 1182, 1183, 1184, 1191, 1194, 1196, 1198, 1201, 1203, 1205, 1206, 1208, 1209, 1210, 1211, 1212, 1213, 1214, 1216, 1219, 1221, 1223, 1224, 1228, 1231, 1232, 1240, 1245, 1297, 1298, 1299, 1300, 1301, 1303, 1306, 1307, 1308, 1309, 1310, 1313, 1314, 1315, 1316, 1317, 1580, 1581, 1583, 1584, 1585, 1591, 1592, 1597, 1600, 1601, 1603, 1609, 1610, 1611, 1619, 1620, 1621, 1623, 1661, 1662, 1663, 1664, 1665, 1666, 1668, 1669, 1670, 1671, 1723, 1724, 1725, 1728, 1729, 1730, 1791, 1793, 1794, 1795, 1796, 1797, 1803, 1806, 1807, 1811, 1812, 1814, 1815, 1818, 1821, 1822, 1823, 1825, 1831, 1832, 1834.
mysql_query: 1724.
mysql_real_connect: 38, 46, 1794, 1807.
MYSQL_RES: 63, 73, 107, 108, 109, 110, 116, 151, 327, 344, 434, 444, 478, 492, 513, 735, 798, 832, 874, 902, 927, 952, 966, 982, 1010, 1027, 1064,
1078, 1122, 1154, 1174, 1206, 1232, 1298, 1314, 1582, 1620, 1662, 1723, 1724, 1728, 1729, 1803.
MYSQL_ROW: 63, 73, 116, 151, 336, 344, 434, 478, 492, 513, 713, 714, 735, 764, 765, 798, 832, 902, 952, 966, 982, 1010, 1027, 1064, 1078, 1122, 1154, 1174, 1206, 1232, 1298, 1314, 1582, 1620, 1662, 1803.
mysql_select_db: 38, 47, 328, 903, 1065, 1079, 1123, 1299, 1583, 1795.
mysql_socket_filename: 41, 46.
mysql_store_result: 1724.
mysql_username: 45, 46.
name: 1038.
NAME_LEN: 31, 323, 547, 563, 751, 886, 995, 1037, 1444, 1460, 1479, 1497, 1515, 1547, 1627.
nanosleep: 228.
new_attrib: 1149, 1150, 1151.
new_flags: 276, 534.
new_local_filename: 1630.
new_local_filename_ptr: 234, 235, 248.
new_response: 151, 153, 157, 158, 160, 161, 163, 168, 169, 171, 172, 173, 175, 176, 179, 181, 182, 184, 185, 186, 189, 190, 191, 194, 195, 196, 197, 384, 388, 390, 391, 393, 394, 395, 396, 397, 400, 403, 404, 405, 406, 408, 409, 410, 411, 413, 414, 415, 444, 462, 533, 535, 539, 550, 551, 552, 553, 555, 556, 685, 690.
new_type_str: 887, 1223, 1224, 1228.
new_val: 1149, 1150, 1151.
no_delay: 9, 13, 19, 23, 25, 646.
NOTE: 69, 73, 101, 203, 239, 472, 1161, 1484, 1609, 1640, 1723, 1760, 1788.
now: 1599, 1681, 1686, 1687, 1688, 1689, 1694, 1695, 1697.
npos: 152, 166, 203, 206, 238, 240, 243, 247, 259, 267, 276, 283, 343, 386, 534, 535, 552, 857, 861, 1288, 1347, 1349, 1594, 1598, 1740, 1743, 1766, 1782.
ntohs: 1483, 1501.
NULL_AUTH_TYPE: 35, 36.
NULL_HANDLE_VALUE_TYPE_INDEX: 752, 753, 763.
NULL_RESPONSE_TYPE: 9, 10, 13, 23.
number_of_pids: 752, 887, 1580, 1581, 1603, 1605, 1812, 1827, 1831.
O_NONBLOCK: 1614.
O_WRONLY: 1614.
oct: 25, 164, 512, 797.
ofstream: 101, 126, 201, 234, 235, 251, 369, 520, 573, 1278, 1279, 1517, 1549.
oid: 1390, 1406.
oid_str: 1404, 1406.
old_cancel_state: 1723, 1724, 1727.

old_filename: 1705, 1706, 1707.
old_pthread_cancel_state: 275, 277, 280, 281, 284, 285, 287.
old_type_str: 887, 1223, 1224, 1225.
oorganization: 709, 710, 711, 712, 1343, 1344.
oorganizationalUnitName: 709, 710, 711, 712, 1343, 1344.
open: 101, 126, 234, 249, 250, 370, 424, 527, 589, 1520, 1559, 1566, 1614.
optimization: 1716.
option_ctr: 550, 551, 552, 553, 555, 556.
options: 9, 13, 19, 23, 25, 253, 340, 341, 344, 345, 346, 347, 348, 350, 351, 352, 354, 355, 357, 360, 363, 364, 369, 370, 371, 372, 378, 380, 381, 395, 530, 531, 532, 535, 539, 550, 685, 686, 796, 797, 798, 926, 927, 928, 931, 933, 935, 940, 1158, 1161, 1169, 1170.
optpsgen_path: 1662.
optval: 1482, 1500.
or_str: 1582, 1606.
organization: 63, 702, 708, 710, 712, 722, 731, 733, 744, 746, 1338, 1344, 1349, 1352, 1355, 1361, 1363, 1365, 1406, 1653.
organizationalUnitName: 63, 702, 708, 710, 712, 723, 731, 733, 744, 746, 1338, 1344, 1349, 1352, 1355, 1361, 1363, 1365, 1406, 1653.
ostream: 297, 512, 1694, 1695.
OTHER_HANDLE_VALUE_TYPE: 919.
OTHER_HANDLE_VALUE_TYPE_INDEX: 752, 753, 763, 919, 1828.
other_type_ctr: 1828, 1829.
out: 249, 250, 1391, 1520.
out_strm: 126, 130, 131, 132, 133, 201, 235, 249, 250, 251, 369, 370, 371, 372, 520, 526, 527, 528, 573, 589, 590, 1278, 1279, 1280, 1517, 1520, 1522, 1549, 1566, 1694, 1695, 1696, 1697, 1698, 1699, 1700, 1701, 1702, 1703, 1704, 1705, 1706, 1707, 1708, 1709, 1710, 1711.
out_strm_filename: 1517, 1520, 1522, 1549, 1566, 1567.
output: 171, 371, 482, 745, 746, 1278, 1279, 1281, 1362, 1363.
output_func: 1549, 1552, 1553, 1559, 1560, 1561, 1562, 1563, 1564, 1565, 1566, 1576.
output_str: 267, 269, 271, 283, 285.
outstr: 744, 1042, 1044, 1122, 1125, 1436, 1437, 1438, 1732, 1733, 1735.
overwrite: 203, 213, 234, 235, 239.
owner: 666.
OWNER_INDEX: 752, 753, 762, 763.
p: 1416.
param: 9, 33, 637, 702, 752, 887, 1012, 1045, 1046, 1176, 1189, 1403, 1404, 1447, 1448, 1450, 1451, 1452, 1453, 1468, 1469, 1470, 1471, 1472, 1473, 1483, 1484, 1485, 1486, 1500, 1501, 1502, 1503, 1505, 1506, 1507, 1508, 1516, 1517, 1518, 1519, 1520, 1521, 1522, 1523, 1524, 1526, 1527, 1528, 1529, 1530, 1531, 1532, 1533, 1534, 1536, 1538, 1539, 1540, 1548, 1549, 1550, 1551, 1552, 1553, 1554, 1555, 1557, 1558, 1559, 1560, 1561, 1562, 1563, 1564, 1565, 1566, 1567, 1569, 1570, 1571, 1628, 1629, 1631, 1632, 1634, 1636, 1637, 1639, 1640, 1641, 1642, 1644, 1645, 1646, 1647, 1648, 1649, 1651, 1652, 1654, 1655.
parameter: 9, 32, 33, 309, 324, 548, 1264, 1319, 1517, 1522, 1524, 1541, 1549, 1567, 1568, 1569, 1570, 1571, 1573.
parse: 426, 1282, 1283, 1285.
parse_metadata: 291, 394, 418, 427.
parse_post_data: 33.
PARSER_DEBUG: 33, 39, 1447.
passphrase: 1769, 1770, 1772, 1781, 1782.
passwd: 1818.
password: 1781, 1782, 1783.
password_length: 1781, 1782, 1783.
path: 329, 378, 400, 405, 407, 408, 474, 475, 535, 539, 887, 1038, 1042, 1044, 1059, 1061, 1069, 1080, 1081, 1097, 1134, 1136, 1144, 1161, 1173, 1175, 1177, 1183, 1191, 1196, 1203, 1210, 1214, 1221, 1223, 1224, 1226, 1227, 1228, 1236, 1248.
pclose: 98, 126, 139, 140, 141, 142, 147, 148, 149, 158, 160, 161, 211, 213, 214, 264, 265, 268, 269, 280, 281, 284, 285, 420, 453, 455, 456, 1015, 1016, 1018, 1019, 1049, 1050, 1056, 1162, 1163, 1178, 1179, 1249, 1250, 1251, 1668, 1685, 1703, 1764, 1765, 1776.
pending_operations_flag: 33, 106.
pending_operations_iter: 33.
pending_response_deque: 33.
perror: 146, 1416, 1494, 1512, 1521.
pid: 1760, 1761, 1766.
pid_ctr: 1582, 1602, 1603, 1605, 1606.
pid_institute_str: 9, 19, 23, 25, 33, 164, 167, 653.
pid_options: 9, 13, 19, 23, 25, 164, 165, 191, 661, 669.
pid_prefix_str: 9, 19, 23, 25, 33, 164, 165, 653.
pid_str: 9, 19, 23, 25, 33, 164, 165, 166, 167, 169, 183, 188, 189, 190, 191, 638, 653, 656, 657, 662, 663, 664, 665, 667, 668, 670, 671, 672, 677, 678, 679, 680, 681, 752, 887, 1580, 1581, 1593, 1594, 1596, 1598, 1599, 1606, 1607, 1616, 1803, 1809, 1827, 1831.
pid_suffix: 33.
pid_suffix_str: 9, 19, 23, 25, 33, 164, 167, 653.

pid_vector: 1831.
pid_vector_ptr: 752, 887, 1580, 1581, 1599, 1606, 1615.
 PIDFNCS_H: 1625.
plain_text: 1769, 1770, 1779.
plain_text_length: 1769, 1770, 1779.
plus_spec: 1746.
pop: 1292.
pop_front: 1534, 1554.
popen: 97, 126, 138, 146, 158, 161, 211, 214, 260, 276, 419, 452, 1014, 1015, 1048, 1162, 1177, 1178, 1248, 1668, 1678, 1682, 1699, 1762, 1763, 1775.
 PORT: 1415.
port_num_anon: 1482, 1483.
port_num_x_509: 1500.
port_str: 1415, 1416.
pos: 152, 154, 166, 201, 203, 206, 235, 238, 240, 243, 247, 256, 259, 267, 274, 276, 283, 343, 384, 386, 530, 534, 535, 552, 856, 857, 859, 860, 861, 863, 864, 1047, 1052, 1053, 1054, 1055, 1287, 1288, 1582, 1593, 1594, 1598, 1740, 1742, 1743, 1763, 1766, 1782.
 POST_HANDLES: 1208, 1211.
ppath: 1041, 1042, 1043, 1044.
ppub_read: 789, 790, 791.
ppub_write: 789, 790, 791.
prefix: 798, 799, 800, 801, 803, 832, 833, 834, 835, 837, 855, 856, 860, 867.
prefix_id: 1597, 1619, 1620, 1623.
prefix_str: 165, 166, 167, 752, 887, 1580, 1581, 1594, 1595, 1596, 1597, 1598, 1599, 1600, 1603, 1604, 1606, 1619, 1620, 1621, 1622, 1623, 1803, 1810, 1827, 1831.
prev_handle_id: 952, 956.
prev_map_iter: 902.
print: 1387.
print_info: 1383, 1384, 1505.
print_x509_certificate_info: 1384, 1388, 1389, 1395.
printable: 1387.
privileges: 33, 39, 91, 106, 297, 309, 315, 317, 319, 479, 491, 494, 510, 511, 512, 662, 665, 684, 685, 686, 690, 691, 796, 797, 802, 809, 810, 814, 815, 830, 831, 842, 1639, 1641, 1647, 1649, 1653, 1655.
privs: 104, 490, 494, 510.
 PROCESS_PENDING_TYPE: 9, 10, 57, 58.
 PRSRFNCS_H: 1658.
pthread_attr_destroy: 1509.
pthread_attr_init: 1464, 1483, 1500.
pthread_attr_setdetachstate: 1464, 1483, 1500.
pthread_attr_t: 1464, 1483, 1500.
pthread_cancel: 1450.
 PTHREAD_CANCEL_DISABLE: 106, 275, 1680, 1724.
 PTHREAD_CANCEL_ENABLE: 34, 38, 39, 106, 1691.
pthread_cleanup_pop: 1451, 1453.
pthread_cleanup_push: 1447, 1449.
pthread_cond_signal: 540, 827, 940, 1171.
pthread_create: 1473, 1486, 1508.
 PTHREAD_CREATE_DETACHED: 1464, 1483, 1500.
pthread_exit: 38, 40, 41, 46, 47, 1450, 1451, 1452, 1453, 1463, 1464, 1465, 1466, 1467, 1469, 1472, 1473, 1482, 1483, 1485, 1487, 1500, 1501, 1507, 1509, 1681, 1682, 1686, 1687, 1688, 1689, 1693.
pthread_kill: 1450, 1451.
pthread_mutex_destroy: 53, 1790, 1802.
pthread_mutex_init: 49, 1791, 1803.
pthread_mutex_lock: 540, 827, 940, 1012, 1171, 1176, 1449, 1452, 1453, 1471, 1484, 1504, 1506, 1631, 1634, 1643, 1650, 1695, 1696, 1697, 1698, 1699, 1700, 1701, 1702, 1703, 1704, 1705, 1706, 1707, 1708, 1709, 1710, 1711, 1724.
pthread_mutex_t: 33, 310, 311, 1694, 1695, 1723.
pthread_mutex_unlock: 540, 827, 940, 1012, 1171, 1176, 1450, 1452, 1453, 1472, 1485, 1504, 1507, 1631, 1632, 1634, 1643, 1650, 1695, 1696, 1697, 1698, 1699, 1700, 1701, 1702, 1703, 1704, 1705, 1706, 1707, 1708, 1709, 1710, 1711, 1724, 1726.
pthread_self: 1450, 1452, 1453.
pthread_setcancelstate: 275, 277, 280, 281, 284, 285, 287, 1680, 1691, 1723, 1724, 1727.
pthread_t: 540, 940, 1171, 1449, 1450, 1452, 1453, 1468, 1472, 1483, 1485, 1501, 1507.
pub_read: 588, 752, 759, 778, 791, 870, 872, 876.
pub_write: 588, 752, 759, 779, 791, 870, 872, 876.
public_key_id: 33, 34, 89, 106, 126, 309, 317, 319.
purge_database_interval: 827, 828, 935, 1157.
purge_database_limit: 685, 798, 799, 800, 802, 804, 805, 807, 809, 810, 812, 813, 814, 815, 817, 819, 825, 935.
purge_database_thread_id: 940.
purge_irods_archive: 1038, 1161, 1173.
purge_irods_archive_cond: 540, 1171.
purge_irods_archive_limit: 1151, 1158.
purge_irods_archive_mutex: 540, 1171.
purge_irods_archive_thread_id: 540, 1171.
purge_limit: 1678.
purge_logs_limit: 1678.
purge_server_database: 1157.
purge_server_database_cond: 827, 940.
purge_server_database_mutex: 827, 940.
purge_server_logs: 97, 1676, 1677, 1693.
 PURGE_SERVER_LOGS_SLEEP: 1685, 1690.

PURGFNCS_H: [1713](#).
push: 1290.
push_back: 96, 100, 153, 157, 158, 160, 161, 163, 168, 169, 171, 172, 173, 175, 176, 179, 181, 182, 184, 185, 186, 189, 190, 191, 194, 195, 196, 197, 204, 211, 213, 328, 329, 334, 335, 336, 337, 338, 349, 369, 372, 388, 390, 391, 393, 394, 395, 396, 397, 400, 403, 404, 405, 406, 408, 409, 410, 411, 413, 414, 415, 423, 426, 430, 462, 463, 464, 467, 481, 485, 527, 535, 539, 551, 552, 553, 555, 556, 589, 646, 685, 690, 798, 799, 800, 802, 804, 805, 807, 809, 810, 811, 812, 813, 814, 815, 817, 819, 825, 833, 834, 838, 839, 841, 842, 843, 845, 846, 852, 900, 921, 956, 961, 1055, 1105, 1117, 1146, 1186, 1199, 1201, 1217, 1236, 1240, 1245, 1520, 1599, 1606, 1608, 1634, 1641, 1647, 1648, 1649, 1651, 1654, 1702, 1829.
push_front: 93, 243.
push_onto_vector: [1140](#), [1141](#), 1146, [1149](#), [1150](#), 1151.
put: [150](#), [151](#), 199, 1634.
put_irods_object: 470, [1132](#), [1133](#), 1139.
pw_name: 1818, 1819.
pwd: [144](#), [145](#), 149, 596, [1818](#), 1819.
pwd_str: [235](#), 239, 241, 243.
PWD_TYPE: [9](#), [10](#), 57, 58.
qualifier: [1287](#), 1288, 1290.
qualifier_ctr: [1287](#), 1288, 1290.
qualifier_map: [1264](#), [1265](#), 1277, 1288, 1290.
qualifier_id: 357, 1280, 1290, 1292, 1307, [1319](#), 1321, 1325, 1329, 1331.
qualifier_map: [1264](#), [1265](#), 1277, 1280, 1288, 1331.
query: [107](#), [108](#), [1723](#), [1724](#).
query_strm: [63](#), 64.
query_vector: [109](#), [110](#), 111, [1232](#), 1240, 1242, [1728](#), [1729](#), 1730.
r: [14](#), [15](#), [18](#), [19](#).
range_iter_0: [919](#).
range_iter_1: [919](#).
rbegin: 919.
rcDisconnect: 50.
read: 425, 1560.
read_data: [1517](#), 1518, 1520, 1522, 1523, [1549](#), 1560.
receive_file: 142, 232, [234](#), [235](#), 252, 1632.
RECEIVE_FILE_CONTENTS: 236, 239, 245, 246, [248](#).
RECEIVE_METADATA_FILE_TYPE: [9](#), [10](#), 57, 58, 1634.
RECEIVE_PUT_FILE_TYPE: [9](#), [10](#), 57, 58, 1634.
reconnect: 44.
recv: 228, 232, 251, 1518, 1563.
recv_func_name: [1549](#), 1564.
ref_irods_object: [471](#), 473, 474, 475.
reference: [9](#), [33](#), [1628](#), [1629](#), 1630, 1631.
refs: [336](#), 582, 583, 584, 585, 586, 587, [752](#), 755, 757, 759, 761, 775, 791, 870, 872, 876, 1608.
refs_length: 581, 582, 583, 584, 585, 586, 587, [752](#), 755, 759, 775, 791, 869, 870, 872, 876.
remote: [1464](#), 1470.
remote_connection: [33](#), 39, 51, 218, 223, 225, 227, 229, 230, 251, 697, 1486, 1508, 1518, 1521, 1526, 1530, 1531, 1532, 1533, 1549, 1552, 1560, 1561, 1562, 1563.
remote_filename: [9](#), 19, 23, 25, 151, 152, 153, 155, 210, [234](#), [235](#), 236, 237, 238, 388, 390, 391, 393, 394, 395, 396, 397, 400, 403, 404, 405, 406, 408, 409, 410, 411, 413, 414, 415, 444, 445, 608, 610.
remote_plain_filename: [152](#), 153, 154, 155.
replace: 259, [873](#), [874](#), 876.
res: 1380, [1818](#).
RESERVED_0_INDEX: [752](#), [753](#), 763.
RESERVED_1_INDEX: [752](#), [753](#), 763.
RESERVED_2_INDEX: [752](#), [753](#), 763.
response: [93](#), [134](#), [135](#), [136](#), 138, [150](#), [151](#), 152, 153, 155, 156, 157, 158, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 175, 176, 179, 181, 182, 183, 184, 185, 186, 188, 189, 190, 191, 193, 194, 195, 196, 197, [201](#), 204, 205, 210, 211, 213, [232](#), [233](#), [273](#), [274](#), 276, [289](#), [290](#), [291](#), [296](#), [299](#), [300](#), [301](#), [302](#), [327](#), 328, 329, 334, 335, 336, 337, 338, [341](#), 372, 377, [384](#), 385, 386, 387, 388, 389, 390, 391, 393, 394, 395, 396, 397, 400, 401, 403, 404, 405, 406, 408, 409, 410, 411, 413, 414, 415, [418](#), 419, 424, 426, [444](#), 445, 462, 463, 467, 470, 471, 472, [478](#), 479, 480, [530](#), 531, 532, 534, 535, 538, 539, [550](#), 551, 552, 565, 566, [568](#), 569, [571](#), [572](#), [573](#), 574, 590, [592](#), [593](#), [595](#), [598](#), 599, [601](#), 602, [604](#), 605, [607](#), 608, 610, [614](#), 616, [618](#), 624, 625, 626, 628, [631](#), 632, [634](#), 635, [637](#), 638, [641](#), [644](#), [649](#), 650, [652](#), 653, [655](#), 656, 657, [660](#), 661, 662, 663, 664, 665, 667, 668, 669, 670, 671, 672, [675](#), 677, 678, 679, 680, 681, [684](#), 685, 686, [689](#), 690, 691, [694](#), 695, [697](#), [798](#), 799, 800, 802, 804, 805, 807, 809, 810, 812, 813, 814, 815, 817, 819, 825, [832](#), 833, 834, 838, 839, 841, 842, 845, 846, 852, [887](#), [1153](#), [1154](#), 1157, 1158, 1161, 1169, 1170, 1171, [1232](#), 1236, [1534](#), 1535, 1536, 1537, 1538, 1539, [1554](#), 1555, 1556, 1557, 1558, [1630](#), 1632, 1633, 1634, [1637](#), 1641, [1645](#), 1647, 1648, 1649, 1651, 1654, 1655.
response_deque: 33, 51, 93, 106, 153, 157, 158, 160, 161, 163, 168, 169, 171, 172, 173, 175, 176, 179, 181, 182, 184, 185, 186, 189, 190, 191, 194, 195, 196, 197, 200, 211, 213, 328, 329, 334, 335,

336, 337, 338, 372, 388, 390, 391, 393, 394, 395, 396, 397, 400, 403, 404, 405, 406, 408, 409, 410, 411, 413, 414, 415, 462, 535, 539, 551, 552, 553, 555, 556, 637, 645, 646, 685, 686, 690, 691, 796, 797, 798, 799, 800, 802, 804, 805, 807, 809, 810, 812, 813, 814, 815, 817, 819, 822, 825, 830, 831, 833, 834, 838, 839, 841, 842, 845, 846, 849, 852, 1527, 1528, 1529, 1531, 1533, 1534, 1549, 1550, 1552, 1553, 1554, 1558, 1641, 1647, 1648, 1649, 1651, 1654.
response_map: 33, 1631, 1634.
response_map_mutex: 33, 49, 53, 1631, 1632, 1634.
response_strm: 33.
Response_Type: 9, 10, 12, 13, 14, 15, 16, 17, 18, 19, 21, 23, 25, 33, 34, 36, 57, 58, 59, 60, 93, 134, 135, 136, 150, 151, 200, 201, 213, 232, 233, 273, 274, 289, 290, 291, 296, 299, 300, 301, 302, 327, 338, 341, 372, 377, 384, 418, 444, 462, 478, 530, 533, 535, 539, 550, 551, 552, 553, 555, 556, 565, 568, 571, 572, 573, 592, 595, 598, 601, 604, 607, 614, 618, 624, 631, 634, 637, 641, 644, 645, 646, 649, 652, 655, 660, 675, 684, 685, 689, 690, 694, 697, 796, 797, 798, 830, 831, 832, 883, 884, 887, 1153, 1154, 1231, 1232, 1236, 1528, 1534, 1536, 1537, 1539, 1554, 1555, 1556, 1630, 1631, 1634, 1637, 1645.
response_vector: 550, 556, 1231, 1232, 1236.
result: 63, 64, 65, 66, 67, 73, 74, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 102, 107, 108, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 151, 173, 175, 176, 177, 327, 334, 335, 336, 337, 344, 345, 346, 347, 348, 349, 350, 351, 352, 354, 355, 357, 359, 360, 361, 363, 364, 367, 420, 434, 435, 437, 438, 444, 455, 478, 480, 481, 483, 485, 486, 492, 493, 496, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 513, 515, 516, 517, 518, 519, 735, 736, 738, 740, 741, 742, 798, 804, 805, 807, 809, 810, 811, 812, 813, 815, 819, 820, 832, 838, 839, 841, 843, 846, 847, 874, 875, 877, 878, 879, 902, 904, 906, 907, 908, 909, 910, 912, 913, 914, 916, 922, 924, 927, 934, 937, 938, 939, 952, 953, 954, 956, 966, 968, 971, 972, 973, 982, 986, 987, 988, 989, 1010, 1021, 1027, 1028, 1029, 1030, 1031, 1032, 1064, 1066, 1067, 1068, 1069, 1071, 1073, 1074, 1078, 1080, 1081, 1083, 1084, 1085, 1086, 1087, 1089, 1090, 1091, 1092, 1093, 1094, 1095, 1096, 1098, 1099, 1100, 1102, 1103, 1104, 1105, 1107, 1109, 1110, 1111, 1112, 1113, 1114, 1115, 1116, 1117, 1122, 1124, 1127, 1129, 1130, 1154, 1159, 1160, 1174, 1182, 1183, 1184, 1185, 1187, 1194, 1196, 1197, 1198, 1199, 1204, 1206, 1209, 1211, 1212, 1213, 1214, 1215, 1216, 1217, 1222, 1298, 1300, 1301, 1302, 1303, 1304, 1305, 1306, 1307, 1308, 1309, 1310, 1311, 1314, 1315, 1316, 1317, 1582, 1583, 1584, 1585, 1587, 1588, 1589, 1591, 1592, 1600, 1601, 1602, 1603, 1609, 1610, 1611, 1620, 1621, 1622, 1623, 1662, 1663, 1664, 1665, 1666, 1668, 1669, 1670, 1671, 1723, 1724, 1749, 1750, 1803, 1821, 1822, 1823, 1825.
result_array: 109, 110, 111, 1232, 1240, 1241, 1242, 1243, 1244, 1728, 1729, 1730.
result_1: 344, 1154, 1159, 1160.
ret: 1428, 1482, 1483, 1500.
ret_val: 159, 160, 161, 212, 213, 214, 267, 268, 269, 283, 284, 285, 604, 605, 606, 655, 656, 657, 658, 660, 662, 663, 664, 665, 667, 668, 670, 671, 674, 675, 677, 678, 679, 680, 683, 684, 685, 686, 688, 689, 690, 691, 693, 874, 876, 877, 880, 902, 905, 906, 907, 909, 910, 912, 913, 914, 916, 922, 924, 925, 927, 937, 938, 940, 941, 966, 971, 975, 982, 987, 988, 990, 1010, 1017, 1022, 1027, 1029, 1030, 1032, 1064, 1067, 1068, 1069, 1071, 1073, 1075, 1077, 1083, 1087, 1118, 1122, 1125, 1127, 1129, 1131, 1154, 1162, 1163, 1206, 1298, 1301, 1302, 1303, 1304, 1305, 1306, 1307, 1308, 1309, 1311, 1314, 1316, 1318, 1582, 1584, 1585, 1587, 1588, 1591, 1592, 1594, 1595, 1597, 1600, 1601, 1602, 1603, 1607, 1609, 1610, 1612, 1613, 1614, 1615, 1616, 1617, 1739, 1745, 1746, 1747, 1762, 1766, 1767.
ret_val_ptr: 159, 212.
return_hvt: 899, 900.
return_hvt_vector: 900, 901, 902, 921.
right: 269.
RM_TYPE: 9.
rotate_log_file: 1686, 1687, 1688, 1689, 1694, 1695, 1711.
rotate_str: 1695, 1698, 1699.
row: 713, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729.
row_count_only: 1076.
row_ctr: 63, 64, 65, 66, 67, 73, 74, 75, 107, 108, 116, 118, 123, 124, 151, 173, 174, 175, 176, 327, 334, 335, 336, 344, 345, 350, 351, 352, 360, 361, 362, 363, 367, 434, 435, 436, 444, 478, 480, 481, 492, 493, 494, 495, 513, 515, 735, 736, 738, 739, 798, 804, 805, 806, 807, 809, 810, 815, 832, 838, 839, 840, 841, 902, 906, 907, 908, 909, 910, 911, 913, 914, 915, 916, 952, 953, 954, 955, 956, 966, 982, 1010, 1027, 1064, 1078, 1080, 1083, 1087, 1099, 1102, 1103, 1107, 1109, 1110, 1122, 1154, 1174, 1183, 1184, 1196, 1197, 1198, 1206, 1214, 1215, 1216, 1298, 1301, 1302, 1314, 1582, 1584,

1589, 1591, 1592, 1600, [1620](#), 1621, 1622, [1662](#), 1664, [1723](#), [1724](#), [1803](#), 1821, 1822.
row_ctr_vector: [109](#), [110](#), 111, [1232](#), 1240, 1242, 1243, 1244, 1257, [1728](#), [1729](#), 1730.
row_ctr_1: [344](#).
rrefs: [789](#), [790](#), 791.
rrefs_length: [789](#), [790](#), 791.
rsa_params: 1399, 1400, 1500.
RSPNTP_H: [29](#).
rv: [1416](#).
s: [24](#), [25](#), [35](#), [62](#), [105](#), [106](#), [254](#), [318](#), [319](#), [327](#), [743](#), [744](#), [871](#), [872](#), [882](#), [897](#), [898](#), [1023](#), [1024](#), [1060](#), [1061](#), [1274](#), [1275](#), [1277](#), [1330](#), [1331](#), [1346](#), [1356](#), [1357](#), [1364](#), [1365](#), [1404](#), [1418](#), [1462](#), [1732](#), [1735](#), [1818](#).
s_addr: 1482, 1500.
S_IROGRP: 245, 1466.
S_IROTH: 1466.
S_IRUSR: 1466.
S_IRWXU: 245.
S_IWGRP: 1466.
S_IWOTH: 1466.
S_IWUSR: 1466.
S_IXGRP: 245, 1466.
S_IXOTH: 1466.
S_IXUSR: 1466.
sa: [1416](#), [1421](#), [1422](#).
SA: [1413](#), 1482, 1483, [1494](#), 1500, 1501, [1512](#).
sa_cli: [1482](#), 1483, [1500](#), 1501.
sa_family: 1422.
sa_serv: [1482](#), [1500](#).
save_dc_metadata_id: [1298](#), 1304, 1309.
save_dc_metadata_sub_id: [1298](#), 1305, 1309.
save_DEBUG: [1567](#), 1571.
save_delay: [887](#), [1153](#), [1154](#), 1158, 1161, 1169, 1170.
save_delay_str: [1161](#).
save_global_thread_ctr: 1471, 1472, 1484, 1485, 1506, 1507.
save_irods_current_dir: [270](#), 271.
save_local_filename: [235](#), 239, 240.
save_local_path: [241](#), 243.
save_temp_files: 52, 1520.
save_thread_id: [1450](#).
Scan_Parse_Parameter_Type: [9](#), [29](#), [33](#), 34, 36, 38, [39](#), 49, 50, [51](#), 55, 57, 59, 60, 62, 68, 70, 77, 78, 102, 106, 107, 108, 109, 110, 113, 115, 126, 131, 133, 136, 142, 145, 149, 151, 199, 201, 215, 218, 231, 232, 233, 235, 245, 252, 253, 256, 272, 273, 274, 288, 301, [309](#), 319, 327, 339, 340, 341, 382, 384, 416, 418, 427, 429, 431, 433, 439, 441, 443, 444, 477, 478, 488, 490, 510, 511, 512, 513, 519, 520, 528, 530, 541, 550, 557, 565, 567, 568, 570, 571, 572, 573, 591, 592, 594, 595, 597, 598, 600, 601, 603, 604, 606, 607, 612, 613, 614, 617, 618, 623, 624, 630, 631, 633, 634, 636, 637, 640, 641, 643, 644, 648, 649, 651, 652, 654, 655, 659, 660, 674, 675, 683, 684, 688, 689, 693, 694, 696, 697, [702](#), [749](#), [752](#), 764, 802, 809, 810, 814, 815, 842, [883](#), [884](#), [887](#), [992](#), [993](#), [996](#), 1012, 1029, [1038](#), 1045, 1046, 1067, 1068, 1176, [1264](#), 1297, [1319](#), [1333](#), [1335](#), [1338](#), 1403, 1404, 1447, 1468, 1469, 1483, 1500, 1501, 1516, 1517, 1526, 1535, 1536, 1537, 1538, 1539, 1548, 1549, 1555, 1556, 1557, 1558, 1628, 1629, 1632, 1636, 1637, 1639, 1640, 1641, 1644, 1645, 1647, 1649, 1678, 1723.
SCPRPMT_H: [306](#).
sd: [1416](#), 1418, [1419](#), [1420](#).
sec_val: [1678](#), 1681.
second: 106, 337, 367, 574, 575, 576, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 668, 763, 894, 898, 916, 919, 945, 949, 987, 1012, 1169, 1170, 1176, 1226, 1227, 1228, 1273, 1275, 1277, 1280, 1288, 1289, 1290, 1307, 1325, 1331, 1449, 1450, 1539, 1558, 1631, 1632, 1646, 1829.
seconds: [130](#), [131](#), [1731](#), [1732](#), [1733](#), [1734](#), [1735](#).
seekg: 425.
send: 223, 225, 228, 230, 232, 697, 1526, 1530, 1531, 1532, 1552, 1560, 1561, 1562.
SEND_FILE_TYPE: [9](#), [10](#), 57, 58, 59, 60, 200, 213.
send_func_str: [218](#), 223, 225, 229, 230.
SEND_HANDLE_TYPE: [9](#), [10](#), 57, 58, 338, 637.
SEND_METADATA_TYPE: [9](#), [10](#), 57, 58, 372.
send_response: [253](#), [340](#).
send_tan_list: [114](#), [115](#), 129, 642.
SEND_TAN_LIST_TYPE: [9](#), [10](#), 57, 58.
send_to_peer: [217](#), [218](#), 231, [232](#), [233](#), 565, 566, 568, 569, 574, 590, 593, 596, 599, 602, 605, 612, 616, 621, 627, 632, 635, 639, 642, 647, 650, 653, 657, 673, 682, 687, 692, 696, 1539, 1558.
separator: [1731](#), [1732](#), 1733.
serial: [1389](#), 1395, [1434](#), 1435.
serialNumber: 63, 92, [702](#), 706, 708, 710, 712, 721, 731, 733, 744, 746, 1435, 1653.
SERVER: [1415](#).
server_action_add_handle_value: 57, [301](#), [655](#), 659.
server_action_cd: 57, [301](#), [598](#), 600.
server_action_command_only: 57, [301](#), [565](#), 567.
server_action_create_handle: 57, [301](#), [652](#), 654.
server_action_delete_handle: 57, [301](#), [660](#), 674.
server_action_delete_handle_value: 57, [301](#), [684](#), 688.
server_action_end_server: 57, [301](#), [618](#), 623.

server_action_get: 57, [301](#), [607](#), 613.
server_action_get_handle: 57, [301](#), [637](#), 640.
server_action_get_metadata: 57, [301](#), [634](#), 636.
server_action_get_user_info: 57, [301](#), [649](#), 651.
server_action_ls: 57, [301](#), [592](#), 594.
server_action_map: [34](#), [36](#), 57, 301, 1536, 1537, 1538, 1539.
server_action_mark_irods_objects_for_deletion: 57, [301](#), [604](#), 606.
server_action_mkdir: 57, [301](#), [601](#), 603.
server_action_name: [1536](#), 1538, 1539.
server_action_name_map: [34](#), [36](#), 58, 1537, 1539.
server_action_process_pending: 57, [301](#), [644](#), 648.
server_action_pwd: 57, [301](#), [595](#), 597.
server_action_receive_metadata_file: 57, [301](#), [572](#).
server_action_receive_put_file: 57, [301](#), [571](#).
server_action_send_file: 57, [301](#), [568](#), 570.
server_action_send_handle: 57, [301](#), [573](#), 591.
server_action_send_metadata: 57, [301](#), [614](#), 617.
server_action_send_tan_list: 57, [301](#), [641](#), 643.
server_action_show_certificate: 57, [301](#), [631](#), 633.
server_action_sleep: 57, [301](#), [624](#), 630.
server_action_undelete_file: 57, [301](#), [694](#), 696.
server_action_undelete_handle: 57, [301](#), [675](#), 683.
server_action_undelete_handle_value: 57, [301](#), [689](#), 693.
server_action_unknown: 57, [301](#), [697](#), 1538.
server_cert: 34.
server_finished: [33](#), 39, 106, 1521, 1528, 1529, 1530, 1531, 1550, 1551, 1552, 1565.
servinfo: [1416](#), 1418.
session: 33, 51, 223, 225, 227, 230, 251, 697, 702, 1383, 1384, 1388, 1389, 1402, 1427, 1428, 1430, 1483, 1489, 1501, 1502, 1503, 1505, 1518, 1526, 1530, 1531, 1552, 1560, 1561, 1562, 1563.
session_data: [1371](#), 1378, 1380.
session_data_size: [1371](#), 1378, 1380, 1382.
session_id: [1371](#), 1378, 1380, 1382.
session_id_size: [1371](#), 1378, 1380, 1382.
set: 79, 163, 183, 188, 337, 408, 446, 467, 471, 481, 535, 552, [711](#), [712](#), [713](#), [714](#), 729, 741, [764](#), [765](#), 788, [789](#), [790](#), 795, 807, 841, 923, 956, 965, 966, 970, 978, [1005](#), [1006](#), [1043](#), [1044](#), 1232, [1345](#), [1346](#), 1349, 1357, 1607, 1608, 1641.
set_debug_level: 39, 51, 62, 70, 110, 115, 136, 145, 151, 201, 218, 235, 256, 274, 327, 341, 384, 418, 429, 433, 441, 444, 478, 490, 513, 520, 530, 550, 565, 568, 573, 592, 595, 598, 601, 604, 607, 614, 618, 624, 631, 634, 637, 641, 644, 649, 652, 655, 660, 675, 684, 689, 694, 697, 714, 735, 757, 759, 761, 763, 765, 790, 797, 831, 856, 874, 892, 894, 900, 902, 927, 944, 949, 952, 960, 966, 978, 982, 1006, 1010, 1026, 1042, 1044, 1046, 1063, 1077, 1120, 1133, 1141, 1150, 1154, 1174, 1206, 1224, 1230, 1232, 1273, 1277, 1279, 1283, 1287, 1292, 1296, 1298, 1314, 1325, 1327, 1346, 1389, 1398, 1400, 1404, 1416, 1428, 1446, 1455, 1462, 1481, 1499, 1517, 1549, 1581, 1620, 1629, 1637, 1645, 1662, 1677, 1695, 1729, 1732, 1735, 1739, 1750, 1753, [1758](#), [1759](#), 1761, 1770, 1782, 1785, 1790, 1791, 1802, 1803.
set_ellipsis: [552](#).
set_expires: [130](#), [131](#).
set_handle_id: 398, [1313](#), [1314](#), 1318.
SET_HANDLE_ID_UNLOCK_TABLES: 1316, [1317](#).
set_password: [1781](#), [1782](#), 1783.
set_privileges: [491](#), 494, 510.
set_user: [69](#), [70](#), 71, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 91, 92, 93, 94.
set_user_info: 33, [103](#), [511](#), 1639, 1641, 1648.
setfill: 744, 1365, 1606, 1750.
setsockopt: 1482, 1500.
setup_java_vm: 40.
setw: 269, 346, 347, 348, 349, 354, 355, 356, 357, 358, 359, 364, 365, 366, 744, 763, 1277, 1365, 1448, 1606, 1750, 1830.
show: [24](#), [25](#), 79, 92, [105](#), [106](#), 162, 170, 171, 181, [318](#), [319](#), 330, 337, 368, 374, 379, 384, 387, 392, 394, 395, 397, 400, 408, 418, 426, 444, 446, 451, 462, 463, 465, 466, 467, 473, 480, 481, 482, 530, 537, 569, 608, 657, 660, 664, 667, 675, 678, 684, 689, 694, 742, [743](#), [744](#), 808, 816, 841, 844, [871](#), [872](#), [882](#), [897](#), [898](#), 900, 921, 933, 956, 963, 1010, 1012, [1023](#), [1024](#), 1027, 1056, [1060](#), [1061](#), 1098, 1117, 1120, 1133, 1141, 1154, 1158, 1168, 1176, 1189, 1220, 1230, 1236, 1247, 1273, [1274](#), [1275](#), 1280, 1284, 1292, 1306, [1330](#), [1331](#), [1364](#), [1365](#), 1439, 1447, 1502, 1534, 1631, 1633, 1639, 1641, 1655, 1832.
SHOW_CERTIFICATE_TYPE: [9](#), [10](#), 57, 58.
show_certificates: [296](#), [478](#), 488, 632.
SHOW_CERTIFICATES_PRIVILEGE: [35](#), [36](#), 106, 319, 479, 507, 512.
show_distinguished_names_bitset: [1639](#), [1641](#).
SHOW_DISTINGUISHED_NAMES_PRIVILEGE: [35](#), [36](#), 106, 319, 508, 512, 1639, 1641.
show_groups: 506.
SHOW_GROUPS_PRIVILEGE: [35](#), [36](#), 106, 506, 512.
show_issuer: [743](#), [744](#).
show_privileges: [297](#), [512](#), 684.
SHOW_PRIVILEGES_PRIVILEGE: [35](#), [36](#), 106, 319, 509, 512.
show_user_info: 505.
SHOW_USER_INFO_PRIVILEGE: [35](#), [36](#), 106, 319,

505, 512, 1647, 1649.
SHUT_RDWR: 1420.
shutdown: 1420.
SIGINT: 1451.
signal_name_map: 1757.
signal_number_map: 1757.
sin_addr: 1422, 1482, 1483, 1500, 1501.
sin_family: 1482, 1500.
sin_port: 1482, 1483, 1500, 1501.
sin6_addr: 1422.
size: 25, 52, 106, 136, 148, 153, 155, 169, 170, 177, 178, 179, 180, 181, 186, 207, 238, 259, 267, 283, 337, 350, 374, 380, 381, 395, 397, 400, 403, 409, 426, 442, 462, 467, 468, 482, 532, 536, 537, 551, 554, 555, 589, 593, 602, 605, 645, 646, 667, 812, 813, 814, 859, 863, 864, 898, 902, 905, 923, 949, 956, 957, 963, 967, 983, 1056, 1061, 1070, 1071, 1072, 1073, 1106, 1117, 1128, 1129, 1136, 1155, 1156, 1164, 1165, 1167, 1170, 1188, 1189, 1194, 1200, 1202, 1208, 1218, 1219, 1230, 1234, 1235, 1247, 1273, 1275, 1279, 1280, 1325, 1331, 1349, 1378, 1380, 1382, 1389, 1390, 1395, 1406, 1434, 1435, 1449, 1450, 1473, 1527, 1528, 1529, 1531, 1533, 1549, 1550, 1552, 1553, 1558, 1615, 1616, 1704, 1707, 1709, 1710, 1830, 1831, 1832, 1833.
size_type: 1346.
sleep: 46, 47, 228, 628, 1692, 1796.
sleep_server_enabled: 625, 626, 628, 629.
SLEEP_TYPE: 9, 10, 57, 58.
sleep_val: 1791, 1796.
sleep_value: 1677, 1692.
SO_REUSEADDR: 1482, 1500.
sock: 33, 39, 51, 106, 223, 225, 228, 229, 230, 251, 697, 1470, 1483, 1501, 1502, 1518, 1520, 1521, 1522, 1524, 1526, 1530, 1531, 1532, 1539, 1552, 1558, 1560, 1561, 1562, 1563.
SOCK_STREAM: 1416, 1464, 1482, 1500.
sockaddr: 1413, 1418, 1421, 1422, 1465, 1470, 1494, 1512.
sockaddr_in: 1416, 1422, 1482, 1500.
sockaddr_in6: 1422.
sockaddr_un: 1464.
socket: 1416, 1464, 1482, 1500.
SOCKET_ERR: 1494, 1512.
socket_path: 1463, 1464, 1466, 1468, 1470.
socklen_t: 1470, 1483, 1501.
SOL_SOCKET: 1482, 1500.
source: 1752, 1753, 1754.
space: 1296.
SPPTFNC1_H: 545.
SPPTFNC2_H: 561.
sprintf: 1387.
sql_mutex: 1723, 1724, 1726.
sql_strm: 73, 116, 118, 122, 123, 127, 151, 173, 177, 327, 332, 333, 334, 344, 350, 360, 433, 434, 444, 478, 480, 492, 493, 513, 514, 515, 519, 735, 736, 798, 803, 804, 811, 815, 816, 818, 819, 832, 837, 838, 843, 844, 845, 846, 874, 876, 877, 902, 906, 909, 912, 913, 916, 918, 922, 927, 934, 935, 936, 937, 939, 952, 953, 966, 968, 970, 971, 973, 982, 986, 987, 988, 989, 1010, 1021, 1027, 1030, 1064, 1066, 1067, 1069, 1070, 1071, 1072, 1073, 1078, 1080, 1099, 1105, 1107, 1117, 1122, 1126, 1127, 1128, 1129, 1154, 1158, 1159, 1174, 1182, 1183, 1187, 1190, 1191, 1192, 1193, 1194, 1196, 1199, 1206, 1209, 1211, 1212, 1213, 1214, 1232, 1236, 1237, 1238, 1239, 1240, 1242, 1298, 1300, 1301, 1306, 1307, 1308, 1309, 1314, 1315, 1316, 1317, 1582, 1583, 1584, 1589, 1591, 1592, 1600, 1603, 1611, 1620, 1621, 1662, 1663, 1664, 1668, 1669, 1670, 1803, 1821.
sql_strm_1: 344, 1154, 1158, 1159, 1298, 1307, 1308, 1309, 1582, 1599, 1604, 1606, 1609.
sql_strm_2: 1582, 1599, 1604, 1606, 1610.
srp_dh_group2048: 1396, 1398.
SRVRACTN_H: 699.
sscanf: 67, 159, 212, 423, 1015, 1016, 1177, 1179, 1252.
sse: 1731, 1732, 1733, 1736, 1737.
serialNumber: 709, 710, 711, 712.
sstateOrProvinceName: 709, 710, 711, 712, 1343, 1344.
stack: 1264.
standalone_handle: 328, 660, 675, 803, 837, 902, 934, 952, 961, 966, 986, 1183, 1208.
standalone_hs: 752, 887, 1580, 1581, 1582, 1583, 1609, 1610, 1831.
start: 423, 1284, 1286, 1287, 1290.
start_empty: 1782.
start_val: 425.
stateOrProvinceName: 702, 708, 710, 712, 727, 731, 733, 744, 746, 1338, 1344, 1349, 1352, 1355, 1361, 1363, 1365, 1406, 1653.
status: 39, 41, 47, 51, 52, 54, 62, 64, 70, 73, 91, 92, 95, 97, 99, 110, 111, 115, 117, 118, 121, 122, 123, 126, 127, 128, 136, 139, 140, 145, 146, 147, 148, 151, 157, 158, 163, 167, 168, 171, 172, 173, 179, 181, 182, 184, 185, 186, 189, 190, 191, 192, 193, 194, 195, 196, 197, 201, 211, 218, 223, 225, 227, 228, 229, 230, 235, 239, 241, 242, 245, 246, 247, 251, 256, 263, 264, 265, 266, 274, 275, 279, 280, 281, 282, 287, 327, 328, 329, 331, 334, 337, 341, 344, 350, 360, 371, 372, 378, 384, 388, 389, 391, 392, 394, 395, 396, 397, 398, 400, 401,

403, 404, 405, 406, 408, 413, 418, 421, 423, 426, 429, 430, 433, 434, 438, 441, 444, 446, 447, 448, 450, 453, 456, 461, 462, 463, 465, 466, 468, 469, 470, 471, 472, 473, 474, 476, 478, 480, 481, 486, 490, 493, 513, 515, 520, 530, 535, 538, 540, 550, 552, 553, 556, 565, 566, 567, 568, 569, 573, 574, 576, 577, 583, 584, 590, 592, 593, 595, 596, 598, 599, 601, 602, 604, 605, 607, 608, 612, 614, 616, 618, 621, 624, 627, 628, 631, 632, 634, 635, 637, 638, 639, 641, 642, 644, 647, 649, 650, 652, 653, 655, 657, 660, 664, 669, 670, 671, 673, 675, 678, 679, 680, 682, 684, 686, 687, 689, 691, 692, 694, 695, 696, 697, 714, 728, 729, 735, 736, 741, 765, 781, 782, 786, 790, 792, 793, 797, 799, 804, 807, 815, 819, 827, 831, 833, 838, 841, 846, 874, 875, 876, 877, 879, 900, 902, 903, 904, 906, 909, 913, 922, 924, 927, 932, 934, 937, 939, 940, 944, 952, 953, 956, 960, 961, 966, 968, 971, 973, 978, 982, 986, 987, 988, 989, 1010, 1012, 1015, 1016, 1021, 1026, 1028, 1029, 1030, 1031, 1046, 1063, 1065, 1066, 1067, 1068, 1069, 1071, 1073, 1074, 1077, 1079, 1080, 1081, 1099, 1100, 1107, 1117, 1120, 1123, 1124, 1127, 1129, 1130, 1133, 1134, 1135, 1137, 1138, 1141, 1144, 1145, 1147, 1150, 1151, 1154, 1159, 1162, 1163, 1166, 1169, 1170, 1171, 1174, 1176, 1178, 1179, 1182, 1183, 1184, 1189, 1191, 1194, 1196, 1201, 1203, 1206, 1208, 1209, 1210, 1211, 1212, 1213, 1214, 1219, 1221, 1225, 1228, 1232, 1240, 1241, 1245, 1249, 1250, 1251, 1252, 1283, 1284, 1298, 1299, 1300, 1301, 1306, 1307, 1308, 1309, 1310, 1314, 1315, 1316, 1317, 1389, 1400, 1404, 1405, 1418, 1428, 1429, 1434, 1439, 1446, 1448, 1449, 1450, 1451, 1452, 1462, 1463, 1465, 1466, 1467, 1473, 1481, 1482, 1486, 1499, 1500, 1501, 1502, 1503, 1508, 1517, 1518, 1519, 1520, 1521, 1524, 1531, 1532, 1533, 1538, 1539, 1549, 1551, 1557, 1558, 1560, 1561, 1562, 1563, 1564, 1565, 1566, 1568, 1569, 1570, 1582, 1583, 1584, 1591, 1592, 1597, 1600, 1603, 1607, 1609, 1610, 1611, 1613, 1614, 1615, 1616, 1620, 1621, 1629, 1632, 1634, 1637, 1639, 1641, 1645, 1648, 1649, 1662, 1663, 1664, 1670, 1671, 1678, 1680, 1683, 1684, 1685, 1686, 1687, 1688, 1689, 1691, 1695, 1696, 1700, 1702, 1703, 1705, 1708, 1709, 1724, 1727, 1729, 1730, 1737, 1750, 1753, 1761, 1764, 1765, 1766, 1770, 1771, 1772, 1773, 1774, 1776, 1782, 1783, 1785, 1790, 1791, 1795, 1796, 1802, 1803, 1821, 1831, 1832.

status.1: 1678, 1683, 1684.

std: 5, 6, 27, 305, 306, 320, 321, 346, 347, 348, 349, 354, 355, 356, 357, 358, 359, 364, 365, 366, 544, 560, 698, 744, 748, 883, 965, 966, 978, 992, 1034, 1035, 1232, 1258, 1259, 1277, 1333, 1335, 1365, 1366, 1409, 1423, 1441, 1457, 1476, 1477, 1494, 1495, 1512, 1513, 1544, 1545, 1576, 1577, 1624, 1625, 1657, 1658, 1672, 1673, 1712, 1713, 1786, 1787, 1798, 1835.

stderr: 251, 576, 582, 771, 791, 872, 1226, 1227, 1284, 1296, 1384, 1428, 1429, 1430, 1433, 1440, 1519, 1678, 1689, 1750, 1754.

stderr_filename: 1689.

stdout: 1549, 1688.

stdout_filename: 1688.

store: 384, 385, 409, 413.

store_dc_metadata: 290, 413, 444, 477.

str: 54, 62, 63, 64, 70, 73, 93, 97, 106, 108, 110, 115, 118, 122, 123, 126, 127, 132, 133, 136, 138, 142, 145, 146, 148, 149, 153, 157, 158, 160, 161, 163, 168, 169, 171, 172, 173, 175, 176, 177, 179, 181, 182, 183, 184, 185, 186, 188, 189, 190, 191, 194, 195, 196, 197, 207, 210, 211, 213, 214, 260, 261, 264, 265, 268, 269, 270, 271, 275, 276, 277, 280, 281, 284, 285, 286, 319, 328, 329, 332, 333, 334, 335, 336, 337, 344, 345, 346, 347, 348, 350, 351, 352, 354, 355, 357, 360, 363, 364, 369, 370, 371, 372, 377, 378, 379, 381, 388, 389, 390, 391, 393, 394, 395, 396, 397, 400, 401, 403, 404, 405, 406, 407, 408, 409, 410, 411, 413, 414, 415, 419, 434, 441, 452, 462, 478, 480, 481, 482, 484, 485, 486, 487, 492, 493, 512, 514, 515, 516, 519, 520, 530, 531, 532, 535, 539, 551, 552, 553, 555, 556, 565, 568, 573, 574, 588, 589, 590, 592, 595, 598, 599, 601, 604, 607, 610, 614, 618, 624, 625, 626, 631, 634, 637, 641, 644, 649, 652, 653, 655, 656, 657, 660, 662, 663, 664, 665, 667, 668, 670, 671, 672, 675, 677, 678, 679, 680, 681, 684, 685, 689, 690, 694, 697, 736, 744, 746, 798, 799, 800, 802, 803, 804, 805, 807, 809, 810, 811, 812, 813, 814, 815, 817, 818, 819, 823, 825, 833, 834, 837, 838, 839, 841, 842, 843, 845, 846, 850, 852, 872, 876, 877, 898, 906, 909, 912, 913, 916, 918, 922, 927, 934, 935, 937, 939, 953, 968, 970, 971, 973, 986, 987, 988, 989, 1013, 1014, 1021, 1024, 1026, 1030, 1061, 1066, 1067, 1069, 1070, 1071, 1072, 1073, 1080, 1099, 1105, 1107, 1117, 1126, 1127, 1128, 1129, 1134, 1136, 1137, 1144, 1158, 1159, 1161, 1162, 1177, 1178, 1182, 1183, 1187, 1193, 1194, 1196, 1199, 1209, 1211, 1212, 1213, 1214, 1236, 1237, 1238, 1239, 1240, 1242, 1248, 1277, 1280, 1292, 1296, 1300, 1301, 1306, 1307, 1308, 1309, 1315, 1316, 1317, 1363, 1365, 1404, 1448, 1468, 1483, 1500, 1549, 1552, 1553, 1559, 1560, 1561, 1562, 1563, 1564, 1565, 1566, 1583, 1584, 1589, 1591, 1592, 1600, 1603,

1604, 1606, 1609, 1610, 1611, 1621, 1629, 1637, 1641, 1645, 1647, 1648, 1649, 1651, 1653, 1654, 1663, 1664, 1666, 1668, 1669, 1670, 1678, 1698, 1705, 1707, 1708, 1709, 1750, 1761, 1763, 1766, 1772, 1775, 1777, 1779, 1783, 1821.

str_size: [132](#), [133](#).

strcat: 141, 526, 590, 1428.

strcmp: 781, 782, 786.

strcpy: 45, 100, 138, 139, 140, 141, 142, 146, 147, 148, 149, 207, 210, 214, 233, 235, 250, 261, 264, 265, 268, 269, 271, 275, 277, 280, 281, 284, 285, 286, 344, 345, 346, 347, 348, 350, 351, 352, 354, 355, 357, 360, 363, 364, 369, 370, 371, 378, 381, 479, 480, 481, 485, 486, 531, 532, 535, 566, 569, 574, 599, 610, 616, 619, 620, 647, 653, 656, 657, 662, 663, 664, 665, 667, 668, 670, 671, 672, 677, 678, 679, 680, 681, 1428, 1464, 1502, 1520, 1526, 1530, 1531, 1551, 1566.

strerror: 41, 52, 95, 96, 99, 100, 138, 157, 158, 176, 211, 223, 225, 227, 229, 230, 239, 245, 246, 250, 268, 275, 284, 287, 346, 347, 348, 354, 355, 357, 364, 369, 437, 452, 453, 485, 486, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 518, 527, 540, 589, 697, 715, 716, 718, 719, 720, 721, 781, 782, 784, 785, 786, 798, 827, 865, 912, 916, 929, 940, 1014, 1042, 1044, 1049, 1086, 1089, 1090, 1091, 1092, 1093, 1094, 1095, 1096, 1105, 1112, 1125, 1157, 1162, 1171, 1178, 1185, 1199, 1217, 1252, 1304, 1305, 1435, 1437, 1438, 1451, 1463, 1464, 1465, 1466, 1467, 1470, 1482, 1500, 1520, 1532, 1560, 1561, 1562, 1564, 1566, 1587, 1602, 1613, 1614, 1615, 1616, 1668, 1680, 1681, 1682, 1683, 1691, 1696, 1699, 1700, 1705, 1724, 1727, 1735, 1737, 1754, 1766, 1771, 1772, 1775, 1782, 1785, 1814, 1818, 1825.

strftime: 133, 744, 781, 782, 786, 792, 793, 1024, 1042, 1044, 1125, 1437, 1438, 1697, 1732, 1733, 1735, 1737.

string: 9, 10, 24, 25, 33, 34, 35, 36, 41, 42, 50, 52, 62, 69, 70, 79, 96, 105, 106, 107, 108, 109, 110, 115, 134, 135, 136, 138, 144, 145, 152, 165, 166, 167, 200, 201, 203, 206, 217, 218, 234, 235, 238, 240, 241, 243, 244, 247, 253, 254, 255, 256, 259, 267, 270, 273, 274, 276, 283, 290, 292, 293, 294, 295, 296, 300, 309, 318, 319, 327, 328, 341, 343, 352, 363, 379, 384, 386, 389, 396, 429, 432, 433, 441, 444, 461, 478, 513, 520, 530, 534, 535, 550, 552, 565, 568, 573, 574, 590, 592, 595, 598, 601, 604, 607, 614, 618, 624, 631, 632, 634, 635, 637, 638, 641, 644, 649, 652, 655, 660, 675, 684, 689, 694, 697, 702, 709, 710, 711, 712, 713, 714, 734, 735, 743, 744, 745, 746, 752, 753, 763, 764, 765, 789, 790, 796, 797, 798, 803, 816, 821, 830, 831, 832, 837, 844, 848, 855, 856, 857, 861, 871, 872, 873, 874, 876, 882, 887, 897, 898, 899, 900, 902, 916, 919, 927, 934, 943, 944, 951, 952, 953, 959, 960, 961, 965, 966, 970, 977, 978, 981, 982, 986, 996, 999, 1000, 1005, 1006, 1023, 1024, 1026, 1038, 1041, 1042, 1043, 1044, 1045, 1046, 1047, 1060, 1061, 1064, 1070, 1072, 1122, 1128, 1132, 1133, 1140, 1141, 1149, 1150, 1153, 1154, 1158, 1161, 1173, 1174, 1183, 1190, 1205, 1206, 1208, 1223, 1224, 1229, 1230, 1231, 1232, 1264, 1265, 1274, 1275, 1277, 1280, 1287, 1288, 1296, 1298, 1307, 1319, 1325, 1330, 1331, 1338, 1343, 1344, 1345, 1346, 1347, 1349, 1356, 1357, 1362, 1363, 1364, 1365, 1404, 1415, 1416, 1446, 1500, 1517, 1536, 1549, 1555, 1576, 1580, 1581, 1582, 1593, 1594, 1598, 1615, 1619, 1620, 1628, 1629, 1630, 1637, 1645, 1646, 1649, 1661, 1662, 1678, 1688, 1689, 1694, 1695, 1705, 1723, 1724, 1728, 1729, 1730, 1731, 1732, 1733, 1734, 1735, 1736, 1737, 1738, 1739, 1740, 1743, 1749, 1750, 1752, 1753, 1761, 1762, 1766, 1769, 1770, 1781, 1782, 1784, 1785, 1803, 1828, 1829, 1831.

string_val: [9](#), [19](#), [23](#), [25](#), [650](#), [685](#), [686](#), [690](#), [691](#).

string_value: [824](#), [851](#).

string_vector: [9](#), [17](#), [19](#), [23](#), [25](#), [34](#), [53](#), [106](#), [136](#), [138](#), [274](#), [532](#), [535](#), [551](#), [552](#), [593](#), [602](#), [605](#).

strings: 1406.

stringstream: 42, 54, 62, 63, 70, 73, 105, 106, 108, 110, 115, 116, 123, 130, 131, 132, 133, 136, 145, 151, 201, 218, 256, 274, 318, 319, 327, 341, 344, 369, 384, 418, 433, 441, 444, 478, 492, 512, 513, 520, 530, 550, 565, 568, 573, 592, 595, 601, 604, 607, 614, 618, 624, 631, 634, 635, 675, 684, 689, 694, 697, 735, 743, 744, 746, 765, 797, 798, 831, 832, 871, 872, 874, 897, 898, 902, 927, 952, 966, 982, 1010, 1023, 1024, 1026, 1027, 1047, 1060, 1061, 1064, 1078, 1120, 1122, 1133, 1141, 1154, 1161, 1174, 1206, 1232, 1263, 1277, 1279, 1283, 1296, 1298, 1314, 1363, 1364, 1365, 1404, 1446, 1468, 1483, 1500, 1517, 1549, 1582, 1620, 1629, 1637, 1645, 1662, 1678, 1695, 1750, 1753, 1761, 1770, 1782, 1803.

strlen: 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 97, 98, 106, 120, 125, 141, 159, 160, 161, 212, 213, 224, 225, 226, 251, 266, 282, 346, 347, 348, 349, 354, 355, 356, 357, 358, 359, 364, 365, 366, 420, 423, 425, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 517, 524, 526, 527, 589, 605, 609, 611, 612, 673, 682, 687, 692, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725,

726, 727, 728, 729, 767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 784, 785, 786, 872, 876, 1017, 1018, 1024, 1050, 1051, 1052, 1053, 1054, 1055, 1085, 1097, 1104, 1110, 1111, 1113, 1114, 1115, 1116, 1284, 1288, 1296, 1304, 1305, 1464, 1502, 1526, 1530, 1531, 1549, 1550, 1551, 1552, 1554, 1561, 1591, 1592, 1666, 1668, 1770, 1772, 1773, 1774, 1783, 1785, 1809, 1810, 1811, 1812, 1813, 1814, 1817, 1818.
strm: 105, 106, 297, 318, 319, 512, 743, 744, 871, 872, 897, 898, 1023, 1024, 1060, 1061, 1364, 1365.
strm_val: 1549, 1552, 1553, 1559, 1560, 1561, 1562, 1563, 1564, 1565, 1566.
strcmp: 423, 1226, 1810.
strcpy: 487, 625, 626, 956, 1772.
strftime: 781, 782, 786, 1024, 1055, 1737.
strtol: 267, 283, 389, 421, 784, 916, 1587, 1766, 1814, 1825.
strtoul: 176, 346, 347, 348, 354, 355, 357, 364, 437, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 518, 715, 716, 718, 719, 720, 721, 774, 780, 781, 782, 785, 786, 865, 912, 1086, 1089, 1090, 1091, 1092, 1093, 1094, 1095, 1096, 1105, 1112, 1185, 1199, 1217, 1243, 1244, 1304, 1305, 1435, 1591, 1592, 1602, 1737, 1743, 1754, 1777.
STX (ASCII control character) : 610.
sub_handle: 798, 799, 800, 801, 803, 832, 833, 834, 835, 837, 855, 856, 867.
subject: 702, 1403, 1404.
submit_mysql_queries: 109, 110, 111, 113, 1240, 1728, 1729, 1730.
submit_mysql_query: 64, 73, 107, 108, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 173, 334, 344, 350, 360, 434, 480, 493, 515, 736, 804, 815, 819, 838, 846, 875, 877, 879, 904, 906, 909, 913, 922, 924, 934, 937, 939, 953, 968, 971, 973, 986, 987, 988, 989, 1021, 1028, 1030, 1031, 1066, 1069, 1071, 1073, 1074, 1080, 1099, 1107, 1124, 1127, 1129, 1130, 1159, 1182, 1183, 1194, 1196, 1209, 1211, 1212, 1213, 1214, 1300, 1301, 1306, 1307, 1308, 1309, 1310, 1315, 1316, 1317, 1583, 1584, 1591, 1592, 1600, 1603, 1609, 1610, 1611, 1621, 1663, 1664, 1665, 1666, 1668, 1669, 1670, 1671, 1723, 1724, 1727, 1730, 1821.
substr: 238, 240, 243, 1288, 1349.
suffix_str: 752, 887, 1580, 1581, 1606, 1803, 1812, 1827, 1831.
sun_family: 1464.
sun_path: 1464.
SUPERUSER_PRIVILEGE: 35, 36, 106, 319, 512, 662, 665, 685, 690, 802, 809, 810, 814, 815, 842.
suppress_prompt: 525.
sysconf: 1818.
system: 54, 184, 189, 372, 408, 1134, 1137, 1144, 1708, 1760, 1762, 1766.
t: 228, 467, 999, 1000, 1005, 1006, 1436, 1464, 1732.
table: 295, 513, 514, 515, 516, 517, 518.
TAN_ctr: 116, 120, 121, 1662, 1666, 1667, 1668, 1670.
tan_strm: 123, 125, 126.
TANFNCS_H: 1673.
tans_per_iteration: 1661, 1662, 1788, 1791.
target: 1661, 1662, 1667, 1668, 1788, 1791.
tcp_close: 51, 1419, 1420.
tcp_connect: 1414, 1415, 1416, 1418.
temp_bitset: 91, 498, 1639, 1641.
temp_buffer: 573, 574, 575, 765, 781, 782, 786, 1770.
temp_cert: 92.
temp_char_ptr: 765, 781, 782, 786.
temp_cstr: 1296.
temp_ctr: 141, 142.
temp_deque: 646.
temp_file_vector: 34, 50, 52, 96, 100, 105, 106, 204, 369, 485, 527, 589, 1520, 1634.
temp_filename: 96, 97, 99, 100, 204, 235, 250, 251, 369, 370, 372, 389, 485, 486, 520, 527, 573, 589, 590, 592, 593, 632, 635, 637, 644, 1517, 1630, 1632, 1633, 1782.
temp_filename_ptr: 234, 235, 250.
temp_handle_id_vector: 1174, 1186, 1188, 1189, 1194, 1206, 1217, 1218, 1219.
temp_handle_value_vector: 336, 337.
temp_handle_vector: 1174, 1189, 1190, 1206, 1219, 1220.
temp_path: 235, 243, 244, 245.
temp_prefix_str: 1593, 1594.
temp_privileges: 91, 93, 491, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510.
temp_ptr: 1047, 1049, 1055.
temp_ret_val: 98.
temp_str: 42, 47, 98, 235, 238, 243, 638, 821, 823, 825, 848, 850, 852, 856, 864, 865, 866, 876, 952, 956, 1047, 1051, 1052, 1053, 1054, 1055, 1064, 1066, 1122, 1206, 1296, 1349, 1428, 1582, 1591, 1592, 1737, 1739, 1743, 1762, 1766, 1810.
temp_str_1: 1762, 1766.
temp_strm: 42, 54, 70, 93, 97, 106, 108, 110, 115, 136, 138, 142, 145, 146, 148, 149, 151, 153, 157, 158, 160, 161, 163, 168, 169, 171, 172, 173, 175, 176, 179, 181, 182, 183, 184, 185, 186, 188, 189, 190, 191, 194, 195, 196, 197, 201, 207, 210, 211,

213, 214, 218, 256, 260, 261, 264, 265, 268, 269, 270, 271, 274, 275, 276, 277, 280, 281, 284, 285, 286, 319, 327, 328, 329, 334, 335, 336, 337, 341, 344, 345, 346, 347, 348, 350, 351, 352, 354, 355, 357, 360, 363, 364, 369, 370, 371, 372, 377, 378, 379, 380, 381, 384, 388, 389, 390, 391, 393, 394, 395, 396, 397, 400, 401, 403, 404, 405, 406, 407, 408, 409, 410, 411, 413, 414, 415, 418, 419, 441, 444, 452, 462, 478, 481, 482, 484, 485, 486, 487, 512, 513, 520, 530, 531, 532, 535, 539, 550, 551, 552, 553, 555, 556, 565, 568, 573, 574, 575, 576, 577, 581, 583, 584, 588, 589, 590, 592, 595, 598, 599, 601, 604, 607, 610, 614, 618, 624, 625, 626, 631, 634, 637, 641, 644, 649, 652, 653, 655, 656, 657, 660, 662, 663, 664, 665, 667, 668, 670, 671, 672, 675, 677, 678, 679, 680, 681, 684, 685, 689, 690, 694, 697, 744, 746, 765, 797, 798, 799, 800, 802, 804, 805, 807, 809, 810, 812, 813, 814, 815, 817, 819, 823, 824, 825, 831, 833, 834, 838, 839, 841, 842, 845, 846, 850, 851, 852, 872, 898, 927, 1024, 1026, 1047, 1061, 1064, 1069, 1120, 1126, 1127, 1133, 1134, 1136, 1137, 1141, 1144, 1206, 1232, 1236, 1238, 1240, 1248, 1279, 1280, 1283, 1296, 1298, 1363, 1365, 1446, 1448, 1468, 1483, 1500, 1517, 1549, 1552, 1553, 1559, 1560, 1561, 1562, 1563, 1564, 1565, 1566, 1582, 1604, 1606, 1629, 1637, 1641, 1645, 1647, 1648, 1649, 1651, 1653, 1654, 1678, 1695, 1698, 1705, 1707, 1708, 1709, 1750, 1753, 1761, 1763, 1766, 1770, 1772, 1775, 1777, 1779, 1782, 1783.
temp_strm_1: 898.
temp_time: 935.
temp_time_val: 714, 728, 729, 1117, 1157, 1158.
temp_time_1: 935.
temp_uint: 1435.
temp_user_id: 92.
temp_username: 79.
temp_val: 341, 346, 347, 348, 354, 355, 357, 364, 389, 390, 391, 453, 454, 455, 456, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 574, 576, 714, 715, 716, 718, 719, 720, 721, 735, 765, 781, 782, 786, 856, 865, 1010, 1016, 1017, 1018, 1077, 1089, 1090, 1091, 1092, 1093, 1094, 1095, 1096, 1120, 1174, 1179, 1180, 1181, 1206, 1217, 1232, 1243, 1244, 1298, 1304, 1305, 1431, 1432, 1737, 1740, 1742, 1743, 1744, 1745, 1763, 1766, 1777, 1778.
temp_val_1: 467, 468, 1174, 1185, 1186, 1199, 1232, 1252, 1253, 1254.
temp_width: 1448.
temporary_file: 419.
temporary_filename: 9, 19, 23, 25, 157, 158, 394, 395, 396, 419, 424, 470, 1633, 1634.
terminate_on_end_input: 522, 1551.
test_len: 1669.
thread_cancel_state: 34, 38, 39, 105, 106.
thread_ctr: 33, 35, 39, 51, 52, 54, 55, 62, 70, 108, 110, 115, 136, 145, 151, 152, 153, 154, 155, 157, 158, 159, 160, 161, 163, 164, 165, 168, 169, 171, 172, 173, 175, 176, 179, 180, 181, 182, 184, 185, 186, 187, 189, 190, 191, 193, 194, 195, 196, 197, 199, 201, 203, 204, 205, 206, 207, 208, 210, 211, 212, 213, 214, 215, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 235, 239, 242, 245, 246, 247, 249, 250, 251, 252, 256, 257, 258, 260, 261, 262, 264, 265, 266, 268, 269, 272, 274, 275, 276, 277, 278, 280, 281, 282, 284, 285, 287, 288, 327, 329, 330, 331, 332, 334, 335, 336, 337, 339, 341, 344, 345, 346, 347, 348, 350, 351, 352, 354, 355, 357, 360, 361, 362, 363, 364, 367, 369, 370, 371, 375, 376, 377, 378, 379, 381, 382, 384, 388, 389, 390, 392, 393, 394, 395, 396, 397, 398, 400, 401, 403, 404, 405, 406, 408, 409, 410, 411, 413, 416, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 430, 434, 435, 436, 437, 438, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 460, 461, 462, 463, 465, 466, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 490, 491, 492, 493, 494, 495, 496, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 520, 530, 565, 568, 573, 592, 595, 598, 601, 604, 607, 614, 618, 624, 631, 634, 637, 641, 644, 649, 652, 655, 660, 669, 675, 684, 689, 694, 697, 764, 765, 926, 927, 1025, 1026, 1140, 1141, 1147, 1149, 1150, 1151, 1450, 1452, 1453, 1471, 1472, 1473, 1474, 1484, 1485, 1486, 1506, 1507, 1508, 1518, 1519, 1520, 1521, 1522, 1523, 1524, 1526, 1527, 1528, 1529, 1530, 1531, 1532, 1533, 1539, 1540, 1549, 1550, 1552, 1553, 1558, 1559, 1560, 1561, 1562, 1563, 1564, 1565, 1566, 1567, 1569, 1570, 1571, 1629, 1637, 1639, 1641, 1645, 1647, 1648, 1649, 1651, 1655, 1760, 1761.
thread_ctr_id_map: 1449, 1450, 1451, 1452, 1453, 1472, 1473, 1485, 1486, 1507, 1508.
thread_ctr_id_map_mutex: 1449, 1450, 1452, 1453.
thread_ctr_mutex: 1471, 1472, 1484, 1485, 1506, 1507.
thread_ctr_str: 62, 63, 64, 65, 66, 67, 68, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 91, 92, 94, 95, 96, 97, 98, 99, 100, 102, 107, 115, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 136, 138, 139, 140, 141, 142, 145, 146, 147, 148, 149, 478, 479, 480, 481, 482, 484, 485, 486, 487, 488, 713,

714, 715, 716, 717, 718, 719, 720, 721, 722, 723,
724, 725, 726, 727, 728, 729, 734, 735, 736, 737,
738, 739, 740, 741, 742, 765, 783, 784, 785, 786,
788, 1404, 1661, 1662, 1663, 1664, 1665, 1666,
1667, 1668, 1669, 1670, 1671, 1723.
thread_id_ctr_map: 1450, 1452, 1453, 1473, 1486,
1508.
thread_str: 110, 111, 112, 113, 300, 441, 520, 521,
523, 524, 527, 528, 530, 531, 532, 534, 535, 536,
537, 538, 540, 541, 550, 552, 553, 554, 555, 556,
557, 565, 566, 567, 568, 569, 570, 573, 574, 576,
577, 578, 579, 580, 583, 584, 585, 586, 587, 589,
590, 591, 592, 593, 594, 595, 596, 597, 598, 599,
600, 601, 602, 603, 604, 605, 606, 607, 608, 609,
612, 613, 614, 615, 616, 617, 618, 620, 621, 622,
623, 624, 626, 627, 628, 629, 630, 631, 632, 633,
634, 635, 636, 637, 638, 639, 640, 641, 642, 643,
644, 645, 647, 648, 649, 650, 651, 652, 653, 654,
655, 656, 657, 658, 659, 660, 662, 663, 664, 665,
667, 668, 670, 671, 673, 674, 675, 677, 678, 679,
680, 681, 682, 683, 684, 685, 686, 687, 688, 689,
690, 691, 692, 693, 694, 695, 696, 697, 796, 797,
798, 799, 800, 801, 802, 803, 804, 805, 806, 807,
808, 809, 810, 812, 813, 814, 815, 817, 818, 819,
820, 826, 827, 828, 830, 831, 833, 834, 835, 837,
838, 839, 840, 841, 845, 846, 847, 853, 855, 856,
857, 859, 860, 861, 862, 863, 864, 865, 866, 867,
868, 887, 927, 928, 929, 930, 931, 932, 933, 934,
935, 937, 938, 939, 940, 941, 942, 951, 952, 953,
954, 955, 956, 959, 960, 961, 962, 963, 964, 965,
966, 967, 968, 969, 970, 971, 972, 973, 974, 975,
976, 977, 978, 979, 980, 981, 982, 983, 984, 985,
986, 987, 988, 989, 990, 991, 1026, 1027, 1028,
1029, 1030, 1031, 1032, 1033, 1153, 1154, 1155,
1157, 1158, 1159, 1160, 1161, 1162, 1163, 1165,
1166, 1167, 1169, 1170, 1171, 1172, 1173, 1174,
1175, 1176, 1178, 1179, 1180, 1181, 1182, 1183,
1184, 1185, 1186, 1188, 1189, 1191, 1193, 1194,
1195, 1196, 1197, 1199, 1200, 1201, 1202, 1203,
1204, 1205, 1206, 1207, 1208, 1209, 1210, 1211,
1212, 1213, 1214, 1215, 1216, 1217, 1218, 1219,
1221, 1222, 1223, 1224, 1225, 1226, 1227, 1229,
1230, 1231, 1232, 1233, 1234, 1235, 1237, 1239,
1240, 1241, 1242, 1243, 1244, 1245, 1246, 1248,
1249, 1250, 1251, 1252, 1253, 1254, 1255, 1629,
1630, 1631, 1632, 1634, 1635, 1637, 1643, 1645,
1656, 1723, 1724, 1725, 1726, 1728, 1729, 1730,
1761, 1763, 1764, 1765, 1766, 1767, 1768, 1769,
1770, 1771, 1772, 1773, 1774, 1775, 1776, 1777,
1778, 1779, 1780, 1781, 1782, 1783, 1784, 1785.
time: 131, 467, 798, 922, 923, 929, 935, 1000,
1006, 1042, 1044, 1125, 1151, 1157, 1431, 1432,
1599, 1606, 1607, 1681, 1732, 1735, 1746.
time_set: 381, 996, 998, 1000, 1004, 1006, 1008,
1024, 1030, 1055, 1072, 1116, 1117.
time_spec: 1738, 1739, 1740, 1741, 1742, 1743.
time_str: 1678, 1682, 1683.
timeout: 44.
times: 1731.
timespec: 228.
timestamp: 581, 752, 755, 759, 774, 791, 870, 872,
876, 1599, 1606, 1678, 1683, 1684, 1695, 1700,
1701, 1702, 1736, 1737.
timezone: 1734, 1735.
TLS_SESSION_CACHE: 1370, 1374, 1378, 1380,
1382, 1402, 1500.
tm: 133, 744, 765, 792, 1024, 1042, 1044, 1047,
1055, 1122, 1436, 1697, 1732, 1735, 1737.
tmp: 133, 744, 792, 793, 1042, 1044, 1122, 1125,
1384, 1436, 1438, 1697, 1732, 1733, 1735.
tmp_ptr: 133, 744, 1436, 1437, 1438.
tmp_tm: 765, 781, 782, 786.
TODO: 34, 142, 167, 210, 226, 228, 400, 402,
445, 472, 522, 666, 702, 816, 824, 851, 1338,
1445, 1484, 1731, 1745, 1769.
tolower: 526.
top: 1292.
topbuf: 1482, 1483, 1500, 1501.
toupper: 1811, 1812.
trace_value: 1758.
true: 39, 40, 46, 47, 50, 51, 52, 62, 69, 70, 71, 110,
115, 134, 135, 136, 145, 151, 167, 192, 201, 203,
218, 223, 225, 227, 229, 230, 234, 235, 236, 239,
245, 246, 249, 250, 251, 253, 256, 274, 327, 340,
341, 369, 384, 385, 391, 392, 396, 409, 413, 418,
423, 429, 433, 441, 444, 446, 448, 456, 457, 461,
471, 474, 478, 490, 491, 513, 520, 528, 530, 550,
552, 565, 568, 573, 574, 589, 590, 592, 595, 598,
601, 604, 607, 614, 618, 624, 631, 634, 635, 637,
641, 644, 649, 652, 653, 655, 660, 668, 675,
684, 689, 694, 697, 714, 735, 757, 759, 761,
763, 765, 790, 797, 798, 818, 825, 831, 845,
852, 856, 870, 873, 874, 892, 894, 900, 902,
919, 923, 927, 929, 930, 932, 944, 949, 952,
960, 965, 966, 977, 978, 982, 1006, 1009, 1010,
1012, 1026, 1029, 1042, 1044, 1046, 1047, 1063,
1067, 1068, 1077, 1090, 1091, 1092, 1120, 1133,
1140, 1141, 1142, 1149, 1150, 1151, 1154, 1161,
1174, 1176, 1206, 1224, 1230, 1232, 1273, 1277,
1279, 1283, 1284, 1287, 1292, 1296, 1298, 1307,
1314, 1325, 1327, 1346, 1355, 1389, 1393, 1398,
1400, 1403, 1404, 1416, 1428, 1446, 1447, 1455,
1462, 1471, 1481, 1484, 1486, 1499, 1503, 1506,
1508, 1517, 1518, 1520, 1524, 1526, 1529, 1530,

1531, 1532, 1549, 1551, 1560, 1561, 1562, 1563, 1565, 1566, 1567, 1580, 1581, 1584, 1585, 1587, 1588, 1591, 1592, 1594, 1595, 1597, 1600, 1601, 1602, 1603, 1607, 1609, 1610, 1613, 1614, 1615, 1616, 1620, 1629, 1637, 1645, 1662, 1677, 1678, 1695, 1724, 1729, 1732, 1734, 1735, 1739, 1746, 1750, 1753, 1759, 1761, 1770, 1782, 1783, 1784, 1785, 1790, 1791, 1802, 1803, 1831.

ttimestamp: 789, 790, 791.

ttl: 581, 752, 755, 759, 773, 791, 870, 872, 876.

ttl_type: 581, 752, 755, 759, 772, 791, 870, 872, 876.

ttl: 789, 790, 791.

ttl_type: 789, 790, 791.

tttype: 789, 790, 791, 882, 899, 900.

turn_off_value: 1758, 1759.

turn_on_value: 1758, 1759.

tv_nsec: 228.

tv_sec: 228.

type: 9, 13, 19, 23, 25, 93, 151, 171, 200, 201, 213, 292, 293, 294, 327, 338, 341, 372, 377, 384, 405, 429, 430, 433, 438, 441, 462, 463, 464, 535, 539, 551, 552, 553, 555, 556, 565, 568, 573, 574, 575, 592, 595, 598, 601, 604, 607, 614, 618, 624, 631, 634, 637, 641, 644, 645, 646, 649, 652, 655, 656, 657, 660, 675, 684, 685, 689, 690, 694, 697, 752, 759, 770, 787, 791, 794, 798, 799, 800, 801, 802, 803, 805, 807, 809, 810, 812, 813, 814, 815, 817, 819, 823, 832, 833, 834, 835, 837, 839, 841, 842, 845, 846, 850, 855, 856, 865, 866, 870, 872, 876, 882, 900, 917, 919, 921, 922, 923, 931, 943, 944, 945, 946, 951, 952, 953, 956, 959, 960, 961, 1236, 1528, 1535, 1536, 1537, 1539, 1555, 1556, 1558, 1634, 1637, 1645, 1829, 1830.

type_idx_map: 752, 753, 762, 763, 1829.

typename_map: 9, 10, 21, 25, 565, 568, 573, 592, 595, 598, 601, 604, 607, 614, 618, 624, 631, 634, 637, 641, 644, 645, 646, 649, 652, 655, 660, 675, 684, 689, 694, 697, 1528, 1537, 1556, 1634.

u: 999, 1000, 1005, 1006.

ui_priv: 91.

UINT_MAX: 1471, 1472, 1484, 1485, 1506, 1507.

ULONG_MAX: 176, 346, 347, 348, 354, 355, 357, 364, 437, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 518, 715, 716, 718, 719, 720, 721, 781, 782, 785, 786, 865, 912, 930, 1086, 1089, 1090, 1091, 1092, 1093, 1094, 1095, 1096, 1105, 1112, 1117, 1185, 1199, 1217, 1243, 1244, 1304, 1305, 1435, 1582, 1591, 1592, 1602, 1737, 1740, 1743, 1745, 1754, 1777.

UNDELETE_FILE_TYPE: 9, 10, 57, 58.

undelete_files: 300, 550, 557, 695, 696.

UNDELETE_HANDLE_TYPE: 9, 10, 57, 58.

UNDELETE_HANDLE_VALUE_TYPE: 9, 10, 57, 58.

undelete_handle_values: 504.

UNDELETE_HANDLE_VALUES_PRIVILEGE: 35, 36, 106, 504, 512, 690.

undelete_irods_objects: 556, 1231, 1232, 1256.

unions: 824, 851.

units: 381, 996, 1000, 1004, 1006, 1008, 1024, 1030, 1054, 1072, 1115, 1136.

unlink: 52, 126, 1463, 1683, 1684.

unlock_cerr_mutex: 39, 40, 41, 45, 46, 47, 49, 51, 52, 54, 55, 62, 63, 64, 65, 66, 67, 68, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 110, 111, 112, 113, 115, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 136, 138, 139, 140, 141, 142, 145, 146, 147, 148, 149, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 193, 194, 195, 196, 197, 199, 201, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 235, 237, 239, 240, 241, 242, 244, 245, 246, 247, 248, 249, 250, 251, 252, 256, 257, 258, 260, 261, 262, 264, 265, 266, 268, 269, 270, 271, 272, 274, 275, 276, 277, 278, 280, 281, 282, 284, 285, 287, 288, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 339, 341, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 379, 381, 382, 384, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 400, 401, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 429, 430, 431, 433, 434, 435, 436, 437, 438, 439, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 460, 461, 462, 463, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 484, 485, 486, 487, 488, 490, 491, 492, 493, 494, 495, 496, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 530, 531, 532, 534, 535, 536, 537, 538, 540, 541, 550, 552, 553, 554, 555, 556, 557, 565, 566, 567, 568, 569, 570, 573, 574, 576, 577, 578, 579, 580, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 612, 613, 614, 615, 616,

617, 618, 620, 621, 622, 623, 624, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 662, 663, 664, 665, 667, 668, 670, 671, 673, 674, 675, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 735, 736, 737, 738, 739, 740, 741, 742, 757, 759, 761, 763, 765, 771, 781, 782, 783, 784, 785, 786, 788, 790, 791, 792, 793, 795, 797, 798, 799, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 812, 813, 814, 815, 816, 817, 818, 819, 820, 826, 827, 828, 829, 831, 833, 834, 835, 837, 838, 839, 840, 841, 844, 845, 846, 847, 853, 854, 856, 857, 859, 860, 861, 862, 863, 864, 865, 866, 867, 868, 874, 875, 876, 877, 878, 879, 880, 881, 892, 894, 900, 902, 903, 904, 905, 906, 907, 908, 909, 910, 911, 912, 913, 914, 915, 916, 917, 919, 920, 921, 922, 924, 925, 927, 928, 929, 930, 931, 932, 933, 934, 935, 937, 938, 939, 940, 941, 942, 944, 946, 949, 952, 953, 954, 955, 956, 957, 960, 961, 962, 963, 964, 966, 967, 968, 969, 970, 971, 972, 973, 974, 975, 976, 978, 979, 980, 982, 983, 984, 985, 986, 987, 988, 989, 990, 991, 1006, 1010, 1011, 1012, 1013, 1014, 1015, 1016, 1017, 1018, 1020, 1021, 1022, 1026, 1027, 1028, 1029, 1030, 1031, 1032, 1033, 1042, 1044, 1046, 1048, 1049, 1050, 1051, 1052, 1053, 1054, 1055, 1056, 1057, 1063, 1065, 1066, 1067, 1068, 1069, 1070, 1071, 1072, 1073, 1074, 1075, 1077, 1079, 1080, 1081, 1082, 1083, 1084, 1085, 1086, 1087, 1088, 1089, 1090, 1091, 1092, 1093, 1094, 1095, 1096, 1098, 1099, 1100, 1101, 1102, 1103, 1104, 1105, 1106, 1107, 1108, 1109, 1110, 1111, 1112, 1113, 1114, 1115, 1116, 1117, 1118, 1120, 1121, 1123, 1124, 1125, 1126, 1127, 1128, 1129, 1130, 1131, 1133, 1134, 1135, 1136, 1137, 1138, 1139, 1141, 1142, 1143, 1144, 1145, 1147, 1148, 1150, 1151, 1152, 1154, 1155, 1156, 1157, 1158, 1159, 1160, 1161, 1162, 1163, 1164, 1165, 1166, 1167, 1168, 1169, 1170, 1171, 1172, 1174, 1175, 1176, 1177, 1178, 1179, 1180, 1181, 1182, 1183, 1184, 1185, 1186, 1188, 1189, 1191, 1193, 1194, 1195, 1196, 1197, 1198, 1199, 1200, 1201, 1202, 1203, 1204, 1206, 1207, 1208, 1209, 1210, 1211, 1212, 1213, 1214, 1215, 1216, 1217, 1218, 1219, 1220, 1221, 1222, 1225, 1226, 1227, 1228, 1230, 1232, 1233, 1234, 1235, 1236, 1237, 1239, 1241, 1242, 1243, 1244, 1245, 1246, 1247, 1248, 1249, 1250, 1251, 1252, 1253, 1254, 1255, 1273, 1277, 1278, 1279, 1280, 1281, 1283, 1284, 1285, 1287, 1288, 1289, 1290, 1292, 1293, 1296, 1298, 1299, 1300, 1301, 1302, 1303, 1304, 1305, 1306, 1307, 1308, 1309, 1310, 1311, 1312, 1314, 1315, 1316, 1317, 1318, 1325, 1327, 1346, 1348, 1349, 1398, 1400, 1404, 1409, 1428, 1429, 1430, 1431, 1432, 1433, 1434, 1435, 1437, 1438, 1439, 1440, 1446, 1447, 1449, 1450, 1451, 1452, 1453, 1455, 1463, 1464, 1465, 1466, 1467, 1468, 1469, 1470, 1472, 1473, 1481, 1482, 1483, 1485, 1486, 1487, 1499, 1500, 1501, 1502, 1503, 1505, 1507, 1508, 1509, 1518, 1519, 1520, 1521, 1522, 1523, 1524, 1526, 1527, 1528, 1529, 1530, 1531, 1532, 1533, 1534, 1537, 1539, 1540, 1549, 1550, 1551, 1552, 1553, 1556, 1558, 1559, 1560, 1561, 1562, 1563, 1564, 1565, 1566, 1567, 1569, 1570, 1571, 1581, 1583, 1584, 1585, 1586, 1587, 1588, 1589, 1590, 1591, 1592, 1593, 1594, 1595, 1596, 1597, 1598, 1599, 1600, 1601, 1602, 1603, 1606, 1607, 1609, 1610, 1611, 1612, 1613, 1614, 1615, 1616, 1617, 1618, 1620, 1621, 1622, 1623, 1629, 1630, 1631, 1632, 1634, 1635, 1637, 1639, 1641, 1643, 1645, 1646, 1647, 1648, 1649, 1651, 1655, 1656, 1662, 1663, 1664, 1665, 1666, 1667, 1668, 1669, 1670, 1671, 1677, 1678, 1680, 1681, 1682, 1683, 1684, 1685, 1686, 1687, 1688, 1689, 1691, 1692, 1716, 1719, 1724, 1725, 1726, 1727, 1729, 1730, 1731, 1732, 1733, 1735, 1737, 1739, 1740, 1741, 1742, 1743, 1744, 1745, 1746, 1747, 1750, 1751, 1753, 1754, 1755, 1759, 1761, 1763, 1764, 1765, 1766, 1767, 1768, 1770, 1771, 1772, 1773, 1774, 1775, 1776, 1777, 1778, 1779, 1780, 1782, 1783, 1785, 1790, 1802, 1805, 1806, 1807, 1811, 1812, 1813, 1814, 1815, 1816, 1817, 1818, 1819, 1820, 1821, 1822, 1823, 1824, 1825, 1826, 1827, 1828, 1829, 1830, 1831, 1832, 1833, 1834.
unlock_cout_mutex: 525, 1453, 1468, 1470, 1483, 1500, 1501, 1551, 1722, 1831.
UNLOCK_TABLES: 876, 877, 879, 937, 938, 939, 971, 973, 987, 988, 989, 1029, 1030, 1031, 1584, 1585, 1587, 1588, 1591, 1592, 1594, 1595, 1597, 1600, 1601, 1602, 1603, 1607, 1609, 1610, 1611.
UNLOCK_TABLES_AND_EXIT: 1301, 1302, 1303, 1304, 1305, 1309, 1310.
UNLOCK_TABLES_UPDATE: 1125, 1127, 1129, 1130.
UNLOCK_TABLES_WTD: 1067, 1068, 1069, 1071, 1073, 1074.
UNLOCK_TANS_TABLE: 1667, 1671.
unmark_for_deletion: 981.
unmark_handle_for_deletion: 679, 981, 982, 991.
unmark_handle_value_for_deletion: 691, 830, 831, 854.
update: 195, 469, 1119, 1120, 1131.

uppercase: 1606.
user_cert: 34, 63, 67, 92, 106, 511, 1438, 1502.
user_id: 33, 39, 67, 69, 72, 77, 79, 91, 92, 93, 106, 127, 163, 167, 171, 181, 182, 309, 315, 317, 319, 344, 347, 387, 395, 396, 401, 403, 404, 405, 406, 426, 446, 461, 463, 465, 466, 471, 479, 480, 491, 511, 530, 535, 538, 552, 653, 657, 665, 669, 685, 686, 690, 691, 702, 706, 708, 710, 712, 716, 731, 733, 735, 736, 737, 738, 744, 746, 752, 796, 797, 802, 809, 810, 814, 815, 830, 831, 842, 887, 899, 900, 901, 902, 922, 923, 926, 927, 932, 996, 998, 1000, 1004, 1006, 1008, 1011, 1012, 1024, 1038, 1040, 1042, 1044, 1059, 1061, 1069, 1080, 1153, 1154, 1173, 1175, 1176, 1183, 1196, 1264, 1267, 1271, 1273, 1275, 1306, 1338, 1342, 1344, 1346, 1352, 1355, 1361, 1365, 1503, 1526, 1580, 1581, 1584, 1587, 1588, 1589, 1590, 1599, 1606, 1607, 1639, 1640, 1641, 1642, 1643, 1646, 1649, 1650, 1652, 1653, 1654, 1803, 1814, 1815, 1816, 1820, 1825, 1826, 1827, 1831, 1832.
user_id_map: 33, 106, 1642, 1646, 1652.
user_info: 69, 70, 71, 73, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 91, 92, 96, 99, 100, 103, 511, 1012, 1013, 1176, 1177, 1203, 1503, 1504, 1637, 1639, 1641, 1642, 1643, 1645, 1648, 1649, 1650, 1652.
user_info_map: 33, 106, 1642, 1646, 1652.
user_info_ptr: 33, 39, 51, 106, 1646, 1648, 1651, 1652, 1653, 1654, 1655.
User_Info_Type: 33, 69, 70, 103, 106, 305, 306, 309, 310, 311, 314, 315, 317, 319, 320, 511, 702, 1012, 1176, 1503, 1504, 1637, 1640, 1643, 1645, 1646.
user_name: 67, 702, 706, 708, 710, 712, 717, 731, 733, 744, 746, 1338, 1342, 1344, 1346, 1352, 1355, 1361, 1365.
username: 33, 64, 65, 67, 78, 93, 95, 97, 98, 101, 106, 123, 167, 309, 315, 317, 319, 396, 461, 479, 511, 653, 665, 685, 686, 690, 691, 752, 796, 797, 802, 809, 810, 814, 815, 830, 831, 842, 887, 1503, 1580, 1581, 1584, 1589, 1590, 1639, 1641, 1642, 1646, 1647, 1648, 1649, 1652, 1653, 1803, 1817, 1818, 1819, 1820, 1821, 1822, 1826, 1827, 1831.
USRINFTP_H: 321.
utc: 1734, 1735.
UTILFNCS_H: 1787.
uuser_id: 709, 710, 711, 712, 734, 735, 736, 999, 1000, 1005, 1006, 1041, 1042, 1043, 1044, 1343, 1344, 1345, 1346.
uuser_name: 709, 710, 711, 712, 1343, 1344, 1345, 1346.
v: 33, 1038, 1418, 1445, 1446, 1461, 1462, 1480, 1481, 1498, 1499, 1676, 1677.
val: 295, 513, 518, 999, 1000, 1005, 1006, 1229, 1230.
Validity_notAfter: 702, 706, 708, 710, 712, 729, 731, 733, 744, 746, 1438, 1653.
Validity_notBefore: 702, 706, 708, 710, 712, 728, 731, 733, 744, 746, 1437, 1653.
value: 331, 359, 380, 381, 996, 1000, 1004, 1006, 1008, 1013, 1024, 1030, 1053, 1072, 1114, 1136, 1144, 1151, 1230, 1280, 1292, 1307, 1319, 1325, 1331, 1349, 1390, 1406.
value_str: 1404, 1406.
value_tag: 1390, 1406.
vector: 9, 25, 33, 34, 50, 52, 96, 106, 109, 110, 136, 138, 151, 169, 171, 181, 253, 274, 291, 294, 331, 336, 337, 340, 341, 350, 368, 371, 374, 380, 381, 384, 394, 395, 396, 400, 401, 404, 418, 423, 425, 426, 430, 440, 441, 444, 461, 463, 481, 482, 535, 537, 539, 550, 552, 752, 807, 816, 821, 841, 844, 848, 887, 900, 901, 902, 917, 918, 951, 952, 953, 961, 965, 966, 970, 977, 978, 1038, 1056, 1061, 1070, 1072, 1117, 1128, 1136, 1151, 1153, 1154, 1158, 1161, 1164, 1168, 1174, 1188, 1189, 1190, 1201, 1203, 1206, 1209, 1210, 1220, 1229, 1230, 1231, 1232, 1236, 1247, 1248, 1257, 1580, 1581, 1615, 1697, 1704, 1728, 1729, 1730, 1803, 1830, 1831, 1832.
verbose: 297, 512.
verify_certificate: 92, 702, 1427, 1428, 1440, 1502.
VValidity_notAfter: 709, 710, 711, 712.
VValidity_notBefore: 709, 710, 711, 712.
wake_purge_thread: 887, 1153, 1154, 1171.
warnings_occurred: 34, 38, 39, 106, 239, 246, 331, 335, 351, 367, 409, 410, 411, 413, 420, 455, 460, 490, 495, 523, 524, 532, 535, 536, 565, 574, 576, 578, 579, 583, 585, 586, 599, 620, 626, 632, 635, 638, 642, 650, 668, 670, 677, 679, 685, 690, 697, 1448, 1539, 1558, 1639, 1641.
WEXITSTATUS: 54, 184, 185, 186, 189, 190, 191, 408, 1134, 1135, 1137, 1138, 1144, 1145, 1708, 1709, 1766.
WIFEXITED: 54, 184, 189, 408, 1134, 1137, 1144, 1708, 1766.
wrap_db_deinit: 1375, 1376.
wrap_db_delete: 1381, 1382, 1402.
wrap_db_fetch: 1379, 1380, 1402.
wrap_db_init: 1373, 1374, 1500.
wrap_db_store: 1377, 1378, 1402.
write: 251, 486, 526, 527, 871, 872, 1296, 1520, 1563, 1566, 1615, 1616, 1772, 1785.
write_string_length: 1784, 1785.

write_to_database: 194, 396, 399, 408, 450, 468, 474, 476, 873, 874, 881, 1025, 1026, 1033, 1062, 1063, 1075, 1147, 1297, 1298, 1312.
write_to_fifo: 97, 1784, 1785.
write_to_file: 520, 526, 528, 574, 589, 590.
wrote_data: 1549, 1563, 1566, 1567.
X_509_AUTH_TYPE: 35, 36, 1501, 1526.
XML_ErrorString: 1284.
XML_FMT_INT_MOD: 1262, 1284.
XML_GetCurrentLineNumber: 1284.
XML_GetErrorCode: 1284.
XML_LARGE_SIZE: 1262.
XML_Parse: 1284, 1293.
XML_Parser: 1284.
XML_ParserCreate: 1284.
XML_ParserFree: 1284, 1285.
XML_SetCharacterDataHandler: 1284.
XML_SetElementHandler: 1284.
XML_SetUserData: 1284.
XML_STATUS_ERROR: 1284.
XML_USE_MSC_EXTENSIONS: 1262.
XMLCALL: 1286, 1287, 1291, 1292.
x509_cert: 702, 1403, 1404, 1406.
x509_cert_ptr: 702, 1427, 1428, 1434, 1435, 1436, 1437, 1438, 1439.
X509_Cert_Type: 34, 92, 306, 309, 481, 482, 702, 705, 706, 707, 708, 709, 710, 711, 712, 714, 729, 730, 731, 733, 735, 742, 744, 745, 746, 1351, 1352, 1366, 1367, 1403, 1404, 1427, 1428.
x509_Cert_Type: 1352.
X509CERT_H: 749.
yy_buffer_state: 1517, 1522, 1524.
YY_BUFFER_STATE: 1517, 1549.
yy_delete_buffer: 1524.
yy_scan_string: 1522.
yylex: 33.
yylex_init: 1517.
yyparse: 9, 32, 33, 34, 153, 200, 255, 309, 324, 479, 531, 548, 629, 649, 702, 824, 851, 1264, 1319, 1522, 1523, 1524, 1541.
yyrestart: 1522.
yyScan_t: 9, 32, 33, 309, 320, 321, 324, 548, 702, 748, 749, 752, 883, 884, 992, 993, 996, 1038, 1264, 1319, 1517, 1541, 1549, 1573.
yyset_extra: 1517.
yyset_in: 1522.
YYSTYPE: 33.
zone: 33.
zz_buffer_state: 1549, 1567, 1569, 1570, 1571.
zz_delete_buffer: 1569, 1570, 1571.
zz_scan_string: 1567.
zzdebug: 1576.

zzlex: 610.
zzlex_init: 1549.
zzparse: 33, 34, 752, 824, 851, 996, 1038, 1530, 1531, 1567, 1568, 1573.
zzrestart: 1567.
zzset_extra: 1549.
zzset_in: 1522, 1567.

⟨ Declare functions 1373, 1375, 1377, 1379, 1381, 1383, 1386, 1397, 1399, 1401, 1415, 1419 ⟩ Used in sections 1409, 1410, 1423, and 1424.
⟨ Declare **class Handle_Type** 887 ⟩ Used in sections 992 and 993.
⟨ Declare **class Handle_Value_Type** 752 ⟩ Used in section 884.
⟨ Declare **class Irods_AVU_Type** 996 ⟩ Cited in section 1034. Used in section 1035.
⟨ Declare **class Irods_Object_Type** 1038 ⟩ Used in section 1259.
⟨ Declare **struct Handle_Value_Triple** 882 ⟩ Used in section 884.
⟨ Declare **struct Initialize_Exception_Type** 4 ⟩ Used in sections 5 and 6.
⟨ Declare *verify_certificate* 1427 ⟩ Used in sections 1441 and 1442.
⟨ Define functions 1374, 1376, 1378, 1380, 1382, 1384, 1387, 1398, 1400, 1402, 1416, 1417, 1418, 1420 ⟩ Used in sections 1409 and 1423.
⟨ Define *verify_certificate* 1428, 1429, 1430, 1431, 1432, 1433, 1434, 1435, 1436, 1437, 1438, 1439, 1440 ⟩ Used in section 1441.
⟨ Delete pointers and clear vectors 1257 ⟩ Cited in section 1240. Used in sections 1241, 1243, 1244, 1245, 1248, 1249, 1250, 1252, 1254, and 1256.
⟨ External function declarations 32, 324, 548, 1541, 1573 ⟩ Used in sections 305, 544, 560, 1544, and 1576.
⟨ File-local variables 1263 ⟩ Used in section 1333.
⟨ Garbage 303, 542, 558, 747, 1407, 1474, 1492, 1510, 1542, 1574 ⟩ Used in sections 305, 544, 560, 748, 1409, 1476, 1494, 1512, 1544, and 1576.
⟨ Global variables 310, 1372, 1396 ⟩ Used in sections 320 and 1409.
⟨ Include files 3, 8, 31, 308, 323, 547, 563, 701, 751, 886, 995, 1037, 1261, 1337, 1369, 1412, 1426, 1444, 1460, 1479, 1497, 1515, 1547, 1579, 1627, 1660, 1675, 1715, 1789, 1801 ⟩ Cited in section 1034. Used in sections 5, 27, 305, 320, 544, 560, 698, 748, 883, 992, 1034, 1258, 1333, 1366, 1409, 1423, 1441, 1457, 1476, 1494, 1512, 1544, 1576, 1624, 1657, 1672, 1712, 1786, 1798, and 1835.
⟨ Initialize **Scan_Parse_Parameter_Type** maps 57, 58, 59, 60 ⟩ Used in section 305.
⟨ Initialize **Scan_Parse_Parameter_Type** static data members 36 ⟩ Used in section 305.
⟨ Initialize **static** constant **Response_Type** member variables 10 ⟩ Used in section 27.
⟨ Initialize **static Dublin-Core-Metadata-Type** data members 1265 ⟩ Used in section 1333.
⟨ Initialize **static Handle_Value_Type** data members 753 ⟩ Used in section 883.
⟨ Mutex function declarations 1718, 1719, 1721, 1722 ⟩ Used in sections 1786 and 1787.
⟨ Parser function declarations 1628, 1636, 1644 ⟩ Used in sections 1657 and 1658.
⟨ Preprocessor definitions 1413 ⟩ Used in section 1423.
⟨ Preprocessor macro definitions 1262, 1370 ⟩ Used in sections 1333, 1409, and 1410.
⟨ Type definitions 1371 ⟩ Used in section 1409.
⟨ *connect.h* 1458 ⟩
⟨ *dcmtddtp.h* 1335 ⟩
⟨ *dstngnmt.h* 1367 ⟩
⟨ *ex_rfc2818.h* 1442 ⟩
⟨ *exchncli.h* 1545 ⟩
⟨ *exchnsrv.h* 1577 ⟩
⟨ *excptntp.h* 6 ⟩
⟨ *gentans.h* 1799 ⟩
⟨ *gntlsfnc.h* 1410 ⟩
⟨ *helper.h* 1424 ⟩
⟨ *hdltype.h* 993 ⟩
⟨ *hdlvltp.h* 884 ⟩
⟨ *irdsavtp.h* 1035 ⟩
⟨ *irdsobtp.h* 1259 ⟩
⟨ *listnlcl.h* 1477 ⟩
⟨ *lstnrmta.h* 1495 ⟩
⟨ *lstnrmtx.h* 1513 ⟩
⟨ *pidfncts.h* 1625 ⟩

```

⟨prsrfncts.h 1658⟩
⟨purgfncts.h 1713⟩
⟨rspnntp.h 29⟩
⟨scprpmtp.h 306⟩
⟨spptfnc1.h 545⟩
⟨spptfnc2.h 561⟩
⟨srvractn.h 699⟩
⟨tanfncts.h 1673⟩
⟨usrinftp.h 321⟩
⟨utilfncts.h 1787⟩
⟨x509cert.h 749⟩
⟨Distinguished_Name_Type::operator≠ definition 1359⟩ Used in section 1366.
⟨Distinguished_Name_Type::operator=(const X509_Cert_Type &) definition 1352⟩ Used in
section 1366.
⟨Distinguished_Name_Type::operator≡ definition 1355, 1357⟩ Used in section 1366.
⟨Distinguished_Name_Type constructor declarations 1341, 1343⟩ Used in section 1338.
⟨Distinguished_Name_Type function declarations 1345, 1351, 1354, 1356, 1358, 1360, 1362, 1364⟩ Used in
section 1338.
⟨Distinguished_Name_Type function definitions 1342, 1344, 1346, 1347, 1348, 1349, 1361, 1363, 1365⟩ Used in
section 1366.
⟨Dublin_Core_Metadata_Sub_Type function declarations 1320, 1322, 1324, 1326, 1328, 1330⟩ Used in
section 1319.
⟨Dublin_Core_Metadata_Sub_Type function definitions 1321, 1323, 1325, 1327, 1329, 1331⟩ Used in
section 1333.
⟨Dublin_Core_Metadata_Type::clear definition 1271⟩ Used in section 1333.
⟨Dublin_Core_Metadata_Type::end definition 1292, 1293, 1294⟩ Used in section 1333.
⟨Dublin_Core_Metadata_Type::handle_data definition 1296⟩ Used in section 1333.
⟨Dublin_Core_Metadata_Type::initialize_maps definition 1277⟩ Used in section 1333.
⟨Dublin_Core_Metadata_Type::operator≡ definition 1273⟩ Used in section 1333.
⟨Dublin_Core_Metadata_Type::output definition 1279, 1280, 1281⟩ Used in section 1333.
⟨Dublin_Core_Metadata_Type::parse definition 1283, 1284, 1285⟩ Used in section 1333.
⟨Dublin_Core_Metadata_Type::set_handle_id definition 1314, 1315, 1316, 1317, 1318⟩ Used in section 1333.
⟨Dublin_Core_Metadata_Type::show definition 1275⟩ Used in section 1333.
⟨Dublin_Core_Metadata_Type::start definition 1287, 1288, 1289, 1290⟩ Used in section 1333.
⟨Dublin_Core_Metadata_Type::write_to_database definition 1298, 1299, 1300, 1301, 1302, 1303, 1304, 1305,
1306, 1307, 1308, 1309, 1310, 1311, 1312⟩ Used in section 1333.
⟨Dublin_Core_Metadata_Type constructor definition 1267⟩ Used in section 1333.
⟨Dublin_Core_Metadata_Type destructor definition 1269⟩ Used in section 1333.
⟨Dublin_Core_Metadata_Type function declarations 1266, 1268, 1270, 1272, 1274, 1276, 1278, 1282, 1286, 1291,
1295, 1297, 1313⟩ Used in section 1264.
⟨Handle_Type::add_values definition 902, 903, 904, 905, 906, 907, 908, 909, 910, 911, 912, 913, 914, 915, 916, 917,
918, 919, 920, 921, 922, 923, 924, 925⟩ Used in section 992.
⟨Handle_Type::add_value definition 900⟩ Used in section 992.
⟨Handle_Type::clear definition 896⟩ Used in section 992.
⟨Handle_Type::created_by_user_id definition 949⟩ Used in section 992.
⟨Handle_Type::delete_from_database definition 927, 928, 929, 930, 931, 932, 933, 934, 935, 936, 937, 938, 939, 940,
941, 942⟩ Used in section 992.
⟨Handle_Type::delete_handle_values definitions 966, 967, 968, 969, 970, 971, 972, 973, 974, 975, 976, 978, 979, 980⟩
Used in section 992.
⟨Handle_Type::fetch_handle_from_database definition 960, 961, 962, 963, 964⟩ Used in section 992.
⟨Handle_Type::fetch_handles_from_database definition 952, 953, 954, 955, 956, 957⟩ Used in section 992.
⟨Handle_Type::find definition 944, 945, 946⟩ Used in section 992.

```

⟨ **Handle_Type** :: **operator=** definition 894 ⟩ Used in section 992.
⟨ **Handle_Type** :: **show** definition 898 ⟩ Used in section 992.
⟨ **Handle_Type** :: *unmark_handle_for_deletion* definition 982, 983, 984, 985, 986, 987, 988, 989, 990, 991 ⟩ Used in section 992.
⟨ **Handle_Type** constructor definitions 890, 892 ⟩ Used in section 992.
⟨ **Handle_Value_Type** :: **clear** definition 870 ⟩ Used in section 883.
⟨ **Handle_Value_Type** :: *delete_handle_value* definition 797, 798, 799, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818, 819, 820, 821, 822, 823, 824, 825, 826, 827, 828, 829 ⟩ Used in section 883.
⟨ **Handle_Value_Type** :: *extract* definition 856, 857, 858, 859, 860, 861, 862, 863, 864, 865, 866, 867, 868 ⟩ Used in section 883.
⟨ **Handle_Value_Type** :: *initialize_maps* definition 763 ⟩ Used in section 883.
⟨ **Handle_Value_Type** :: **operator=** definition 759 ⟩ Used in section 883.
⟨ **Handle_Value_Type** :: *set* definitions 765, 766, 767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786, 787, 788, 790, 791, 792, 793, 794, 795 ⟩ Used in section 883.
⟨ **Handle_Value_Type** :: **show** definition 872 ⟩ Used in section 883.
⟨ **Handle_Value_Type** :: *unmark_handle_value_for_deletion* definition 831, 832, 833, 834, 835, 836, 837, 838, 839, 840, 841, 842, 843, 844, 845, 846, 847, 848, 849, 850, 851, 852, 853, 854 ⟩ Used in section 883.
⟨ **Handle_Value_Type** :: *write_to_database* definition 874, 875, 876, 877, 878, 879, 880, 881 ⟩ Used in section 883.
⟨ **Handle_Value_Type** constructor definitions 755, 757 ⟩ Used in section 883.
⟨ **Handle_Value_Type** destructor definition 761 ⟩ Used in section 883.
⟨ **Handle_Value_Type** function declarations 754, 756, 758, 760, 762, 764, 789, 796, 830, 855, 869, 871, 873 ⟩ Used in section 752.
⟨ **Irods_AVU_Type** :: **clear** definition 1008 ⟩ Used in section 1034.
⟨ **Irods_AVU_Type** :: *delete_irods_avu* definition 1010, 1011, 1012, 1013, 1014, 1015, 1016, 1017, 1018, 1019, 1020, 1021, 1022 ⟩ Used in section 1034.
⟨ **Irods_AVU_Type** :: **operator=** definition 1004 ⟩ Used in section 1034.
⟨ **Irods_AVU_Type** :: *set* definition 1006 ⟩ Used in section 1034.
⟨ **Irods_AVU_Type** :: **show** definition 1024 ⟩ Used in section 1034.
⟨ **Irods_AVU_Type** :: *write_to_database* definition 1026, 1027, 1028, 1029, 1030, 1031, 1032, 1033 ⟩ Used in section 1034.
⟨ **Irods_AVU_Type** constructor definitions 998, 1000, 1002 ⟩ Used in section 1034.
⟨ **Irods_AVU_Type** function declarations 997, 999, 1001, 1003, 1005, 1007, 1009, 1023, 1025 ⟩ Used in section 996.
⟨ **Irods_Object_Type** :: *add_avu_cond* definition 1150, 1151, 1152 ⟩ Used in section 1258.
⟨ **Irods_Object_Type** :: *add_avu* definition 1141, 1142, 1143, 1144, 1145, 1146, 1147, 1148 ⟩ Used in section 1258.
⟨ **Irods_Object_Type** :: *add_handle_value* definition 1224, 1225, 1226, 1227, 1228 ⟩ Used in section 1258.
⟨ **Irods_Object_Type** :: **clear** definition 1059 ⟩ Used in section 1258.
⟨ **Irods_Object_Type** :: *delete_from_archive* definition 1174, 1175, 1176, 1177, 1178, 1179, 1180, 1181, 1182, 1183, 1184, 1185, 1186, 1187, 1188, 1189, 1190, 1191, 1192, 1193, 1194, 1195, 1196, 1197, 1198, 1199, 1200, 1201, 1202, 1203, 1204 ⟩ Used in section 1258.
⟨ **Irods_Object_Type** :: *delete_from_gwirdsf_db* definition 1206, 1207, 1208, 1209, 1210, 1211, 1212, 1213, 1214, 1215, 1216, 1217, 1218, 1219, 1220, 1221, 1222 ⟩ Used in section 1258.
⟨ **Irods_Object_Type** :: *find_avu* definition 1230 ⟩ Used in section 1258.
⟨ **Irods_Object_Type** :: *get_avus_from_irods_system* definition 1046, 1047, 1048, 1049, 1050, 1051, 1052, 1053, 1054, 1055, 1056, 1057 ⟩ Used in section 1258.
⟨ **Irods_Object_Type** :: *get_from_database* definition 1077, 1078, 1079, 1080, 1081, 1082, 1083, 1084, 1085, 1086, 1087, 1088, 1089, 1090, 1091, 1092, 1093, 1094, 1095, 1096, 1097, 1098, 1099, 1100, 1101, 1102, 1103, 1104, 1105, 1106, 1107, 1108, 1109, 1110, 1111, 1112, 1113, 1114, 1115, 1116, 1117, 1118 ⟩ Used in section 1258.
⟨ **Irods_Object_Type** :: *mark_for_deletion* definition 1154, 1155, 1156, 1157, 1158, 1159, 1160, 1161, 1162, 1163, 1164, 1165, 1166, 1167, 1168, 1169, 1170, 1171, 1172 ⟩ Used in section 1258.
⟨ **Irods_Object_Type** :: *put_irods_object* definition 1133, 1134, 1135, 1136, 1137, 1138, 1139 ⟩ Used in section 1258.
⟨ **Irods_Object_Type** :: *set* definitions 1044 ⟩ Used in section 1258.

⟨ **Irods_Object_Type** :: *show* definition 1061 ⟩ Used in section 1258.
⟨ **Irods_Object_Type** :: *undelete_irods_objects* definition 1232, 1233, 1234, 1235, 1236, 1237, 1238, 1239, 1240, 1241, 1242, 1243, 1244, 1245, 1246, 1247, 1248, 1249, 1250, 1251, 1252, 1253, 1254, 1255, 1256 ⟩ Used in section 1258.
⟨ **Irods_Object_Type** :: *update* definition 1120, 1121, 1122, 1123, 1124, 1125, 1126, 1127, 1128, 1129, 1130, 1131 ⟩ Used in section 1258.
⟨ **Irods_Object_Type** :: *write_to_database* definition 1063, 1064, 1065, 1066, 1067, 1068, 1069, 1070, 1071, 1072, 1073, 1074, 1075 ⟩ Used in section 1258.
⟨ **Response_Type** constructor definitions 1040, 1042 ⟩ Used in section 1258.
⟨ **Response_Type** function declarations 1039, 1041, 1043, 1045, 1058, 1060, 1062, 1076, 1119, 1132, 1140, 1149, 1153, 1173, 1205, 1223, 1229, 1231 ⟩ Used in section 1038.
⟨ **Response_Type** :: *operator=* definition 19 ⟩ Used in section 27.
⟨ **Response_Type** constructor definitions 13, 15 ⟩ Used in section 27.
⟨ **Response_Type** destructor definition 17 ⟩ Used in section 27.
⟨ **Response_Type** function declarations 12, 14, 16, 18, 20, 22, 24 ⟩ Used in section 9.
⟨ **Response_Type** function definitions 21, 23, 25 ⟩ Used in section 27.
⟨ **Scan_Parse_Parameter_Type** :: *add_metadata* definition 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416 ⟩ Used in section 544.
⟨ **Scan_Parse_Parameter_Type** :: *cd* definition 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272 ⟩ Used in section 305.
⟨ **Scan_Parse_Parameter_Type** :: *fetch_handle_from_database* definition 429, 430, 431, 433, 434, 435, 436, 437, 438, 439 ⟩ Used in section 544.
⟨ **Scan_Parse_Parameter_Type** :: *fetch_handles_from_database* definition 441, 442, 443 ⟩ Used in section 544.
⟨ **Scan_Parse_Parameter_Type** :: *get_database_username* definition 62, 63, 64, 65, 66, 67, 68 ⟩ Used in section 305.
⟨ **Scan_Parse_Parameter_Type** :: *get_expires* definition 133 ⟩ Used in section 305.
⟨ **Scan_Parse_Parameter_Type** :: *get_handle* definition 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339 ⟩ Used in section 544.
⟨ **Scan_Parse_Parameter_Type** :: *get_highest_value* definition 513, 514, 515, 516, 517, 518, 519 ⟩ Used in section 544.
⟨ **Scan_Parse_Parameter_Type** :: *get_input* definition 520, 521, 522, 523, 524, 525, 526, 527, 528 ⟩ Used in section 544.
⟨ **Scan_Parse_Parameter_Type** :: *get_metadata* definition 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382 ⟩ Used in section 544.
⟨ **Scan_Parse_Parameter_Type** :: *get_privileges* definition 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510 ⟩ Used in section 544.
⟨ **Scan_Parse_Parameter_Type** :: *get_user* definition 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102 ⟩ Used in section 305.
⟨ **Scan_Parse_Parameter_Type** :: *get* definition 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215 ⟩ Used in section 305.
⟨ **Scan_Parse_Parameter_Type** :: *ls* definition 136, 137, 138, 139, 140, 141, 142 ⟩ Used in section 305.
⟨ **Scan_Parse_Parameter_Type** :: *mark_irods_objects_for_deletion* definition 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541 ⟩ Used in section 544.
⟨ **Scan_Parse_Parameter_Type** :: *mkdir* definition 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288 ⟩ Used in section 305.
⟨ **Scan_Parse_Parameter_Type** :: *parse_metadata* definition 418, 419, 420, 421, 422, 423, 424, 425, 426, 427 ⟩ Used in section 544.
⟨ **Scan_Parse_Parameter_Type** :: *put* definition 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199 ⟩ Used in section 305.
⟨ **Scan_Parse_Parameter_Type** :: *pwd* definition 145, 146, 147, 148, 149 ⟩ Used in section 305.

⟨ **Scan_Parser_Parameter_Type** :: *receive_file* definition 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252 ⟩ Used in section 305.

⟨ **Scan_Parser_Parameter_Type** :: *send_tan_list* definition 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129 ⟩ Used in section 305.

⟨ **Scan_Parser_Parameter_Type** :: *send_to_peer* definitions 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 233 ⟩ Used in section 305.

⟨ **Scan_Parser_Parameter_Type** :: *server_action_add_handle_value* definition 655, 656, 657, 658, 659 ⟩ Used in section 698.

⟨ **Scan_Parser_Parameter_Type** :: *server_action_cd* definition 598, 599, 600 ⟩ Used in section 698.

⟨ **Scan_Parser_Parameter_Type** :: *server_action_command_only* definition 565, 566, 567 ⟩ Used in section 698.

⟨ **Scan_Parser_Parameter_Type** :: *server_action_create_handle* definition 652, 653, 654 ⟩ Used in section 698.

⟨ **Scan_Parser_Parameter_Type** :: *server_action_delete_handle_value* definition 684, 685, 686, 687, 688 ⟩ Used in section 698.

⟨ **Scan_Parser_Parameter_Type** :: *server_action_delete_handle* definition 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674 ⟩ Used in section 698.

⟨ **Scan_Parser_Parameter_Type** :: *server_action_end_server* definition 618, 619, 620, 621, 622, 623 ⟩ Used in section 698.

⟨ **Scan_Parser_Parameter_Type** :: *server_action_get_handle* definition 637, 638, 639, 640 ⟩ Used in section 698.

⟨ **Scan_Parser_Parameter_Type** :: *server_action_get_metadata* definition 634, 635, 636 ⟩ Used in section 698.

⟨ **Scan_Parser_Parameter_Type** :: *server_action_get_user_info* definition 649, 650, 651 ⟩ Used in section 698.

⟨ **Scan_Parser_Parameter_Type** :: *server_action_get* definition 607, 608, 609, 610, 611, 612, 613 ⟩ Used in section 698.

⟨ **Scan_Parser_Parameter_Type** :: *server_action_ls* definition 592, 593, 594 ⟩ Used in section 698.

⟨ **Scan_Parser_Parameter_Type** :: *server_action_mark_irods_objects_for_deletion* definition 604, 605, 606 ⟩ Used in section 698.

⟨ **Scan_Parser_Parameter_Type** :: *server_action_mkdir* definition 601, 602, 603 ⟩ Used in section 698.

⟨ **Scan_Parser_Parameter_Type** :: *server_action_process_pending* definition 644, 645, 646, 647, 648 ⟩ Used in section 698.

⟨ **Scan_Parser_Parameter_Type** :: *server_action_pwd* definition 595, 596, 597 ⟩ Used in section 698.

⟨ **Scan_Parser_Parameter_Type** :: *server_action_receive_metadata_file* definition 572 ⟩ Used in section 698.

⟨ **Scan_Parser_Parameter_Type** :: *server_action_receive_put_file* definition 571 ⟩ Used in section 698.

⟨ **Scan_Parser_Parameter_Type** :: *server_action_send_file* definition 568, 569, 570 ⟩ Used in section 698.

⟨ **Scan_Parser_Parameter_Type** :: *server_action_send_handle* definition 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591 ⟩ Used in section 698.

⟨ **Scan_Parser_Parameter_Type** :: *server_action_send_metadata* definition 614, 615, 616, 617 ⟩ Used in section 698.

⟨ **Scan_Parser_Parameter_Type** :: *server_action_send_tan_list* definition 641, 642, 643 ⟩ Used in section 698.

⟨ **Scan_Parser_Parameter_Type** :: *server_action_show_certificate* definition 631, 632, 633 ⟩ Used in section 698.

⟨ **Scan_Parser_Parameter_Type** :: *server_action_sleep* definition 624, 625, 626, 627, 628, 629, 630 ⟩ Used in section 698.

⟨ **Scan_Parser_Parameter_Type** :: *server_action_undelete_file* definition 694, 695, 696 ⟩ Used in section 698.

⟨ **Scan_Parser_Parameter_Type** :: *server_action_undelete_handle_value* definition 689, 690, 691, 692, 693 ⟩ Used in section 698.

⟨ **Scan_Parser_Parameter_Type** :: *server_action_undelete_handle* definition 675, 676, 677, 678, 679, 680, 681, 682, 683 ⟩ Used in section 698.

⟨ **Scan_Parser_Parameter_Type** :: *server_action_unknown* definition 697 ⟩ Used in section 698.

⟨ **Scan_Parser_Parameter_Type** :: *set_expires* definition 131 ⟩ Used in section 305.

⟨ **Scan_Parser_Parameter_Type** :: *set_user_info* definition 511 ⟩ Used in section 544.

⟨ **Scan_Parser_Parameter_Type** :: *show_certificates* definition 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488 ⟩ Used in section 544.

⟨ **Scan_Parser_Parameter_Type** :: *show_privileges* definition 512 ⟩ Used in section 544.

⟨ **Scan_Parser_Parameter_Type** :: *show* definition 106 ⟩ Used in section 305.

⟨ **Scan_Parse_Parameter_Type** :: *store_dc_metadata* definition 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477 ⟩ Used in section 544.

⟨ **Scan_Parse_Parameter_Type** :: *submit_mysql_query* definition 108, 110, 111, 112, 113 ⟩ Used in section 305.

⟨ **Scan_Parse_Parameter_Type** :: *undelete_files* definition 550, 551, 552, 553, 554, 555, 556, 557 ⟩ Used in section 560.

⟨ **Scan_Parse_Parameter_Type** constructor definitions 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49 ⟩ Used in section 305.

⟨ **Scan_Parse_Parameter_Type** destructor definition 51, 52, 53, 54, 55 ⟩ Used in section 305.

⟨ **Scan_Parse_Parameter_Type** function declarations 38, 50, 56, 61, 69, 103, 104, 105, 107, 109, 114, 130, 132, 135, 144, 150, 200, 217, 232, 234, 253, 254, 255, 273, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302 ⟩ Used in section 35.

⟨ **User_Info_Type** :: *clear* definition 317 ⟩ Used in section 320.

⟨ **User_Info_Type** :: *show* definition 319 ⟩ Used in section 320.

⟨ **User_Info_Type** constructor declarations 314 ⟩ Used in section 309.

⟨ **User_Info_Type** constructor definitions 315 ⟩ Used in section 320.

⟨ **User_Info_Type** function declarations 316, 318 ⟩ Used in section 309.

⟨ **X509_Cert_Type** :: *clear* definition 733 ⟩ Used in section 748.

⟨ **X509_Cert_Type** :: *get_from_database* definition 735, 736, 737, 738, 739, 740, 741, 742 ⟩ Used in section 748.

⟨ **X509_Cert_Type** :: *operator=* definition 731 ⟩ Used in section 748.

⟨ **X509_Cert_Type** :: *output* definition 746 ⟩ Used in section 748.

⟨ **X509_Cert_Type** :: *set* definitions 712, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729 ⟩ Used in section 748.

⟨ **X509_Cert_Type** :: *show* definition 744 ⟩ Used in section 748.

⟨ **X509_Cert_Type** constructor definitions 706, 708, 710 ⟩ Used in section 748.

⟨ **X509_Cert_Type** function declarations 705, 707, 709, 711, 713, 730, 732, 734, 743, 745 ⟩ Used in section 702.

⟨ *check_irods_server* declaration 1760 ⟩ Used in sections 1786 and 1787.

⟨ *check_irods_server* definition 1761, 1762, 1763, 1764, 1765, 1766, 1767, 1768 ⟩ Used in section 1786.

⟨ *check_prefix* declaration 1619 ⟩ Used in sections 1624 and 1625.

⟨ *check_prefix* definition 1620, 1621, 1622, 1623 ⟩ Used in section 1624.

⟨ **class Distinguished_Name_Type** declaration 1338 ⟩ Used in sections 1366 and 1367.

⟨ **class Dublin_Core_Metadata_Sub_Type** declaration 1319 ⟩ Used in sections 1333 and 1335.

⟨ **class Dublin_Core_Metadata_Type** declaration 1264 ⟩ Used in sections 1333 and 1335.

⟨ **class Handle_Type** function declarations 889, 891, 893, 895, 897, 899, 901, 926, 943, 948, 951, 959, 965, 977, 981 ⟩ Used in section 887.

⟨ **class Response_Type** declaration 9 ⟩ Used in section 29.

⟨ **class Scan_Parse_Parameter_Type** declaration 33, 34, 35 ⟩ Used in sections 305 and 306.

⟨ **class User_Info_Type** declaration 309 ⟩ Used in sections 320 and 321.

⟨ **class X509_Cert_Type** declaration 702 ⟩ Used in sections 748 and 749.

⟨ *cleanup_handler* declaration 1454 ⟩ Used in sections 1457 and 1458.

⟨ *cleanup_handler* definition 1455 ⟩ Used in section 1457.

⟨ *client_sending_file_rule_func* definition 1629, 1630, 1631, 1632, 1633, 1634, 1635 ⟩ Used in section 1657.

⟨ *connect_func* declaration 1445 ⟩ Used in sections 1457 and 1458.

⟨ *connect_func* definition 1446, 1447, 1448, 1449, 1450, 1451, 1452, 1453 ⟩ Used in section 1457.

⟨ *convert_seconds* declaration 1734 ⟩ Used in sections 1786 and 1787.

⟨ *convert_seconds* definition 1735 ⟩ Used in section 1786.

⟨ *convert_time_spec* declaration 1738 ⟩ Used in sections 1786 and 1787.

⟨ *convert_time_spec* definition 1739, 1740, 1741, 1742, 1743, 1744, 1745, 1746, 1747 ⟩ Used in section 1786.

⟨ *decrypt* declaration 1769 ⟩ Used in sections 1786 and 1787.

⟨ *decrypt* definition 1770, 1771, 1772, 1773, 1774, 1775, 1776, 1777, 1778, 1779, 1780 ⟩ Used in section 1786.

⟨ *distinguished_name_rule_func* definition 1637, 1638, 1639, 1640, 1641, 1642, 1643 ⟩ Used in section 1657.

⟨ *exchange_data_with_client* declaration 1516 ⟩ Used in sections 1544 and 1545.

⟨ *exchange_data_with_client* definition 1517, 1518, 1519, 1520, 1521, 1522, 1523, 1524, 1525, 1526, 1527, 1528, 1529, 1530, 1531, 1532, 1533, 1534, 1535, 1536, 1537, 1538, 1539, 1540 ⟩ Used in section 1544.

⟨ *exchange_data_with_server* declaration 1548 ⟩ Used in sections 1576 and 1577.

⟨ *exchange_data_with_server* definition 1549, 1550, 1551, 1552, 1553, 1554, 1555, 1556, 1557, 1558, 1559, 1560, 1561, 1562, 1563, 1564, 1565, 1566, 1567, 1568, 1569, 1570, 1571, 1572 ⟩ Used in section 1576.

⟨ **extern** declarations for global variables 311 ⟩ Used in section 321.

⟨ *extract_dn_fields* declaration 1403 ⟩ Used in sections 1409 and 1410.

⟨ *extract_dn_fields* definition 1404, 1405, 1406 ⟩ Used in section 1409.

⟨ *finish* definition 1790, 1802 ⟩ Used in sections 1798 and 1835.

⟨ *generate_dh_params* declaration 1490 ⟩ Used in sections 1494 and 1495.

⟨ *generate_dh_params* definition 1491 ⟩ Used in section 1494.

⟨ *generate_pids* declaration 1580 ⟩ Used in sections 1624 and 1625.

⟨ *generate_pids* definition 1581, 1582, 1583, 1584, 1585, 1586, 1587, 1588, 1589, 1590, 1591, 1592, 1593, 1594, 1595, 1596, 1597, 1598, 1599, 1600, 1601, 1602, 1603, 1604, 1605, 1606, 1607, 1608, 1609, 1610, 1611, 1612, 1613, 1614, 1615, 1616, 1617, 1618 ⟩ Used in section 1624.

⟨ *generate_tans* declaration 1661 ⟩ Used in sections 1672 and 1673.

⟨ *generate_tans* definition 1662, 1663, 1664, 1665, 1666, 1667, 1668, 1669, 1670, 1671 ⟩ Used in section 1672.

⟨ *get_timestamp* declaration 1731 ⟩ Used in sections 1786 and 1787.

⟨ *get_timestamp* definition 1732, 1733 ⟩ Used in section 1786.

⟨ *get_in_addr* declaration 1421 ⟩ Used in sections 1423 and 1424.

⟨ *get_in_addr* definition 1422 ⟩ Used in section 1423.

⟨ *get_seconds_since_epoch* declaration 1736 ⟩ Used in sections 1786 and 1787.

⟨ *get_seconds_since_epoch* definition 1737 ⟩ Used in section 1786.

⟨ *get_user_info_func* definition 1645, 1646, 1647, 1648, 1649, 1650, 1651, 1652, 1653, 1654, 1655, 1656 ⟩ Used in section 1657.

⟨ *hexl_decode* declaration 1752 ⟩ Used in sections 1786 and 1787.

⟨ *hexl_decode* definition 1753, 1754, 1755 ⟩ Used in section 1786.

⟨ *hexl_encode* declaration 1749 ⟩ Used in sections 1786 and 1787.

⟨ *hexl_encode* definition 1750, 1751 ⟩ Used in section 1786.

⟨ *initialize_signal_maps* declaration 1756 ⟩ Used in sections 1786 and 1787.

⟨ *initialize_signal_maps* definition 1757 ⟩ Used in section 1786.

⟨ *initialize_tls_session* declaration 1488 ⟩ Used in sections 1494 and 1495.

⟨ *initialize_tls_session* definition 1489 ⟩ Used in section 1494.

⟨ *listen_local* declaration 1461 ⟩ Used in sections 1476 and 1477.

⟨ *listen_local* definition 1462, 1463, 1464, 1465, 1466, 1467, 1468, 1469, 1470, 1471, 1472, 1473 ⟩ Used in section 1476.

⟨ *listen_remote_X_509* declaration 1498 ⟩ Used in sections 1512 and 1513.

⟨ *listen_remote_X_509* definition 1499, 1500, 1501, 1502, 1503, 1504, 1505, 1506, 1507, 1508, 1509 ⟩ Used in section 1512.

⟨ *listen_remote_anon* declaration 1480 ⟩ Used in sections 1494 and 1495.

⟨ *listen_remote_anon* definition 1481, 1482, 1483, 1484, 1485, 1486, 1487 ⟩ Used in section 1494.

⟨ *main* definition 1791, 1792, 1793, 1794, 1795, 1796, 1797, 1803, 1804, 1805, 1806, 1807, 1809, 1810, 1811, 1812, 1813, 1814, 1815, 1816, 1817, 1818, 1819, 1820, 1821, 1822, 1823, 1824, 1825, 1826, 1827, 1828, 1829, 1830, 1831, 1832, 1833, 1834 ⟩ Used in sections 1798 and 1835.

⟨ *print_x509_certificate_info* declaration 1388 ⟩ Used in sections 1409 and 1410.

⟨ *print_x509_certificate_info* definition 1389, 1390, 1391, 1392, 1393, 1394, 1395 ⟩ Used in section 1409.

⟨ *purge_server_logs* declaration 1676 ⟩ Used in sections 1712 and 1713.

⟨ *purge_server_logs* definition 1677, 1678, 1679, 1680, 1681, 1682, 1683, 1684, 1685, 1686, 1687, 1688, 1689, 1690, 1691, 1692, 1693 ⟩ Used in section 1712.

⟨ *rotate_log_file* declaration 1694 ⟩ Used in sections 1712 and 1713.

⟨ *rotate_log_file* definition 1695, 1696, 1697, 1698, 1699, 1700, 1701, 1702, 1703, 1704, 1705, 1706, 1707, 1708, 1709, 1710, 1711 ⟩ Used in section 1712.

⟨ *set_debug_level* declaration 1758 ⟩ Used in sections 1786 and 1787.

⟨ *set_debug_level* definition 1759 ⟩ Used in section 1786.

⟨ *set_password* declaration 1781 ⟩ Used in sections 1786 and 1787.
⟨ *set_password* definition 1782, 1783 ⟩ Used in section 1786.
⟨ *submit_mysql_queries* declaration 1728 ⟩ Used in sections 1786 and 1787.
⟨ *submit_mysql_queries* definition 1729, 1730 ⟩ Used in section 1786.
⟨ *submit_mysql_query* declaration 1723 ⟩ Used in sections 1786 and 1787.
⟨ *submit_mysql_query* definition 1724, 1725, 1726, 1727 ⟩ Used in section 1786.
⟨ *write_to_fifo* declaration 1784 ⟩ Used in sections 1786 and 1787.
⟨ *write_to_fifo* definition 1785 ⟩ Used in section 1786.

GWDG Archive Interface II:

Version 1.0

by Laurence D. Finston

September 2013

	Section	Page
GWDG iRODS/Handle Interface Client-Server Application	1	1
Exception Types (excptntp.web)	2	1
struct Initialize_Exception_Type	4	2
class Response_Type	7	3
Scan_Parse_Parameter_Type	30	12
class User_Info_Type (usrinftp.web)	307	184
Scan_Parse_Parameter_Type function definitions 1	322	190
Scan_Parse_Parameter_Type function definitions 2	546	350
Scan_Parse_Parameter_Type server action function definitions	562	358
Types for X.509 Certificates (x509cert.web)	700	449
Handle_Value_Type (hdlvltp.web)	750	478
Handle_Type (hdltype.web)	885	558
Irods_AVU_Type (irdsavtp.web)	994	617
Irods_Object_Type (irdsobtp.web)	1036	640
class Irods_Object_Type	1038	642
class Dublin_Core_Metadata_Type	1260	776
class Distinguished_Name_Type (dstngnmt.web)	1336	826
Functions for GNUTLS (gntlsfnc.web)	1368	835
Connect function for server	1443	865
Listen function for local connections	1459	873
Listen function for remote connections	1478	885
Listen function for remote connections with X.509 authentication	1496	896
Exchange data with client	1514	909
Exchange data with server	1546	928
PID functions (pidfncts.web)	1578	949
Functions for use in the parsers	1626	985
TAN functions (tanfncts.web)	1659	1004
Functions for culling and purging	1674	1014
Utility functions (utilfncts.web)	1714	1038
Generate TANs (gentans.web)	1788	1090
Generate PIDs (genpids.web)	1800	1096
Index	1836	1117

1. GWDG iRODS/Handle Interface Client-Server Application.**2. Scanner for server program gwirdsif. [LDF 2012.06.25.]****3.**

⟨ Include files 3 ⟩ ≡

```
#include <stdlib.h>
#include <stdio.h>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <map>
#include <string>
#include <time.h>
#include <math.h>
#include <sstream>
#include <vector>
#include <deque>
#include <stack>
#include <set>
#include <gcrypt.h> /* for gcry_control */
#include <gnutls/gnutls.h>
#include <pthread.h>
#include <expat.h>
#if HAVE_CONFIG_H
#include <config.h>
#endif
#include <mysql.h>
#undef NAME_LEN
#undef LOCAL_HOST
#include "glblcnst.h++"
#include "glblvrbl.h++"
#include "utilfncts.h++"
#include "grouptp.h++"
#include "hndlvtpp.h++"
#include "irdsavtp.h++"
#include "rspnstp.h++"
#include "irdsobtp.h++"
#include "hndltype.h++"
#include "dcmtddtp.h++"
#include "x509cert.h++"
#include "dstngnmt.h++"
#include "parser.h++"
#include "scprpmtp.h++"
```

See also sections 124, 259, and 399.

This code is used in sections 122, 257, 397, and 560.

4. Start conditions.

⟨ Start conditions 4 ⟩ ≡

This code is used in section 122.

5. Options.

$\langle \text{Options } 5 \rangle \equiv$
[`%option_header-file="scanner.hxx"`] [`%option_bison-bridge`] [`%option_reentrant`]

See also sections 126, 260, and 401.

This code is used in sections 122, 257, 397, and 560.

6. Name definitions.

$\langle \text{Name definitions } 6 \rangle \equiv$

This code is used in section 122.

7. Local variables for **yylex**. [LDF 2012.06.25.]

$\langle \text{Local variables for } yylex \ 7 \rangle \equiv$
`bool SCANNER_DEBUG = false; /* true */`

This code is used in section 122.

8. Code to be executed each time **yylex** is entered. This code must be indented or it causes an error when FLEX is run. The start condition on entry to **yylex** can be set here. [LDF 2012.06.27.]

$\langle \text{Execute on entry to } yylex \ 8 \rangle \equiv$

[`Scan_Parse_Parameter_Type* param = static_cast<Scan_Parse_Parameter_Type*>(yyextra);`]

This code is used in section 122.

9. Rules.**10.** Punctuation.

11. Asterisk (*). [LDF 2013.04.25.]

Log

[LDF 2013.04.25.] Added this rule.

```

⟨ Rules 11 ⟩ ≡
  \* {
#if DEBUG_COMPILE
  if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'yylex': " << yytext << "(asterisk)" << endl;
    unlock_cerr_mutex();
  } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
  return ASTERISK_YY; []

```

See also sections 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 130, 131, 133, 134, 136, 137, 138, 139, 140, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 153, 154, 155, 156, 157, 159, 160, 161, 162, 163, 164, 165, 166, 168, 169, 171, 172, 174, 175, 177, 178, 179, 180, 181, 182, 183, 184, 185, 187, 188, 189, 190, 191, 192, 194, 195, 196, 197, 198, 199, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 216, 218, 220, 221, 223, 224, 226, 227, 228, 229, 230, 231, 232, 233, 235, 236, 237, 238, 239, 240, 242, 244, 246, 247, 248, 250, 251, 252, 253, 254, 255, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 407, 408, 410, 411, 412, 413, 414, 415, 416, 417, 418, 420, 421, 423, 424, 425, 426, 427, 429, 430, 431, 432, 433, 434, 435, 436, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 480, 481, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 519, 520, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, and 558.

This code is used in sections 122, 257, 397, and 560.

12. Tilde (~). [LDF 2013.04.25.]

Log

[LDF 2013.04.25.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
  ~ {
#if DEBUG_COMPILE
  if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'yylex': " << yytext << "(tilde)" << endl;
    unlock_cerr_mutex();
  } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
  return TILDE_YY; []

```

13. Dollar sign (\$). [LDF 2013.04.25.]

Log

[LDF 2013.04.25.] Added this rule.

```
<Rules 11> +≡
[\$]
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In `yylex': " << yytext << "(dollar)" << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return DOLLAR YY; []

```

14. Equals sign (\$). [LDF 2013.08.07.]

Log

[LDF 2013.08.07.] Added this rule.

```
<Rules 11> +≡
[=]
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In `yylex': " << yytext << "(equals)" << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return EQUALS YY; []

```

15. Whitespace.

```
<Rules 11> +≡
[[space]\x0d]+[]
```

16. Comments. [LDF 2012.07.04.]

Log

[LDF 2012.07.04.] Added this rule.

```
<Rules 11> +≡
[#.*$]
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In `yylex': Comment: " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */ /* Ignore */
[]
```

17. DISTINGUISHED_NAME. [LDF 2012.07.11.]

Log

[LDF 2012.07.11.] Added this rule.

```
(Rules 11) +≡
[Distinguished_Name|distinguished_name|DISTINGUISHED_NAME]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return DISTINGUISHED_NAME_YY; []
}
```

18. ADD. [LDF 2012.12.14.]

Log

[LDF 2012.12.14.] Added this rule.

```
(Rules 11) +≡
[ADD|add]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return ADD_YY; []
}
```

19. FORCE. [LDF 2012.12.31.]

Log

[LDF 2012.12.31.] Added this rule.

```
(Rules 11) +≡
[--)?(FORCE|force){}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return FORCE_YY; []
}
```

20. FORCE_ADD. [LDF 2013.03.06.]

Log

[LDF 2013.03.06.] Added this rule.

```
(Rules 11) +≡
  (--)?(FORCE-ADD|force-add){}
#ifndef DEBUG_COMPILE
  if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'yylex': " << yytext << endl;
    unlock_cerr_mutex();
  } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
  return FORCE_ADD_YY; []
```

21. FORCE_PUT. [LDF 2013.03.06.]

Log

[LDF 2013.03.06.] Added this rule.

```
(Rules 11) +≡
  (--)?(FORCE-PUT|force-put){}
#ifndef DEBUG_COMPILE
  if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'yylex': " << yytext << endl;
    unlock_cerr_mutex();
  } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
  return FORCE_PUT_YY; []
```

22. FORCE_STORE. [LDF 2013.03.06.]

Log

[LDF 2013.03.06.] Added this rule.

```
(Rules 11) +≡
  (--)?(FORCE-STORE|force-store){}
#ifndef DEBUG_COMPILE
  if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'yylex': " << yytext << endl;
    unlock_cerr_mutex();
  } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
  return FORCE_STORE_YY; []
```

23. FORCE_ALL. [LDF 2013.03.06.]**Log**

[LDF 2013.03.06.] Added this rule.

```

⟨Rules 11⟩ +≡
  ⟨--⟩?(FORCE_ALL|force-all)⟨{
#if DEBUG_COMPILE
  if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'yylex': " << yytext << endl;
    unlock_cerr_mutex();
  } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
  return FORCE_ALL YY; []

```

24. STORE. [LDF 2013.02.28.]**Log**

[LDF 2013.02.28.] Added this rule.

```

⟨Rules 11⟩ +≡
  ⟨--⟩?(STORE|store)⟨{
#if DEBUG_COMPILE
  if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'yylex': " << yytext << endl;
    unlock_cerr_mutex();
  } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
  return STORE YY; []

```

25. FILE. [LDF 2012.06.26.]**Log**

[LDF 2012.06.26.] Added this rule.

```

⟨Rules 11⟩ +≡
  [FILE|file]⟨{
#if DEBUG_COMPILE
  if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'yylex': " << yytext << endl;
    unlock_cerr_mutex();
  } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
  return FILE YY; []

```

26. LOCAL_FILENAME. [LDF 2012.11.19.]**Log**

[LDF 2012.11.19.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
[LOCAL_FILENAME|local_filename]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return LOCAL_FILENAME_YY; []

```

27. REMOTE_FILENAME. [LDF 2012.11.19.]**Log**

[LDF 2012.11.19.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
[REMOTE_FILENAME|remote_filename]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return REMOTE_FILENAME_YY; []

```

28. DIRECTORY. [LDF 2012.06.26.]**Log**

[LDF 2012.06.26.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
[DIRECTORY|directory]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return DIRECTORY_YY; []

```

29. LOCAL. [LDF 2012.06.26.]**Log**

[LDF 2012.06.26.] Added this rule.

```
<Rules 11> +≡
[LOCAL|local]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return LOCAL YY; []
```

30. REMOTE. [LDF 2012.06.26.]**Log**

[LDF 2012.06.26.] Added this rule.

```
<Rules 11> +≡
[REMOTE|remote]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return REMOTE YY; []
```

31. RM. [LDF 2012.06.26.]**Log**

[LDF 2012.06.26.] Added this rule.

```
<Rules 11> +≡
[RM|rm]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return RM YY; []
```

32. CREATE. [LDF 2012.06.26.]

Log

[LDF 2012.06.26.] Added this rule.

```
<Rules 11> +≡
[CREATE|create]{
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return CREATE YY; []
```

33. DELETE. [LDF 2012.06.25.]

Log

[LDF 2012.06.25.] Added this rule.

```
<Rules 11> +≡
[DELETE|delete]{
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return DELETE YY; []
```

34. UNDELETE. [LDF 2013.07.04.]

Log

[LDF 2013.07.04.] Added this rule.

```
<Rules 11> +≡
[UNDELETE|undelete]{
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return UNDELETE YY; []
```

35. DELETION. [LDF 2013.07.04.]**Log**

[LDF 2013.07.04.] Added this rule.

```
<Rules 11> +≡
  [DELETION|deletion]{
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
      lock_cerr_mutex();
      cerr << "In 'yylex': " << yytext << endl;
      unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return DELETION YY; []
}
```

36. IMMEDIATE. [LDF 2013.07.04.]**Log**

[LDF 2013.07.04.] Added this rule.

```
<Rules 11> +≡
  [(--|\+)?(IMMEDIATE|immediate)]{
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
      lock_cerr_mutex();
      cerr << "In 'yylex': " << yytext << endl;
      unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return IMMEDIATE YY; []
}
```

37. MARK. [LDF 2013.07.04.]**Log**

[LDF 2013.07.04.] Added this rule.

```
<Rules 11> +≡
  [MARK|mark]{
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
      lock_cerr_mutex();
      cerr << "In 'yylex': " << yytext << endl;
      unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return MARK YY; []
}
```

38. UNMARK. [LDF 2013.07.04.]**Log**

[LDF 2013.07.04.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
  [UNMARK|unmark]{
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return UNMARK_YY; []
}

```

39. COPY. [LDF 2012.06.26.]**Log**

[LDF 2012.06.26.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
  [COPY|copy]{
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return COPY_YY; []
}

```

40. MOVE. [LDF 2012.06.26.]**Log**

[LDF 2012.06.26.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
  [MOVE|move]{
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return MOVE_YY; []
}

```

41. RENAME. [LDF 2012.06.26.]

Log

[LDF 2012.06.26.] Added this rule.

```
<Rules 11> +≡
[RENAME|rename]{
#endif DEBUG_COMPILE
if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In `yylex': " << yytext << endl;
    unlock_cerr_mutex();
} /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
return RENAME_YY; []
```

42. CHMOD. [LDF 2012.06.26.]

Log

[LDF 2012.06.26.] Added this rule.

```
<Rules 11> +≡
[CHMOD|chmod]{
#endif DEBUG_COMPILE
if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In `yylex': " << yytext << endl;
    unlock_cerr_mutex();
} /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
return CHMOD_YY; []
```

43. SHOW. [LDF 2012.06.27.]

Log

[LDF 2012.06.27.] Added this rule.

```
<Rules 11> +≡
[SHOW|show]{
#endif DEBUG_COMPILE
if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In `yylex': " << yytext << endl;
    unlock_cerr_mutex();
} /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
return SHOW_YY; []
```

44. USER. [LDF 2012.06.27.]

Log

[LDF 2012.06.27.] Added this rule.

```
(Rules 11) +≡
[USER|user]{}
#ifndef DEBUG_COMPILE
if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'yylex': " << yytext << endl;
    unlock_cerr_mutex();
} /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
return USER_YY; []
```

45. INFO. [LDF 2012.06.27.]

Log

[LDF 2012.06.27.] Added this rule.

```
(Rules 11) +≡
[INFO|info]{}
#ifndef DEBUG_COMPILE
if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'yylex': " << yytext << endl;
    unlock_cerr_mutex();
} /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
return INFO_YY; []
```

46. PWD. [LDF 2012.06.27.]

Log

[LDF 2012.06.27.] Added this rule.

```
(Rules 11) +≡
[PWD|pwd]{}
#ifndef DEBUG_COMPILE
if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'yylex': " << yytext << endl;
    unlock_cerr_mutex();
} /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
return PWD_YY; []
```

47. CD. [LDF 2012.06.27.]

Log

[LDF 2012.06.27.] Added this rule.

```
(Rules 11) +≡
[CD|cd]{}
#ifndef DEBUG_COMPILE
if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In `yylex': " << yytext << endl;
    unlock_cerr_mutex();
} /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
return CD_YY; []
```

48. METADATA. [LDF 2012.06.27.]

Log

[LDF 2012.06.27.] Added this rule.

```
(Rules 11) +≡
[METADATA|metadata]{}
#ifndef DEBUG_COMPILE
if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In `yylex': " << yytext << endl;
    unlock_cerr_mutex();
} /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
return METADATA_YY; []
```

49. OUTPUT. [LDF 2012.12.31.]

Log

[LDF 2012.12.31.] Added this rule.

```
(Rules 11) +≡
[(-)?(OUTPUT|output){}
#ifndef DEBUG_COMPILE
if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In `yylex': " << yytext << endl;
    unlock_cerr_mutex();
} /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
return OUTPUT_YY; []
```

50. URI. [LDF 2012.06.27.]**Log**

[LDF 2012.06.27.] Added this rule.

```
<Rules 11> +≡
[URI|uri_]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return URI_YY; []

```

51. PASSWORD. [LDF 2012.06.27.]**Log**

[LDF 2012.06.27.] Added this rule.

```
<Rules 11> +≡
[PASSWORD|password_]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return PASSWORD_YY; []

```

52. SU. [LDF 2012.06.27.]**Log**

[LDF 2012.06.27.] Added this rule.

```
<Rules 11> +≡
[SU|su_]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return SU_YY; []

```

53. SET. [LDF 2012.06.27.]

Log

[LDF 2012.06.27.] Added this rule.

```
(Rules 11) +≡
[SET|set]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return SET_YY; []

```

54. GET. [LDF 2012.06.27.]

Log

[LDF 2012.06.27.] Added this rule.

```
(Rules 11) +≡
[GET|get]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return GET_YY; []

```

55. PUT. [LDF 2012.09.26.]

Log

[LDF 2012.09.26.] Added this rule.

```
(Rules 11) +≡
[PUT|put]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return PUT_YY; []

```

56. SESSION. [LDF 2012.06.27.]**Log**

[LDF 2012.06.27.] Added this rule.

```
<Rules 11> +≡
[SESSION|session_]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return SESSION_YY; []

```

57. LS. [LDF 2012.06.27.]**Log**

[LDF 2012.06.27.] Added this rule.

```
<Rules 11> +≡
[LS|ls_]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return LS_YY; []

```

58. REPOSITORY. [LDF 2012.06.27.]**Log**

[LDF 2012.06.27.] Added this rule.

```
<Rules 11> +≡
[REPOSITORY|repository_]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return REPOSITORY_YY; []

```

59. CONNECT. [LDF 2012.06.27.]

Log

[LDF 2012.06.27.] Added this rule.

```
(Rules 11) +≡
[CONNECT|connect_]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return CONNECT_YY; []

```

60. DISCONNECT. [LDF 2012.06.27.]

Log

[LDF 2012.06.27.] Added this rule.

```
(Rules 11) +≡
[DISCONNECT|disconnect_]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return DISCONNECT_YY; []

```

61. CONNECTION. [LDF 2012.06.27.]

Log

[LDF 2012.06.27.] Added this rule.

```
(Rules 11) +≡
[CONNECTION|connection_]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return CONNECTION_YY; []

```

62. RESET. [LDF 2012.06.27.]**Log**

[LDF 2012.06.27.] Added this rule.

```

⟨Rules 11⟩ +≡
RESET|reset_{
#if DEBUG_COMPILE
if (SCANNER_DEBUG) {
lock_cerr_mutex();
cerr << "In 'yylex': " << yytext << endl;
unlock_cerr_mutex();
} /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
return RESET_YY; []

```

63. WRITE. [LDF 2012.06.27.]**Log**

[LDF 2012.06.27.] Added this rule.

```

⟨Rules 11⟩ +≡
WRITE|write_{
#if DEBUG_COMPILE
if (SCANNER_DEBUG) {
lock_cerr_mutex();
cerr << "In 'yylex': " << yytext << endl;
unlock_cerr_mutex();
} /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
return WRITE_YY; []

```

64. READ. [LDF 2012.06.27.]**Log**

[LDF 2012.06.27.] Added this rule.

```

⟨Rules 11⟩ +≡
READ|read_{
#if DEBUG_COMPILE
if (SCANNER_DEBUG) {
lock_cerr_mutex();
cerr << "In 'yylex': " << yytext << endl;
unlock_cerr_mutex();
} /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
return READ_YY; []

```

65. TO. [LDF 2012.06.27.]

Log

[LDF 2012.06.27.] Added this rule.

```
<Rules 11> +≡
[TO|to]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return TO_YY; []
}
```

66. Time specification. [LDF 2013.08.07.]

Log

[LDF 2013.08.07.] Added this rule.

```
<Rules 11> +≡
[[[:digit:]]*:] {1,3} [[:digit:]]*:? {
    bool save_scanner_debug = SCANNER_DEBUG;
    SCANNER_DEBUG = true; /* false */
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': time specification: " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    string temp_str = yytext;
    unsigned long int delay = convert_time_spec(temp_str);
    if (delay == 0UL || delay == ULONG_MAX) {
        lock_cerr_mutex();
        cerr << "WARNING! In 'yylex': Time specification rule: " << endl <<
            "'convert_time_spec' failed, returning " << delay << "." << endl <<
            "Invalid time specification. Setting 'yylval->ulint_value' to 'delay' <<
            "and will try to continue." << endl;
        unlock_cerr_mutex();
        ++param_warnings_occurred;
    }
    yylval.ulint_value = delay;
    SCANNER_DEBUG = save_scanner_debug;
    return TIME_SPECIFICATION_YY; []
}
```

67. Handle value specification. [LDF 2013.08.30.]

Log

[LDF 2013.08.30.] Added this rule.

```
<Rules 11> +≡
[[[:digit:]]{5}\[[[:alnum:]]\-\_]++[[[:alnum:]]\-\_]+\{\

    bool save_scanner_debug = SCANNER_DEBUG;
    SCANNER_DEBUG = true; /* false */
#endif DEBUG_COMPILE

    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': handle_value_specification: " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
strcpy(yyval->string_value, yytext);
SCANNER_DEBUG = save_scanner_debug;
return HANDLE_VALUE_SPECIFICATION_YY; []
```

68. Integer. [LDF 2012.06.27.]

Log

[LDF 2012.06.27.] Added this rule.

```
<Rules 11> +≡
[-+]?[0-9][0-9]*\{

#endif DEBUG_COMPILE

    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': integer: " << yytext << endl << "Returning 'INTEGER_YY'." << endl;
        unlock_cerr_mutex();
    }
#endif /* DEBUG_COMPILE */
sscanf(yytext, "%d", &yyval->int_value);
return INTEGER_YY; []
```

69. Unsigned Integer. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

```

⟨Rules 11⟩ +≡
[0-9] [0-9]* [uU] {
#if DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': unsigned_integer: " << yytext << endl <<
            "Returning 'UNSIGNED_INTEGER_YY'." << endl;
        unlock_cerr_mutex();
    }
#endif /* DEBUG_COMPILE */
    sscanf(yytext, "%u", &yyval.uint_value);
    return UNSIGNED_INTEGER_YY; []

```

70. Unsigned Long Integer. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

```

⟨Rules 11⟩ +≡
[0-9] [0-9]* [uU] [lL] {
#if DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': unsigned_integer: " << yytext << endl <<
            "Returning 'UNSIGNED_LONG_INTEGER_YY'." << endl;
        unlock_cerr_mutex();
    }
#endif /* DEBUG_COMPILE */
    sscanf(yytext, "%lu", &yyval.ulint_value);
    return UNSIGNED_LONG_INTEGER_YY; []

```

71. SERVER. [LDF 2013.04.19.]

Log

[LDF 2013.04.19.] Added this rule.

```

⟨Rules 11⟩ +≡
[SERVER|server] {
#if DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return SERVER_YY; []

```

72. RESPONSE. [LDF 2013.04.19.]

Log

[LDF 2013.04.19.] Added this rule.

```
(Rules 11) +≡
[RESPONSE|response_{]
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return RESPONSE_YY; []]
```

73. END. [LDF 2012.06.26.]

Log

[LDF 2012.06.26.] Added this rule.

```
(Rules 11) +≡
[END|end_{]
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return END_YY; []]
```

74. END_SERVER. [LDF 2013.04.03.]

Log

[LDF 2013.04.03.] Added this rule.

```
(Rules 11) +≡
[END_SERVER|end_server_{]
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return END_SERVER_YY; []]
```

75. MKDIR. [LDF 2012.06.26.]

Log

[LDF 2012.06.26.] Added this rule.

```
<Rules 11> +≡
[MKDIR|mkdir]{
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return MKDIR YY; []
}
```

76. SEND. [LDF 2012.07.19.]

Log

[LDF 2012.07.19.] Added this rule.

```
<Rules 11> +≡
[SEND|send]{
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return SEND YY; []
}
```

77. SENDING. [LDF 2012.09.27.]

Log

[LDF 2012.09.27.] Added this rule.

```
<Rules 11> +≡
[SENDING|sending]{
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return SENDING YY; []
}
```

78. REFERENCE. [LDF 2012.11.22.]

Log

[LDF 2012.11.22.] Added this rule.

```
<Rules 11> +≡
[REFERENCE|reference]{
#endif DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return REFERENCE_YY; []
```

79. TAN. [LDF 2012.07.19.]

Log

[LDF 2012.07.19.] Added this rule.

```
<Rules 11> +≡
[TAN|tan]{
#endif DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return TAN_YY; []
```

80. LIST. [LDF 2012.07.19.]

Log

[LDF 2012.07.19.] Added this rule.

```
<Rules 11> +≡
[LIST|list]{
#endif DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return LIST_YY; []
```

81. PID. [LDF 2012.07.19.]

Log

[LDF 2012.07.19.] Added this rule.

```
(Rules 11) +≡
[\+?(PID|pid)\{]
#endif DEBUG_COMPILE
if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'yylex': " << yytext << endl;
    unlock_cerr_mutex();
} /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
return PID YY; []
```

82. PIDS. [LDF 2012.10.09.]

Log

[LDF 2012.10.09.] Added this rule.

```
(Rules 11) +≡
[PIDS|pids\{]
#endif DEBUG_COMPILE
if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'yylex': " << yytext << endl;
    unlock_cerr_mutex();
} /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
return PIDS YY; []
```

83. HANDLE. [LDF 2012.10.12.]

Log

[LDF 2012.10.12.] Added this rule.

```
(Rules 11) +≡
[\+?(HANDLE|handle)\{]
#endif DEBUG_COMPILE
if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'yylex': " << yytext << endl;
    unlock_cerr_mutex();
} /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
return HANDLE YY; []
```

84. HANDLES. [LDF 2012.10.12.]

Log

[LDF 2012.10.12.] Added this rule.

```
(Rules 11) +≡
[HANDLES|handles_{]
#ifndef DEBUG_COMPILE
if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'yylex': " << yytext << endl;
    unlock_cerr_mutex();
} /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
return HANDLES_YY; []]
```

85. HANDLE_VALUE. [LDF 2012.10.12.]

Log

[LDF 2012.10.12.] Added this rule.

```
(Rules 11) +≡
[\+?(HANDLE_VALUE|handle_value){]
#ifndef DEBUG_COMPILE
if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'yylex': " << yytext << endl;
    unlock_cerr_mutex();
} /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
return HANDLE_VALUE_YY; []]
```

86. HANDLE_VALUES. [LDF 2012.10.12.]

Log

[LDF 2012.10.12.] Added this rule.

```
(Rules 11) +≡
[HANDLE_VALUES|handle_values_{]
#ifndef DEBUG_COMPILE
if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'yylex': " << yytext << endl;
    unlock_cerr_mutex();
} /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
return HANDLE_VALUES_YY; []]
```

87. IDX. [LDF 2013.06.16.]

Log

[LDF 2013.06.16.] Added this rule.

```

⟨Rules 11⟩ +≡
[\+?(IDX|idx)⊣{
#define DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return IDX_YY; []
}

```

88. TYPE. [LDF 2013.06.16.]

Log

[LDF 2013.06.16.] Added this rule.

```

⟨Rules 11⟩ +≡
[\+?(TYPE|type)⊣{
#define DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return TYPE_YY; []
}

```

89. DATA. [LDF 2013.06.16.]

Log

[LDF 2013.06.16.] Added this rule.

```

⟨Rules 11⟩ +≡
[\+?(DATA|data)⊣{
#define DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return DATA_YY; []
}

```

90. GENERATE. [LDF 2012.09.28.]

Log

[LDF 2012.09.28.] Added this rule.

```
(Rules 11) +≡
[\+?(GENERATE|generate)\u{1}{

#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In\u{'yylex'}:\u{1}" << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return GENERATE_YY; \}
}
```

91. GENERATE (abbreviated). [LDF 2012.09.28.]

Log

[LDF 2012.09.28.] Added this rule.

```
(Rules 11) +≡
[\+?(GEN|gen)\u{1}{

#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In\u{'yylex'}:\u{1}" << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return GENERATE_YY; \}
}
```

92. INSTITUTE. [LDF 2012.09.28.]

Log

[LDF 2012.09.28.] Added this rule.

```
(Rules 11) +≡
[\+?(INSTITUTE|institute)\u{1}{

#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In\u{'yylex'}:\u{1}" << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return INSTITUTE_YY; \}
}
```

93. SUFFIX. [LDF 2012.09.28.]

Log

[LDF 2012.09.28.] Added this rule.

```
<Rules 11> +≡
[\+?(SUFFIX|suffix)\{  

#if DEBUG_COMPILE  

  if (SCANNER_DEBUG) {  

    lock_cerr_mutex();  

    cerr << "In 'yylex': " << yytext << endl;  

    unlock_cerr_mutex();  

  } /* if (SCANNER_DEBUG) */  

#endif /* DEBUG_COMPILE */  

  return SUFFIX YY; []
```

94. PREFIX. [LDF 2012.10.08.]

Log

[LDF 2012.10.08.] Added this rule.

```
<Rules 11> +≡
[\+?(PREFIX|prefix)\{  

#if DEBUG_COMPILE  

  if (SCANNER_DEBUG) {  

    lock_cerr_mutex();  

    cerr << "In 'yylex': " << yytext << endl;  

    unlock_cerr_mutex();  

  } /* if (SCANNER_DEBUG) */  

#endif /* DEBUG_COMPILE */  

  return PREFIX YY; []
```

95. CLIENT. [LDF 2012.07.30.]

Log

[LDF 2012.07.30.] Added this rule.

```
<Rules 11> +≡
[CLIENT|client\{  

#if DEBUG_COMPILE  

  if (SCANNER_DEBUG) {  

    lock_cerr_mutex();  

    cerr << "In 'yylex': " << yytext << endl;  

    unlock_cerr_mutex();  

  } /* if (SCANNER_DEBUG) */  

#endif /* DEBUG_COMPILE */  

  return CLIENT YY; []
```

96. FINISHED. [LDF 2012.07.30.]

Log

[LDF 2012.07.30.] Added this rule.

```
(Rules 11) +≡
[FINISHED|finished]{}
#endif DEBUG_COMPILE
if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'yylex': " << yytext << endl;
    unlock_cerr_mutex();
} /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
return FINISHED_YY; []
```

97. SLEEP. [LDF 2013.04.19.]

Log

[LDF 2013.04.19.] Added this rule.

```
(Rules 11) +≡
[SLEEP|sleep]{}
#endif DEBUG_COMPILE
if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'yylex': " << yytext << endl;
    unlock_cerr_mutex();
} /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
return SLEEP_YY; []
```

98. SIGNAL. [LDF 2013.05.02.]

Log

[LDF 2013.05.02.] Added this rule.

```
(Rules 11) +≡
[SIGNAL|signal]{}
#endif DEBUG_COMPILE
if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'yylex': " << yytext << endl;
    unlock_cerr_mutex();
} /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
return SIGNAL_YY; []
```

99. CERTIFICATE. [LDF 2013.05.03.]

Log

[LDF 2013.05.03.] Added this rule.

```
(Rules 11) +≡
[CERTIFICATE|certificate_{]
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return CERTIFICATE_YY; []
}
```

100. CERTIFICATES. [LDF 2013.05.03.]

Log

[LDF 2013.05.03.] Added this rule.

```
(Rules 11) +≡
[CERTIFICATES|certificates_{]
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return CERTIFICATES_YY; []
}
```

101. FOR. [LDF 2013.05.03.]

Log

[LDF 2013.05.03.] Added this rule.

```
(Rules 11) +≡
[FOR|for_{]
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return FOR_YY; []
}
```

102. ALL. [LDF 2013.05.03.]**Log**

[LDF 2013.05.03.] Added this rule.

```
<Rules 11> +≡
[ALL|all]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return ALL YY; []
}
```

103. GET_USER_INFO. [LDF 2013.05.19.]**Log**

[LDF 2013.05.19.] Added this rule.

```
<Rules 11> +≡
[GET_USER_INFO|get_user_info]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return GET_USER_INFO YY; []
}
```

104. WHOAMI. [LDF 2013.05.03.]**Log**

[LDF 2013.05.03.] Added this rule.

```
<Rules 11> +≡
[WHOAMI|whoami]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return WHOAMI YY; []
}
```

105. PROCESS. [LDF 2013.05.23.]**Log**

[LDF 2013.05.23.] Added this rule.

```
<Rules 11> +≡
[PROCESS|process_{]
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return PROCESS_YY; []
}
```

106. PENDING. [LDF 2013.05.23.]**Log**

[LDF 2013.05.23.] Added this rule.

```
<Rules 11> +≡
[PENDING|pending_{]
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return PENDING_YY; []
}
```

107. OPERATIONS. [LDF 2013.05.23.]**Log**

[LDF 2013.05.23.] Added this rule.

```
<Rules 11> +≡
[OPERATIONS|operations_{]
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return OPERATIONS_YY; []
}
```

108. DELAY. [LDF 2013.05.24.]**Log**

[LDF 2013.05.24.] Added this rule.

```
<Rules 11> +≡
  ((--|\+)?(DELAY|delay)[])
#if DEBUG_COMPILE
  if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In `yylex': " << yytext << endl;
    unlock_cerr_mutex();
  } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
  return DELAY YY; []
```

109. NO_DELAY. [LDF 2013.05.24.]**Log**

[LDF 2013.05.24.] Added this rule.

```
<Rules 11> +≡
  ((--|\+)?(NO [-]DELAY|no [-]delay)[])
#if DEBUG_COMPILE
  if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In `yylex': " << yytext << endl;
    unlock_cerr_mutex();
  } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
  return NO_DELAY YY; []
```

110. DATABASE. [LDF 2013.08.09.]**Log**

[LDF 2013.08.09.] Added this rule.

```
<Rules 11> +≡
  ((--|\+)?(DATABASE|database)[])
#if DEBUG_COMPILE
  if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In `yylex': " << yytext << endl;
    unlock_cerr_mutex();
  } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
  return DATABASE YY; []
```

111. DATABASE_ONLY. [LDF 2013.08.09.]**Log**

[LDF 2013.08.09.] Added this rule.

```

⟨Rules 11⟩ +≡
  (--)|(\+)?(DATABASE[_-]ONLY|database[_-]only) {
#ifndef DEBUG_COMPILE
  if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'yylex': " << yytext << endl;
    unlock_cerr_mutex();
  } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
  return DATABASE_ONLY_YY; []

```

112. GROUP. [LDF 2013.06.04.]**Log**

[LDF 2013.06.04.] Added this rule.

```

⟨Rules 11⟩ +≡
  GROUP|group {
#ifndef DEBUG_COMPILE
  if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'yylex': " << yytext << endl;
    unlock_cerr_mutex();
  } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
  return GROUP_YY; []

```

113. GROUPS. [LDF 2013.06.04.]**Log**

[LDF 2013.06.04.] Added this rule.

```

⟨Rules 11⟩ +≡
  GROUPS|groups {
#ifndef DEBUG_COMPILE
  if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'yylex': " << yytext << endl;
    unlock_cerr_mutex();
  } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
  return GROUPS_YY; []

```

114. DUMMY_STATEMENT. [LDF 2013.04.05.]

Log

[LDF 2013.04.05.] Added this rule.

```
<Rules 11> +≡
  [DUMMY_STATEMENT|dummy_statement]{
#ifndef DEBUG_COMPILE
  if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'yylex': " << yytext << endl;
    unlock_cerr_mutex();
  } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
  return DUMMY_STATEMENT_YY; []
```

115. End-of-file (EOF).

```
<Rules 11> +≡
  [<<EOF>>]{
#ifndef DEBUG_COMPILE
  if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'yylex': Read_EOF" << endl;
    unlock_cerr_mutex();
  } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
  return END_YY; []
```

116. Flag string. [LDF 2012.09.27.]

```
<Rules 11> +≡
  [-[:alpha:]] [--[:alpha:]] [:alpha:]*{
#ifndef DEBUG_COMPILE
  if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'yylex': flag_string: " << yytext << endl;
    unlock_cerr_mutex();
  } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
  string temp_str = yytext;
  strcpy(yyval->string_value, temp_str.c_str());
  return FLAG_YY; []
```

117. Delimited string: "....". [LDF 2012.06.26.]

Log

[LDF 2012.06.26.] Added this rule.

[LDF 2012.09.27.] Now allowing all characters except for " in a delimited string.

[LDF 2013.04.05.] Added code for converting ASCII STX and ETX (“Start of Text” and “End of Text”, respectively) to double quotation marks. Please note that two different characters are both converted to the same character, thus losing some information, albeit a slight amount. It would be possible to convert them to different characters or strings, e.g., ‘ and ’ or ‘‘ and ’’.

```

⟨Rules 11⟩ +≡
[\"[^\"}*\"]
    string temp_str = yytext;
    string::size_type s = temp_str.size();
    temp_str.erase(0, 1);
    temp_str.erase(s - 2, s - 1);
    size_t pos = string::npos;
    do {
        pos = temp_str.find('\002'); /* ASCII STX (“Start of Text”) character */
        if (pos != string::npos) temp_str.replace(pos, 1, 1, '\042'); /* Double quotation mark */
    } while (pos != string::npos);
    pos = string::npos;
    do {
        pos = temp_str.find('\003'); /* ASCII ETX (“End of Text”) character */
        if (pos != string::npos) temp_str.replace(pos, 1, 1, '\042'); /* Double quotation mark */
    } while (pos != string::npos);
#endif DEBUG_COMPILE
if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'yylex': Delimited_string == " << yytext << endl << "Returning 'STRING_YY'." <<
        endl;
    unlock_cerr_mutex();
}
#endif /* DEBUG_COMPILE */
strcpy(yyval->string_value, temp_str.c_str());
return STRING_YY; []

```

118. Undelimited string. [LDF 2012.07.04.]

Log

[LDF 2012.07.04.] Added this rule.

```

⟨Rules 11⟩ +≡
[[[:alnum:] . /; ~@() +\? \$]] [:alnum:] . /; \- _~ +@() \? \$] * { strcpy(yyval->string_value, yytext);
#endif DEBUG_COMPILE
if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'yylex': Undelimited_string == " << yytext << endl <<
        "Returning 'STRING_YY'." << endl;
    unlock_cerr_mutex();
}
#endif /* DEBUG_COMPILE */
return STRING_YY; }

```

119. Other characters. [LDF 2010.05.26.]

Log

[LDF 2010.05.26.] Added this rule.

```

⟨Rules 11⟩ +≡
.{ lock_cerr_mutex();
cerr << "In 'yylex': Other_character: " << yytext << ". Continuing." << endl;
unlock_cerr_mutex(); /* Do nothing. */
}

```

120. Additional functions.

```

⟨yywrap definition 120⟩ ≡
int yywrap(YYSTYPE parameter)
{
    return 1;
}

```

This code is used in section 122.

121.

```

⟨yyerror definition 121⟩ ≡
int yyerror(void *v, const char *s)
{
    return 0;
}

```

This code is used in section 122.

122. Putting scanner together.

```
%{  
⟨ Include files 3 ⟩  
using namespace std;  
using namespace gwrdifpk; %% ⟨ Start conditions 4 ⟩  
⟨ Options 5 ⟩  
⟨ Name definitions 6 ⟩  
%% %% ⟨ Local variables for yylex 7 ⟩  
}% ⟨ Execute on entry to yylex 8 ⟩  
⟨ Rules 11 ⟩  
%% ⟨ yywrap definition 120 ⟩  
⟨ yyerror definition 121 ⟩
```

123. Parser for server program gwirdsif. [LDF 2012.07.10.]

124. Include files.

```
<Include files 3> +≡
#include <stdlib.h>
#include <stdio.h>
#include <stdarg.h>
#include <limits.h>
#include <ctype.h>
#include <signal.h>
#include <algorithm>
#include <bitset>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <iostream>
#include <iterator>
#include <time.h>
#include <math.h>
#include <sstream>
#include <map>
#include <vector>
#include <deque>
#include <stack>
#include <set>
#include <pthread.h>
#if HAVE_CONFIG_H
#include "config.h"
#endif
#include <gcrypt.h> /* for gcry-control */
#include <gnutls/gnutls.h>
#include <gnutls/x509.h>
#include <mysql.h>
#include <expat.h>
#undef NAME_LEN
#undef LOCAL_HOST
#include "rspercds.h++"
#include "glblcnst.h++"
#include "glblvrbl.h++"
#include "utilfncts.h++"
#include "grouptp.h++"
#include "hndlvltp.h++"
#include "irdsavtp.h++"
#include "pidfncts.h++"
#include "tanfncts.h++"
#include "parser.h++"
#include "scanner.h++"
#include "rspnstp.h++"
#include "irdsobtp.h++"
#include "hndltype.h++"
#include "dcmtddtp.h++"
#include "x509cert.h++"
#include "dstngnmt.h++"
#include "usrinftp.h++"
```

```
#include "scprpmtp.h++"
#include "prsrfncts.h++"
```

125. Declarations of additional functions.

$\langle \text{Declarations of additional functions } 125 \rangle \equiv$

```
int yylex(YYSTYPE *lvalp, yyscan_t parameter);
int yywrap(void);
int yyerror(void *v, char const *s);
```

See also section 400.

This code is used in sections 257 and 560.

126. Options.

$\langle \text{Options } 5 \rangle + \equiv$

%verbose
%pure-parser
%parse-param_{yyscan_t_parameter}
%lex-param_{yyscan_t_parameter}
%debug

127. union declaration.

Log

[LDF 2012.09.28.] Added **unsigned int uint_value**.
 [LDF 2012.10.16.] Added **unsigned int ulint_value**.
 [LDF 2013.04.28.] Added **void *pointer_value**.

$\langle \text{union declaration } 127 \rangle \equiv$

%union{
int int_value;
unsigned int uint_value;
unsigned long int ulint_value;
float float_value;
char string_value[1024];
void *pointer_value; }

See also section 402.

This code is used in sections 257 and 560.

128. Token and type declarations.

Log

[LDF 2012.07.11.] Added token declaration for DISTINGUISHED_NAME_YY.
 [LDF 2012.07.19.] Added token declarations for SEND_YY, TAN_YY, LIST_YY and PID_YY.
 [LDF 2012.09.26.] Added token declaration for PUT_YY.
 [LDF 2012.09.27.] Added token declaration for SENDING_YY.
 [LDF 2012.09.28.] Added the token declarations for GENERATE_YY, INSTITUTE_YY and SUFFIX_YY.
 [LDF 2012.10.08.] Added the token declaration for PREFIX_YY.
 [LDF 2012.10.12.] Added the token declarations for HANDLE_YY, HANDLES_YY, HANDLE_VALUE_YY and HANDLE_VALUES_YY.
 [LDF 2012.10.16.] Added token declarations for UNSIGNED_INTEGER_YY and UNSIGNED_LONG_INTEGER_YY.
 [LDF 2012.11.19.] Added the token declaration for LOCAL_FILENAME_YY.
 [LDF 2012.11.21.] Added the token declaration for REMOTE_FILENAME_YY.
 [LDF 2012.11.22.] Added token declaration for REFERENCE_YY.
 [LDF 2012.12.14.] Added token declaration for ADD_YY.
 [LDF 2012.12.31.] Added token declarations for OUTPUT_YY and FORCE_YY.
 [LDF 2013.02.28.] Added token declaration for STORE_YY.
 [LDF 2013.03.06.] Added token declarations for FORCE_ADD_YY, FORCE_PUT_YY, FORCE_STORE_YY and FORCE_ALL_YY.
 [LDF 2013.04.03.] Added token declarations for END_SERVER_YY.
 [LDF 2013.04.05.] Added token declaration for DUMMY_STATEMENT_YY.
 [LDF 2013.04.19.] Added token declarations for SLEEP_YY, SERVER_YY and RESPONSE_YY.
 [LDF 2013.05.02.] Added token declaration for SIGNAL_YY.
 [LDF 2013.05.03.] Added token declarations for CERTIFICATE_YY, CERTIFICATES_YY, FOR_YY and ALL_YY.
 [LDF 2013.05.17.] Added token declaration for WHOAMI_YY.
 [LDF 2013.05.19.] Added token declaration for GET_USER_INFO_YY.
 [LDF 2013.05.23.] Added token declarations for PROCESS_YY, PENDING_YY and OPERATIONS_YY.
 [LDF 2013.05.24.] Added token declarations for DELAY_YY and NO_DELAY_YY.
 [LDF 2013.06.04.] Added token declarations for GROUP_YY and GROUPS_YY.
 [LDF 2013.06.16.] Added token declarations for IDX_YY, TYPE_YY and DATA_YY.
 [LDF 2013.07.04.] Added token declarations for UNDELETE_YY, DELETION_YY, MARK_YY and UNMARK_YY.
 [LDF 2013.07.04.] Added token declaration for IMMEDIATE_YY.
 [LDF 2013.08.07.] Added token declarations for EQUALS_YY and TIME_SPECIFICATION_YY.
 [LDF 2013.08.09.] Added token declarations for DATABASE_YY and DATABASE_ONLY_YY.
 [LDF 2013.08.30.] Added token declaration for HANDLE_VALUE_SPECIFICATION_YY.

< Token and type declarations 128 > ≡

```
% token < int_value > EQUALS_YY
% token < int_value > INTEGER_YY
% token < uint_value > UNSIGNED_INTEGER_YY
% token < ulong_value > UNSIGNED_LONG_INTEGER_YY
% token < ulong_value > TIME_SPECIFICATION_YY
% token < string_value > HANDLE_VALUE_SPECIFICATION_YY % token < int_value > ADD_YY
% token < int_value > ALL_YY
% token < int_value > CLIENT_YY
% token < int_value > END_YY
% token < int_value > FILE_YY
% token < int_value > FINISHED_YY
% token < int_value > FOR_YY
% token < int_value > INFO_YY
% token < int_value > LOCAL_YY
% token < int_value > READ_YY
```

```
% token < int_value > REMOTE_YY
% token < int_value > RESPONSE_YY
% token < int_value > SENDING_YY
% token < int_value > SEND_YY
% token < int_value > SERVER_YY
% token < int_value > SIGNAL_YY
% token < int_value > SLEEP_YY
% token < int_value > TO_YY
% token < int_value > USER_YY
% token < int_value > WRITE_YY
% token < int_value > END_SERVER_YY
% token < int_value > DUMMY_STATEMENT_YY
% token < int_value > DELAY_YY
% token < int_value > NO_DELAY_YY
% token < int_value > DATABASE_YY
% token < int_value > DATABASE_ONLY_YY
% token < int_value > FORCE_YY
% token < int_value > FORCE_ADD_YY
% token < int_value > FORCE_PUT_YY
% token < int_value > FORCE_STORE_YY
% token < int_value > FORCE_ALL_YY
% token < int_value > STORE_YY
% token < string_value > LOCAL_FILENAME_YY
% token < string_value > REMOTE_FILENAME_YY
% token < int_value > DIRECTORY_YY
% token < int_value > MKDIR_YY
% token < int_value > RM_YY
% token < int_value > CREATE_YY
% token < int_value > DELETE_YY
% token < int_value > UNDELETE_YY
% token < int_value > DELETION_YY
% token < int_value > IMMEDIATE_YY
% token < int_value > MARK_YY
% token < int_value > UNMARK_YY
% token < int_value > COPY_YY
% token < int_value > MOVE_YY
% token < int_value > RENAME_YY
% token < int_value > CHMOD_YY
% token < int_value > CHOWN_YY
% token < int_value > SHOW_YY
% token < int_value > PWD_YY
% token < int_value > CD_YY
% token < int_value > METADATA_YY
% token < int_value > OUTPUT_YY
% token < int_value > URI_YY
% token < int_value > PASSWORD_YY
% token < int_value > SU_YY
% token < int_value > SET_YY
% token < int_value > GET_YY
% token < int_value > PUT_YY
% token < string_value > FLAG_YY
% token < int_value > SESSION_YY
```

```
% token < int_value > LS_YY
% token < int_value > REPOSITORY_YY
% token < int_value > CONNECT_YY
% token < int_value > DISCONNECT_YY
% token < int_value > CONNECTION_YY
% token < int_value > RESET_YY
% token < int_value > DISTINGUISHED_NAME_YY
% token < string_value > STRING_YY
% token < int_value > REFERENCE_YY
% token < int_value > TAN_YY
% token < int_value > LIST_YY
% token < int_value > HANDLE_YY
% token < int_value > HANDLES_YY
% token < int_value > HANDLE_VALUE_YY
% token < int_value > HANDLE_VALUES_YY
% token < int_value > PID_YY
% token < int_value > PIDS_YY
% token < int_value > IDX_YY
% token < int_value > TYPE_YY
% token < int_value > DATA_YY
% token < int_value > GENERATE_YY
% token < int_value > INSTITUTE_YY
% token < int_value > PREFIX_YY
% token < int_value > SUFFIX_YY
% token < int_value > CERTIFICATE_YY
% token < int_value > CERTIFICATES_YY
% token < int_value > WHOAMI_YY
% token < int_value > PROCESS_YY
% token < int_value > PENDING_YY
% token < int_value > OPERATIONS_YY
% token < int_value > GET_USER_INFO_YY
% token < int_value > GROUP_YY
% token < int_value > GROUPS_YY
```

See also sections 129, 141, 152, 158, 167, 170, 173, 176, 186, 193, 200, 225, 234, 245, 249, 403, 404, 405, 406, 419, 422, 428, 437, and 501.

This code is used in sections 257 and 560.

129. Punctuation characters. [LDF 2013.04.25.]

Log

[LDF 2013.04.25.] Added this section with the token declarations for ASTERISK_YY, TILDE_YY and DOLLAR_YY. █

⟨ Token and type declarations 128 ⟩ +≡
% token < int_value > ASTERISK_YY
% token < int_value > TILDE_YY
% token < int_value > DOLLAR_YY

130. Rules.

⟨ Rules 11 ⟩ +≡

131. Program.

```
( Rules 11 ) +≡
[program:statement_listEND_YY]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type>(yyget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread_" << param->thread_ctr << "] In 'yparse', rule " <<
            "'program:statement_listEND_YY'. " << endl <<
            "Exiting 'yparse' with return value 0. " << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0;
}
;
```

132. {statement list}. [LDF 2012.06.25.]

133. {statement list} → EMPTY. This rule ensures that an empty file won't cause an error. [LDF 2012.06.25.] ■

```
( Rules 11 ) +≡
[statement_list://*Empty*/]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type>(yyget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread_" << param->thread_ctr << "] In 'yparse', rule " <<
            "'statement_list://*Empty*/'. " << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
}
;
```

134. $\langle \text{statement list} \rangle \rightarrow \langle \text{statement} \rangle$. [LDF 2012.06.25.]

$\langle \text{Rules 11} \rangle +\equiv$

```

statement_list : statement_list statement
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type>(yyget_extra(parameter));
#endif DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " statement_list : statement <<
            endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
}
;
```

135. Statement. [LDF 2012.06.25.]

136. $\langle \text{statement} \rangle \rightarrow \text{END_SERVER_YY}$. [LDF 2013.04.03.]

This rule is needed for profiling the server program `gwirdsif`. The latter is a daemon process and in normal operation is not intended to exit. However, it must exit in order for profiling information to be generated.

This command is disabled by default and enabled by using the `--end-server-enable` option. [LDF 2013.04.03.] ■

Log

[LDF 2013.04.03.] Added this rule.

$\langle \text{Rules 11} \rangle +\equiv$

```

statement : END_SERVER_YY
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type>(yyget_extra(parameter));
#endif DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " statement : END_SERVER_YY << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    Response_Type response;
    if (end_server_enabled) {
        response.type = Response_Type::END_SERVER_TYPE;
    } /* if (end_server_enabled) */
    else {
        response.type = Response_Type::COMMAND_ONLY_TYPE;
        response.command = "END_SERVER_RESPONSE_1";
    }
    param->response_deque.push_back(response);
}
;
```

137. ⟨statement⟩ → DISTINGUISHED_NAME_YY STRING_YY. [LDF 2012.07.11.]

Log

[LDF 2012.07.11.] Added this rule.

[LDF 2013.05.16.] Added code for testing whether the current user has the “show distinguished names” privilege. If it does, the distinguished name data are sent to the client. Otherwise, an error message is sent.

[LDF 2013.05.22.] Removed code from this rule to *distinguished_name_rule_func*, which is defined in `prsrfncts.web`. ■

```

⟨Rules 11⟩ +≡
  statement: DISTINGUISHED_NAME_YY STRING_YY
{
    Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] In 'yparse', rule" <<
            "statement:DISTINGUISHED_NAME_YY STRING_YY' ." << endl << "'STRING_YY'" <<
            "$2" << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    status = distinguished_name_rule_func(param, $2);
    if (status ≡ 2) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] ERROR! In 'yparse', rule" <<
            "'statement:DISTINGUISHED_NAME_YY STRING_YY': " << endl <<
            "'distinguished_name_rule_func' failed, returning 2 (authentication_error)." <<
            endl << "Exiting 'yparse' unsuccessfully with return value 2." << endl;
        unlock_cerr_mutex();
        ++param->errors_occurred;
        return 2;
    }
    else if (status ≠ 0) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] ERROR! In 'yparse', rule" <<
            "'statement:DISTINGUISHED_NAME_YY STRING_YY': " << endl <<
            "'distinguished_name_rule_func' failed, returning" << status << "." <<
            endl << "Continuing." << endl;
        unlock_cerr_mutex();
        ++param->errors_occurred;
    } /* else if (status ≠ 0) */
#endif DEBUG_COMPILE
    else
        if (param->PARSER_DEBUG) {
            lock_cerr_mutex();
            cerr << "[Thread" << param->thread_ctr << "] In 'yparse', rule" <<
                "'statement:DISTINGUISHED_NAME_YY STRING_YY': " << endl <<
                "'distinguished_name_rule_func' succeeded, returning 0." << endl;
        }
}

```

```

        unlock_cerr_mutex();
    } /* else if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
}
;

```

138. ⟨statement⟩ → WHOAMI_YY. [LDF 2013.05.17.]

Log

[LDF 2013.05.17.] Added this rule.

```

⟨Rules 11⟩ +≡
[statement: WHOAMI_YY]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "In 'yparse', rule 'statement:WHOAMI_YY'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    Response_Type response;
    response.type = Response_Type::COMMAND_ONLY_TYPE;
    temp_strm.str("");
    temp_strm << "WHOAMI" << RESPONSE << 0 << USER_ID << param->user_id << "U" << "USER_NAME" << "\n" <<
        param->username << "\n";
    if (param->user_cert.serialNumber > 0) {
        temp_strm << "COMMON_NAME" << param->user_cert.commonName << "\n";
    }
    response.command = temp_strm.str();
    temp_strm.str("");
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "In 'yparse', rule 'statement:WHOAMI_YY': " << endl << "response.command" <== " <<
            endl << response.command << endl;
        param->show("param:");
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->response_deque.push_back(response);
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

139. ⟨statement⟩ → GET_USER_INFO_YY ⟨delay option⟩. [LDF 2013.05.22.]

Log

[LDF 2013.05.22.] Added this rule.

[LDF 2013.05.24.] Added ⟨delay option⟩.

```

⟨Rules 11⟩ +≡
statement: GET_USER_INFO_YY delay_option
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
            << "In 'yparse', rule 'statement: GET_USER_INFO_YY delay_option'." <<
            endl << "'delay_option'" <= <= $2 << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    if ($2 ≡ 0) /* No delay */
    {
        status = get_user_info_func(param);
        if (status ≠ 0) {
            lock_cerr_mutex();
            cerr << "[Thread" << param->thread_ctr << "]"
                << "ERROR! In 'yparse', rule 'statement: GET_USER_INFO_YY delay_option':"
                << endl << "'get_user_info_func' failed, returning" << status << "."
                << endl;
            unlock_cerr_mutex();
            ++param->errors_occurred;
        } /* if (status ≠ 0) */
#ifndef DEBUG_COMPILE
    else
        if (param->PARSER_DEBUG) {
            lock_cerr_mutex();
            cerr << "[Thread" << param->thread_ctr << "]"
                << "In 'yparse', rule 'statement: GET_USER_INFO_YY delay_option':"
                << endl << "'get_user_info_func' succeeded, returning 0."
                << endl;
            unlock_cerr_mutex();
        } /* else if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    } /* if ($2 ≡ 0) (No delay) */
    else /* Delay */
    {
        Response_Type response;
        response.type = Response_Type::GET_USER_INFO_TYPE;
        param->delayed_response_deque.push_back(response);
    } /* else (Delay) */
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}

```

;

140. ⟨statement⟩ → GET_USER_INFO_YY STRING_YY ⟨delay option⟩. [LDF 2013.05.19.]

Log

[LDF 2013.05.19.] Added this rule.
 [LDF 2013.05.22.] Removed code from this rule to *get_user_info_func*, which is defined in *prsrfncs.web*.
 [LDF 2013.05.24.] Added ⟨delay option⟩.

```

⟨Rules 11⟩ +≡
[statement: GET_USER_INFO_YY STRING_YY delay_option]
{
  Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
    *>(yyget_extra(parameter));
  bool save_PARSER_DEBUG = param->PARSER_DEBUG;
  param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
  if (param->PARSER_DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread]" << param->thread_ctr << "] " <<
      "In 'yparse', rule 'statement: GET_USER_INFO_YY STRING_YY delay_option'." <<
      endl << "'delay_option'" <= " " << [$3] << endl;
    unlock_cerr_mutex();
  } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
  if ([\$3] == 0) /* No delay */
  {
    status = get_user_info_func(param, [\$2]);
    if (status != 0) {
      lock_cerr_mutex();
      cerr << "[Thread]" << param->thread_ctr << "] " << "ERROR! In 'yparse'
        , rule 'statement: GET_USER_INFO_YY STRING_YY delay_option':"
        << "'get_user_info_func' failed, returning" << status << "." << endl;
      unlock_cerr_mutex();
      ++param->errors_occurred;
    } /* if (status != 0) */
#endif DEBUG_COMPILE
  else
    if (param->PARSER_DEBUG) {
      lock_cerr_mutex();
      cerr << "[Thread]" << param->thread_ctr << "] " <<
        "In 'yparse', rule 'statement: GET_USER_INFO_YY STRING_YY delay_option':"
        << "'get_user_info_func' succeeded, returning 0." << endl;
      unlock_cerr_mutex();
    } /* else if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
  } /* if ([\$3] == 0) (No delay) */
  else /* Delay */
  {
    Response_Type response;
    response.type = Response_Type::GET_USER_INFO_TYPE;
    response.string_val = [\$2];
    param->delayed_response_deque.push_back(response);
  } /* else (Delay) */
}

```

```

    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
;
```

141. <delay option> and <sub-delay option>. [LDF Undated.]

Log

[LDF 2013.08.07.] Changed the type of <delay option> from *int_value* to *ulint_value*.
 [LDF 2013.08.26.] Added type declaration for <sub-delay option>.

< Token and type declarations 128 > +≡

%type<ulint_value>delay_option	%type<ulint_value>sub_delay_option
--------------------------------	------------------------------------

142. <delay option> → Empty. [LDF 2013.05.24.]

Log

[LDF 2013.05.24.] Added this rule.

< Rules 11 > +≡

delay_option:/*Empty*/

```

{
  Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
    *>(yyget_extra(parameter));
  bool save_PARSER_DEBUG = param->PARSER_DEBUG;
  param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
  if (param->PARSER_DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << param->thread_ctr << "]In'yparse',rule"
      << "delay_option:/*Empty*/." << endl;
    unlock_cerr_mutex();
  } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
  param->delay_value = $$ = 0UL;
  param->PARSER_DEBUG = save_PARSER_DEBUG;
}
```

143. <delay option> → <sub-delay option> [LDF 2013.08.26.]

Log

[LDF 2013.08.26.] Added this rule.

```
<Rules 11> +≡
[delay_option:⊣sub_delay_option⊣]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread⊣" << param->thread_ctr << "]⊣In⊣'yparse',⊣rule⊣" <<
            "'delay_option:⊣sub_delay_option'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    $$ = $1;
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

144. ⟨sub-delay option⟩ → DELAY_YY. [LDF 2013.05.24.]

Log

[LDF 2013.05.24.] Added this rule.

```

⟨Rules 11⟩ +≡
  sub_delay_option:DELAY_YY
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] In 'yyparse', rule"
            << " 'sub_delay_option:DELAY_YY' ." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->delay_value = [ $$ ] = 1UL;
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] In 'yyparse', rule"
            << " 'sub_delay_option:DELAY_YY' :" << endl << "$$ ==" << [ $$ ] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

145. <sub-delay option> → DELAY_YY INTEGER_YY. [LDF 2013.08.07.]

Log

[LDF 2013.08.07.] Added this rule.

```

⟨Rules 11⟩ +≡
  sub_delay_option : ↳DELAY_YY ↳INTEGER_YY
  {
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
      *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
      lock_cerr_mutex();
      cerr << "[Thread" << param->thread_ctr << "] In 'yparse', rule"
        << "sub_delay_option:DELAY_YY INTEGER_YY." << endl << "'INTEGER_YY' ($2)=="
        << "$2" << endl;
      unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    Response_Type response;
    response.type = Response_Type::COMMAND_ONLY_TYPE;
    if ([§2] == 1) {
      temp_strm.str("");
      temp_strm << "DELAY_OPTION_RESPONSE_2\"WARNING! Argument 1 used.\n" <<
        "This value is used to indicate that the default delay" <<
        "amount should be used.\n" << "It is not currently possible to specify"
        "a delay of one second.\"";
      response.command = temp_strm.str();
      param->response_deque.push_back(response);
      param->delay_value = [§§] = 1UL;
    }
    else if ([§2] > 1) param->delay_value = [§§] = static_cast<unsigned long int>([§2]);
    else {
      temp_strm.str("");
      temp_strm << "DELAY_OPTION_RESPONSE_2\"WARNING! Invalid argument" << [§2] <<
        " used.\n" << "Will use default delay amount.\n";
      response.command = temp_strm.str();
      param->response_deque.push_back(response);
      param->delay_value = [§§] = 1UL;
    }
    param->PARSER_DEBUG = save_PARSER_DEBUG;
  }
;
```

146. <sub-delay option> → DELAY_YY EQUALS_YY INTEGER_YY. [LDF 2013.08.07.]

Log

[LDF 2013.08.07.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
  sub_delay_option : [DELAY_YY EQUALS_YY INTEGER_YY]
  {
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
      *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
      lock_cerr_mutex();
      cerr << "[Thread" << param->thread_ctr << "] In 'yparse', rule"
        << "'sub_delay_option:[DELAY_YY EQUALS_YY INTEGER_YY]' << endl
        << "'INTEGER_YY' ($3) == " << [$3] << endl;
      unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    Response_Type response;
    response.type = Response_Type::COMMAND_ONLY_TYPE;
    if ([\$3] == 1) {
      temp_strm.str("");
      temp_strm << "DELAY_OPTION_RESPONSE_2\"WARNING! Argument 1 used.\n"
        << "This value is used to indicate that the default delay"
        << "amount should be used.\nIt is not currently possible to specify"
        << "a delay of one second.\"";
      response.command = temp_strm.str();
      param->response_deque.push_back(response);
      param->delay_value = [\$\$] = 1UL;
    }
    else if ([\$3] > 1) param->delay_value = [\$\$] = static_cast<unsigned long int>([\$3]);
    else {
      temp_strm.str("");
      temp_strm << "DELAY_OPTION_RESPONSE_2\"WARNING! Invalid argument" << [\$3] <<
        " used.\n"
        << "Will use default delay amount." "";
      response.command = temp_strm.str();
      param->response_deque.push_back(response);
      param->delay_value = [\$\$] = 1UL;
    }
    param->PARSER_DEBUG = save_PARSER_DEBUG;
  }
;
```

147. <sub-delay option> → DELAY_YY TIME_SPECIFICATION_YY. [LDF 2013.08.07.]

Log

[LDF 2013.08.07.] Added this rule.

```

⟨Rules 11⟩ +≡
  sub_delay_option :1DELAY_YY1TIME_SPECIFICATION_YY
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] In 'yparse', rule"
            << "sub_delay_option:1DELAY_YY1TIME_SPECIFICATION_YY'." << endl <<
            "'TIME_SPECIFICATION_YY'" <=$2<=" << $2 << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    if ($2 > 0 & $2 < ULONG_MAX) param->delay_value = $$ = $2;
    else {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] WARNING! In 'yparse', rule"
            << "sub_delay_option:1DELAY_YY1TIME_SPECIFICATION_YY':" <<
            endl << "'TIME_SPECIFICATION_YY'" <=$0 or "'ULONG_MAX'." <<
            endl << "Invalid time specification." << endl <<
            "Setting value of rule ('sub_delay_option') to 1UL and will try to continue." <<
            endl;
        unlock_cerr_mutex();
        ++param->warnings_occurred;
        param->delay_value = $$ = 1UL;
    } /* else */
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

148. <sub-delay option> → DELAY_YY EQUALS_YY TIME_SPECIFICATION_YY. [LDF 2013.08.07.]

Log

[LDF 2013.08.07.] Added this rule.

```

⟨Rules 11⟩ +≡
  sub_delay_option:DELAY_YY EQUALS_YY TIME_SPECIFICATION_YY
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] In 'yparse', rule"
            << "'sub_delay_option:DELAY_YY EQUALS_YY TIME_SPECIFICATION_YY'." << endl
            << "'TIME_SPECIFICATION_YY' ($3) == " << $3 << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    if (($3) > 0 & ($3) < ULONG_MAX) param->delay_value = $$ = $3;
    else {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] WARNING! In 'yparse', rule"
            << "'sub_delay_option:DELAY_YY TIME_SPECIFICATION_YY':"
            << endl << "'TIME_SPECIFICATION_YY' == 0 or 'ULONG_MAX'." <<
            endl << "Invalid time specification." << endl <<
            "Setting value of rule ('sub_delay_option') to 1UL and will try to continue." <<
            endl;
        unlock_cerr_mutex();
        ++param->warnings_occurred;
        param->delay_value = $$ = 1UL;
    } /* else */
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

149. ⟨sub-delay option⟩ → NO_DELAY_YY. [LDF 2013.05.24.]

Log

[LDF 2013.05.24.] Added this rule.

```
{Rules 11} +≡
sub_delay_option: NO_DELAY_YY
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] In 'yparse', rule"
            << " 'sub_delay_option: NO_DELAY_YY'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->delay_value = $$ = 0UL;
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

150. ⟨statement⟩ → PWD_YY. [LDF 2012.07.04.]

Log

[LDF 2012.07.04.] Added this rule.

[LDF 2013.02.05.] Removed code for the cases *jargon_trunk* ≡ *true* and *jargon_core* ≡ *true*. Not currently using either of the Jargon APIs.

[LDF 2013.04.03.] Removed code from this rule. Now pushing a *Response_Type* object with *type* ≡ *Response_Type* :: PWD_TYPE onto *param→response_deque*. This makes it possible to have pending operations performed before processing the response created by this rule. Previously, the ipwd command was executed immediately.

```
⟨Rules 11⟩ +≡
[statement: ↳PWD_YY ↲]
{
    Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
        *>(yyget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param→PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param→thread_ctr << "] " ↳statement: ↳PWD_YY" << endl;
        unlock_cerr_mutex();
    } /* if (param→PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    Response_Type response;
    response.type = Response_Type :: PWD_TYPE;
    param→response_deque.push_back(response);
}
;
```

151. ⟨statement⟩ → LS_YY ⟨string list optional⟩ ⟨delay option⟩. [LDF 2012.07.04.]

Log

[LDF 2012.07.04.] Added this rule.

[LDF 2012.09.11.] Added code for calling *lsUtil*.

[LDF 2012.11.27.] Removed code. Now creating a *Response_Type* object of type *Response_Type* :: LS_TYPE and pushing it onto *param-response_deque*. This is necessary to ensure that commands like `mkdir`, `put`, etc., that change the directory structure are executed first.

[LDF 2013.04.25.] Added ⟨flags optional⟩ and ⟨string list optional⟩.

[LDF 2013.05.24.] Added ⟨delay option⟩.

⟨Rules 11⟩ +≡

```

[statement:LS_YY	flags_optional	string_list_optional	delay_option]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] In 'yparse':"
            << "Rule 'statement:LS_YY	flags_optional	string_list_optional	delay_option'." <<
            endl << "flags_optional==" << $2 << endl << "delay_option==" << $4 << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    Response_Type response;
    response.type = Response_Type :: LS_TYPE;
    response.string_vector = param->string_vector;
    response.flags = $2;
    if ($4 == -1) response.no_delay = 1;
    param->string_vector.clear();
    param->response_deque.push_back(response);
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

152. ⟨string list optional⟩. [LDF 2013.04.25.]

Log

[LDF 2013.04.25.] Added this type declaration.

⟨Token and type declarations 128⟩ +≡

```
%type<int_value>	string_list_optional
```

153. ⟨string list optional⟩ → Empty. [LDF 2013.04.25.]

Log

[LDF 2013.04.25.] Added this rule.

```
⟨Rules 11⟩ +≡
[string_list_optional:/*Empty*/]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] In 'yparse', rule"
            << "string_list_optional:/*Empty*/." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->string_vector.clear();
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

154. ⟨string list optional⟩ → ⟨string list optional⟩ STRING_YY. [LDF 2013.04.25.]

Log

[LDF 2013.04.25.] Added this rule.

```
⟨Rules 11⟩ +≡
[string_list_optional:string_list_optional_STRING_YY]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] In 'yparse', rule"
            << "string_list_optional:string_list_optional_STRING_YY."
            << endl <<
            "'STRING_YY'" << [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->string_vector.push_back(string([$2]));
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

155. <statement> → MKDIR_YY <flags optional> <filename list>. [LDF 2012.06.25.]

Log

[LDF 2012.06.25.] Added this rule.

[LDF 2012.11.29.] Added <flags optional>. Replaced STRING_YY with <filename list>.

<Rules 11> +≡

```

statement :_MKDIR_YY(flags_optional,filename_list)
{
    Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
        *>(yyget_extra(parameter));
#endif DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread_" << param->thread_ctr << "] In 'yparse', rule"
            << "statement:_MKDIR_YY(flags_optional,filename_list):" << endl <<
            "'flags_optional'==" << $2 << endl << "'param->filename_vector':" << endl;
        for (vector<string>::const_iterator iter = param->filename_vector.begin();
            iter != param->filename_vector.end(); ++iter) {
            cerr << *iter << endl;
        } /* for */
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    Response_Type response;
    response.type = Response_Type::MKDIR_TYPE;
    response.flags = $2;
    response.string_vector = param->filename_vector;
    param->filename_vector.clear();
    param->response_deque.push_back(response);
}
;
```

156. ⟨statement⟩ → RM_YY ⟨flags optional⟩ ⟨rm option list⟩ ⟨filename list⟩. [LDF 2012.06.26.]

Log

[LDF 2012.11.30.] Changed STRING_YY to ⟨filename list⟩. Added ⟨flags optional⟩.

```

⟨ Rules 11 ⟩ +≡
statement: RM_YY flags_optional rm_option_list filename_list {
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
    *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] In "yyparse", rule" <<
            "'statement: RM_YY flags_optional rm_option_list':"
            << endl <<
            "'flags_optional'" << "$2" << endl << "'rm_option_list'" << "$3" <<
            oct << "0" << "$3" << "(octal)" << dec << endl << "'param->delay_value'" <<
            param->delay_value << endl << "'filename_vector'" << endl;
        for (vector<string>::const_iterator iter = param->filename_vector.begin();
            iter != param->filename_vector.end(); ++iter) {
            cerr << *iter << endl;
        }
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    Response_Type response;
    Response_Type process_pending_response;
    process_pending_response.type = Response_Type::PROCESS_PENDING_TYPE;
    response.flags = $2;
    response.string_vector = param->filename_vector;

```

157. Handle ‘database’ and ‘database-only’ options. [LDF 2013.08.12.]

This conditional is needed because *response.options* uses different bits for database and database-only than ⟨rm option⟩ does. [LDF 2013.08.12.]

Log

[LDF 2013.08.12.] Added this section.

```

⟨Rules 11⟩ +≡
  if ([\$3] & 1_U)    /* Delete from database */
  {
    response.options |= 2_U;
  }
  else if ([\$3] & 2_U)  /* Delete from database only */
  {
    response.options |= 4_U;
  }
param->response_deque.push_back(process_pending_response);
response.delay_value = param->delay_value;
if (response.delay_value > 0) {
  response.no_delay = false;
  response.options |= 1_U;
}
response.type = Response_Type::MARK_IRODS_OBJECTS_FOR_DELETION_TYPE;
/* !! START HERE: LDF 2013.08.15. Look into this. See '00TODO'. */
if (param->response_deque.size() > 0) param->delayed_response_deque.push_back(response);
else param->response_deque.push_back(response);
param->filename_vector.clear();
param->delay_value = 0_UL;
param->PARSER_DEBUG = save_PARSER_DEBUG; } ;

```

158. ⟨rm option list⟩. [LDF 2013.08.09.]

Log

[LDF 2013.08.09.] Added this section.

⟨Token and type declarations 128⟩ +≡

%type <uint_value> rm_option_list	%type <uint_value> rm_option
-----------------------------------	------------------------------

159. ⟨rm option list⟩ → Empty [LDF 2013.08.09.]

Log

[LDF 2013.08.09.] Added this rule.

```

⟨Rules 11⟩ +≡
rm_option_list:/*Empty*/
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] In 'yparse', rule"
            << 'rm_option_list:/*Empty*/' << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->delay_value = 0UL;
    $$ = 0U;
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

160. ⟨rm option list⟩ → ⟨rm option list⟩ ⟨rm option⟩ [LDF 2013.08.09.]

Log

[LDF 2013.08.09.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
  rm_option_list : rm_option_list rm_option
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
    Response_Type response;
    response.type = Response_Type::COMMAND_ONLY_TYPE;
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] In 'yyparse', rule"
            << "rm_option_list:rm_option" << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    if ($2 ≡ 1U ∧ $1 ∧ 2U) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] WARNING!" << endl << "In 'yyparse': Opti\
            on 'database' used and option 'database-only'" << "already set." << endl <<
            "The last option used takes precedence: Setting 'database' option" << endl <<
            "and unsetting 'database-only'." << endl;
        unlock_cerr_mutex();
        temp_strm.str();
        temp_strm << "DATABASE OPTION RESPONSE 2 \"WARNING! Multiple database op\
            tions used.\" << endl << "The last one takes precedence." << endl <<
            "Setting 'database' and unsetting 'database-only'.\"";
        response.command = temp_strm.str();
        temp_strm.str();
        param->response_deque.push_back(response);
        ++param->warnings_occurred;
    };
    $$ &= ~2U;
    $$ |= 1U;
}
else if ($2 ≡ 2U ∧ $1 ∧ 1U) {
    lock_cerr_mutex();
    cerr << "[Thread" << param->thread_ctr << "] WARNING!" << endl << "In 'yyparse': Opti\
        on 'database-only' used and option 'database'" << "already set." << endl <<
        "The last option used takes precedence: Setting 'database-only' option" <<
        endl << "and unsetting 'database'." << endl;
    unlock_cerr_mutex();
    temp_strm.str();
    temp_strm << "DATABASE OPTION RESPONSE 2 \"WARNING! Multiple database op\
        tions used.\" << endl << "The last one takes precedence." << endl <<
        "Setting 'database-only' and unsetting 'database'.”";
}

```

```

response.command = temp_strm.str();
temp_strm.str();
++param->warnings_occurred;
;
param->response_deque.push_back(response);
[$$] &= ~1U;
[$$] |= 2U;
}
else [$$] = [$1] | [$2];
param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

161. ⟨rm option list⟩ → ⟨rm option list⟩ ⟨delay option⟩ [LDF 2013.08.09.]

Log

[LDF 2013.08.09.] Added this rule.

```

⟨Rules 11⟩ +≡
[rm_option_list: rm_option_list delay_option]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] In 'yyparse', rule"
            << " 'rm_option_list: rm_option_list delay_option'" << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    [$$] = [$1];
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

162. ⟨rm option⟩ → ⟨database option⟩. [LDF 2013.08.09.]

Log

[LDF 2013.08.09.] Added this rule.

[LDF 2013.08.15.] Replaced DATABASE_YY with ⟨database option⟩.

```
⟨Rules 11⟩ +≡
[rm_option:database_option]
{
    Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false;      /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] In 'yparse', rule"
            << " 'rm_option:database_option'" << endl;
        unlock_cerr_mutex();
    }      /* if (param->PARSER_DEBUG) */
#endif      /* DEBUG_COMPILE */
    $$ |= $1;
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

163. ⟨statement⟩ → PUT_YY ⟨flags optional⟩ ⟨pid options⟩ STRING_YY ⟨server-side filename optional⟩.
[LDF 2012.09.26.]

Log

[LDF 2012.09.26.] Added this rule.

[LDF 2012.09.27.] Removed code to **Scan_Parser_Parameter_Type**::*put* in **scprpmtp.web**.

[LDF 2012.11.22.] Rewrote this rule. Now creating a *Response_Type* object and putting it onto *param-response_map*. This object stores the information passed in the symbols of this rule so that it can be retrieved when the client's response for sending the file is processed.

[LDF 2013.08.29.] Moved STRING_YY from the second to the fourth position. It now follows the flags and PID options. This makes it more consistent with the icommand **iput**.

```
<Rules 11> +≡
statement :_PUT_YY	flags_optional pid_options STRING_YY	server_side_filename_optional {
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread_" << param->thread_ctr << "] In '_yparse', ,rule_" <<
            "'statement:_PUT_YY	flags_optional pid_options'" <<
            "STRING_YY	server_side_filename_optional'::" << endl << "'flags_optional'==" <<
            "$2" << endl << "'pid_options'==" << oct << "$3" << dec << endl <<
            "'STRING_YY'==" << "$4" << endl << "'server_side_filename_optional'==" <<
            "$5" << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    Response_Type response;
```

164. Error handling: Client-side filename is a directory. This isn't permitted at the present time. However, in the future I may make it possible to send the contents of an entire directory. !! TODO: Think about this. [LDF 2012.11.23.]

Log

[LDF 2012.11.23.] BUG FIX: Added this section with code for testing whether the client-side filename stored in the STRING_YY token is a directory.

```

⟨Rules 11⟩ +≡
string client_side_filename = [$4];
if (client_side_filename[client_side_filename.size() - 1] ≡ '/') {
    lock_cerr_mutex();
    cerr ≪ "[Thread" ≪ param→thread_ctr ≪ "] ERROR! In 'yparse', rule" ≪
        "statement: PUT_YY	flags_optional pid_options" ≪
        "STRING_YY server_side_filename_optional:" ≪ endl ≪ "Client-side_filename" ≪
        client_side_filename ≪ "' is a directory. This isn't permitted." ≪ endl ≪
        "Will try to continue." ≪ endl;
    unlock_cerr_mutex();
    response.type = Response_Type::COMMAND_ONLY_TYPE;
    temp_strm.str("");
    temp_strm ≪ "PUT RESPONSE" ≪ client_side_filename ≪ "\u001a\ Input_error" ≪
        "Client-side_filename is a directory" ≪ client_side_filename ≪ "\u001a";
    response.command = temp_strm.str();
    param→response_deque.push_back(response);
    temp_strm.str("");
    ++param→errors_occurred;
    goto END_PUT_RULE;
} /* if (client_side_filename[client_side_filename.size() - 1] ≡ '/') */

```

165.

```

⟨ Rules 11 ⟩ +≡
  response.type = Response_Type::RECEIVE_PUT_FILE_TYPE;
  response.remote_filename = [$4]; /* Client-side filename */
  if (strlen([$5]) > 0) /* Server-side filename */
    response.local_filename = [$5];
  else response.local_filename = response.remote_filename;
  response.command = "PUT";
  response.flags = [$2];
  response.pid_options = [$3];
  response.pid_str = param->pid_str;
  response.pid_prefix_str = param->pid_prefix_str;
  response.pid_suffix_str = param->pid_suffix_str;
  response.pid_institute_str = param->pid_institute_str;
#ifndef DEBUG_COMPILE
  if (param->PARSER_DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << param->thread_ctr << "] In 'yparse', rule"
      << "statement: PUT_YY_flags_optional pid_options" <<
      "STRING_YY_server_side_filename_optional:" << endl << "response.pid_str==" <<
      response.pid_str << endl << "response.pid_prefix_str==" << response.pid_prefix_str <<
      endl << "response.pid_suffix_str==" << response.pid_suffix_str << endl <<
      "response.pid_institute_str==" << response.pid_institute_str << endl;
    unlock_cerr_mutex();
  } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
  unsigned int ref_ctr;
  pthread_mutex_lock(&param->response_map_mutex);
  if (param->response_map.size() == 0) {
    ref_ctr = 0;
    param->response_map[0] = response;
  }
  else {
    map<unsigned int, Response_Type>::const_reverse_iterator iter =
      param->response_map.rbegin();
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
      lock_cerr_mutex();
      cerr << "iter->first==" << iter->first << endl;
      iter->second.show("iter->second:");
      unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    ref_ctr = iter->first + 1;
    param->response_map[ref_ctr] = response;
  } /* else */
  pthread_mutex_unlock(&param->response_map_mutex);
#ifndef DEBUG_COMPILE
  if (param->PARSER_DEBUG) {
    lock_cerr_mutex();
    cerr << "param->response_map[" << ref_ctr << "] :" << endl;
    param->response_map[ref_ctr].show();
  }

```

```
    unlock_cerr_mutex();
} /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
response.clear();
response.type = Response_Type::PROCESS_PENDING_TYPE;
param->response_deque.push_back(response);
response.clear();
response.type = Response_Type::COMMAND_ONLY_TYPE;
temp_strm.str("");
temp_strm << "SENDFILE\" " << [$4] << "\"REFERENCE\" " << ref_ctr;
response.command = temp_strm.str();
#endif DEBUG_COMPILE
if (param->PARSER_DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << param->thread_ctr << "] In 'yparse', rule"
        << "statement:PUT_YY_flags_optional_pid_options"
        << "STRING_YY_server_side_filename_optional' :" << endl
        << "temp_strm.str()' == "
        << endl << temp_strm.str() << endl;
    unlock_cerr_mutex();
} /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
temp_strm.str("");
param->response_deque.push_back(response);
END_PUT_RULE: ;
param->PARSER_DEBUG = save_PARSER_DEBUG; } ;
```

This rule assumes that the file named by STRING_YY is available on the machine where gwirdsif is running. [LDF 2012.10.02.]

Log

- [LDF 2012.10.02.] Added this rule.
- [LDF 2012.11.19.] Added `<client-side filename optional>`.
- [LDF 2013.04.04.] Removed code from this rule. Now pushing a `Response_Type` object of `type ≡ Response_Type::GET_TYPE` onto `Scan_Parse_Parameter_Type::response deque`.
- [LDF 2013.08.29.] Moved `STRING_YY` from the second to the third position. It now follows the optional flags. This makes it more consistent with the `icommand ige`.

167. ⟨flags optional⟩.

Log

[LDF 2012.09.27.] Added this type declaration.

⟨ Token and type declarations 128 ⟩ +≡
%type \llcorner <string_value> \lrcorner flags_optional

168. $\langle \text{flags optional} \rangle \rightarrow \text{Empty}.$

Log

[LDF 2012.09.27.] Added this rule.

```
<Rules 11> +≡
[flags_optional:/*Empty*/]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
    *>(yyget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
        << "In 'yparse', rule "flags_optional:/*Empty*/' ."
        << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    strcpy([$], "");
}
```

169. ⟨flags optional⟩ → ⟨flags optional⟩ FLAG_YY.

Log

[LDF 2012.09.27.] Added this rule.

[LDF 2012.11.29.] Added error-handling code for the case that *param*→*commands_flag_str* gets too long. The limit is 64 characters, including a trailing space. This is a fairly strict limit, so it may be necessary to increase it.

[LDF 2013.04.25.] Removed code referring to *param*→*commands_flag_str*.

```

⟨Rules 11⟩ +≡
  flags_optional:	flags_optional FLAG_YY
{
  Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
    *>(yyget_extra(parameter));
#ifndef DEBUG_COMPILE
  if (param→PARSER_DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << param→thread_ctr << "] " <<
      "In 'yparse', rule 'flags_optional:flags_optional FLAG_YY'." <<
      endl << "'flags_optional'(on right-hand-side)==" << "$1" << endl <<
      "'FLAG_YY'" == " " << "$2" << endl;
    unlock_cerr_mutex();
  } /* if (param→PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
  if (strlen("$1") + strlen("$2") + 1 > 63) {
    lock_cerr_mutex();
    cerr << "[Thread" << param→thread_ctr << "] " <<
      "WARNING! In 'yparse', rule 'flags_optional:flags_optional FLAG_YY':"
      endl << "Cannot append flag " << "$2" << "(plus trailing space)" <<
      endl << "to the value of 'flags_optional': " << "$1" << endl <<
      "The resulting string would be too long (> 63 characters)." << endl <<
      "Leaving value 'flags_optional' unchanged. Continuing." << endl;
    unlock_cerr_mutex();
  }
  else {
    strcpy($$, "$1");
    strcat($$, " ");
    strcat($$, "$2");
  }
#endif DEBUG_COMPILE
  if (param→PARSER_DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << param→thread_ctr << "] " <<
      "In 'yparse', rule 'flags_optional:flags_optional FLAG_YY'." <<
      endl << "'flags_optional'(value of result)==" << $$ << endl;
    unlock_cerr_mutex();
  } /* if (param→PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
}
;
```

170. ⟨server-side filename optional⟩. [LDF 2012.11.21.]

Log

[LDF 2012.11.21.] Added this type declaration.

⟨ Token and type declarations 128 ⟩ +≡

%type<string_value>server_side_filename_optional

171. ⟨server-side filename optional⟩ → Empty.

Log

[LDF 2012.11.21.] Added this rule.

⟨ Rules 11 ⟩ +≡

server_side_filename_optional:/*Empty*/

{

Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
*>(yyget_extra(parameter));

#if DEBUG_COMPILE

if (param->PARSER_DEBUG) {
 lock_cerr_mutex();
 cerr << "[Thread" << param->thread_ctr << "] " <<
 "In 'yyparse', rule 'server_side_filename_optional:/*Empty*/'." << endl;
 unlock_cerr_mutex();

} /* if (param->PARSER_DEBUG) */

#endif /* DEBUG_COMPILE */

strcpy(\$\$, "");

}

;

172. ⟨server-side filename optional⟩ → REMOTE_FILENAME_YY STRING_YY. [LDF 2012.11.21.]

!! PLEASE NOTE: "Remote" here refers to the server-side. Since this is potentially confusing, the left-hand-side of the rule makes this explicit. [LDF 2012.11.21.]

Log

[LDF 2012.11.21.] Added this rule.

```

⟨Rules 11⟩ +≡
[server_side_filename_optional:REMOTE_FILENAME_YY_STRING_YY]
{
    Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
        *>(yyget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
        << "In 'yparse', rule 'server_side_filename_optional:'"
        << "REMOTE_FILENAME_YY_STRING_YY'." << endl << "'STRING_YY' == "
        << [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    strcpy([$$, $2]);
}
;
```

173. ⟨client-side filename optional⟩. [LDF 2012.11.19.]

Log

[LDF 2012.11.19.] Added this type declaration.

```

⟨ Token and type declarations 128⟩ +≡
%type<string_value> client_side_filename_optional
```

174. ⟨client-side filename optional⟩ → Empty.

Log

[LDF 2012.11.19.] Added this rule.

```

⟨Rules 11⟩ +≡
  client_side_filename_optional:/*Empty*/
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
            << "In 'yparse', rule 'client_side_filename_optional:/*Empty*/' ."
            << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    strcpy($$, "");
}
;
```

175. ⟨client-side filename optional⟩ → LOCAL_FILENAME_YY STRING_YY. [LDF 2012.11.19.]

!! PLEASE NOTE: "Local" here refers to the client-side. Since this is potentially confusing, the left-hand-side of the rule makes this explicit. [LDF 2012.11.19.]

Log

[LDF 2012.11.19.] Added this rule.

```

⟨Rules 11⟩ +≡
  client_side_filename_optional:LOCAL_FILENAME_YYSTRING_YY
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread"
            << param->thread_ctr << "]"
            << "In 'yparse', rule 'client_side_filename_optional:'"
            << "LOCAL_FILENAME_YYSTRING_YY' ."
            << endl << "'STRING_YY'" << "$2"
            << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    strcpy($$, $2);
}
;
```

176. ⟨pid options⟩.

Log

[LDF 2012.09.28.] Added this type declaration.

⟨ Token and type declarations 128 ⟩ +≡
 [%type<uint_value>pid_options]

177. ⟨pid options⟩ → Empty.

Log

[LDF 2012.09.28.] Added this rule.

⟨ Rules 11 ⟩ +≡
 pid_options : /*Empty*/
 {
 Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
 *>(yyget_extra(parameter));
 #if DEBUG_COMPILE
 if (param->PARSER_DEBUG) {
 lock_cerr_mutex();
 cerr << "[Thread" << param->thread_ctr << "] " <<
 "In 'yparse', rule 'pid_options:/*Empty*/' ." << endl;
 unlock_cerr_mutex();
 } /* if (param->PARSER_DEBUG) */
 #endif /* DEBUG_COMPILE */
 [\$\$] = 0U;
 param->pid_str = "";
 param->pid_suffix_str = "";
 param->pid_prefix_str = "";
 param->pid_institute_str = "";
 }
 ;

178. $\langle \text{pid options} \rangle \longrightarrow \langle \text{pid options} \rangle \text{ PID_YY}.$

Log

[LDF 2012.09.28.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
[pid_options:pid_options_PID_YY]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
            << "In 'yparse', rule 'pid_options:pid_options_PID_YY'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    $$ += 1;
}
;
```

179. $\langle \text{pid options} \rangle \longrightarrow \langle \text{pid options} \rangle \text{ PID_YY STRING_YY}.$

Log

[LDF 2012.09.28.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
[pid_options:pid_options_PID_YY_STRING_YY]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
            << "In 'yparse', rule 'pid_options:pid_options_PID_YY_STRING_YY'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    $$ += 1;
    param->pid_str = $3;
}
;
```

180. <pid options> → <pid options> GENERATE_YY. [LDF 2012.09.28.]

!! PLEASE NOTE: This option doesn't seem to have any effect at present. [LDF 2012.11.22.]

Log

[LDF 2012.09.28.] Added this rule.

```
<Rules 11> +≡
[pid_options:pid_options GENERATE_YY]
{
    Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
        *>(yyget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
            << "In 'yparse', rule 'pid_options:pid_options GENERATE_YY'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    [$$] += 2;
}
;
```

181. <pid options> → <pid options> INSTITUTE_YY STRING_YY.

Log

[LDF 2012.09.28.] Added this rule.

```
<Rules 11> +≡
[pid_options:pid_options INSTITUTE_YY STRING_YY]
{
    Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
        *>(yyget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
            << "In 'yparse', rule 'pid_options:pid_options INSTITUTE_YY STRING_YY'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    [$$] += 4;
    param->pid_institute_str = [$3];
}
;
```

182. $\langle \text{pid options} \rangle \rightarrow \langle \text{pid options} \rangle \text{ SUFFIX_YY STRING_YY}.$

Log

[LDF 2012.09.28.] Added this rule.

```

⟨Rules 11⟩ +≡
[pid_options:pid_options]SUFFIX_YY]STRING_YY
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
    *>(yyget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
        "In 'yparse', rule 'pid_options:pid_options'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    $$ += 8;
    param->pid_suffix_str = [\$3];
}
;
```

183. $\langle \text{pid options} \rangle \rightarrow \langle \text{pid options} \rangle \text{ PREFIX_YY STRING_YY}.$

Log

[LDF 2012.09.28.] Added this rule.

```

⟨Rules 11⟩ +≡
[pid_options:pid_options]PREFIX_YY]STRING_YY
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
    *>(yyget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
        "In 'yparse', rule 'pid_options:pid_options'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    $$ += 16;
    param->pid_prefix_str = [\$3];
}
;
```

184. ⟨statement⟩ → GET_YY METADATA_YY STRING_YY. ⟨get metadata options⟩. [LDF 2012.10.09.]

Log

[LDF 2012.10.09.] Added this rule.
 [LDF 2012.12.31.] Added ⟨get metadata options⟩.

```

⟨Rules 11⟩ +≡
  statement: GET_YY METADATA_YY STRING_YY get_metadata_options
{
  Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
    *>(yyget_extra(parameter));
#ifndef DEBUG_COMPILE
  if (param->PARSER_DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread_" << param->thread_ctr << "] In " << 'yparse' << ", rule"
      << "statement: GET_YY METADATA_YY STRING_YY get_metadata_options'" << endl <<
      "'STRING_YY'" <= <= " " << $3 << endl << "'get_metadata_options'" <= <= " " << $4 << endl;
    unlock_cerr_mutex();
  } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
/* !! TODO: LDF 2012.12.13. Add flags and/or options for choosing what kind of metadata, iRODS
   user metadata (AVUs) and/or XML from the “Dublin Core” tables in the gwirdsif database. */
Response_Type response;
response.type = Response_Type::PROCESS_PENDING_TYPE;
param->response_deque.push_back(response);
response.type = Response_Type::GET_METADATA_TYPE;
response.int_val = 0;
response.local_filename = $3;
param->response_deque.push_back(response);
}
;
```

185. <statement> → GET_YY PID_YY METADATA_YY STRING_YY. <get metadata options>. [LDF 2013.03.21.]

Log

[LDF 2013.03.21.] Added this rule.

<Rules 11> +≡

```

statement: GET_YY METADATA_YY PID_YY STRING_YY get_metadata_options
{
    /* !! TODO: LDF 2012.12.13. Add flags and/or options for choosing what kind of metadata, iRODS
       user metadata (AVUs) and/or XML from the "Dublin Core" tables in the gwirdsif database. This
       doesn't work yet! LDF 2013.05.23. Currently, Scan_Parse_Parameter_Type::get_metadata
       only takes a path argument (for the iRODS object). */
    Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
        *>(yyget_extra(parameter));
    lock_cerr_mutex();
    cerr << "[Thread" << param->thread_ctr << "] WARNING! In 'yparse', rule"
        << "'statement: GET_YY METADATA_YY PID_YY STRING_YY get_metadata_options':"
        << endl << "'STRING_YY'" <= < $4 << endl << "'get_metadata_options'" <=
        < $5 << endl << "PLEASE_NOTE: This rule is currently non-functional, since"
        << "'Scan_Parse_Parameter_Type::get_metadata' has not yet been"
        << endl << "adapted to take an PID argument." << endl <<
        "Sending a message to client and continuing." << endl;
    unlock_cerr_mutex();
    ++param->warnings_occurred;
    Response_Type response;
#ifndef 0 /* This will be needed when I make this rule functional. [LDF 2013.05.24.] */
    response.type = Response_Type::PROCESS_PENDING_TYPE;
    param->response_deque.push_back(response);
#endif
    response.type = Response_Type::COMMAND_ONLY_TYPE;
    temp_strm.str("");
    temp_strm << "GET_METADATA_RESPONSE" << < $4 << "\u1\u0" << < $5 << "U"
        << "\u002GET_METADATA_PID\u003command not yet implemented";
    response.command = temp_strm.str();
    temp_strm.str("");
    param->response_deque.push_back(response);
}
;
```

186. <get metadata options>.

Log

[LDF 2012.12.31.] Added this type declaration.

< Token and type declarations 128> +≡

```
%type <uint_value> get_metadata_options
```

187. $\langle \text{get metadata options} \rangle \rightarrow \text{Empty}$.

Log

[LDF 2012.12.31.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
  get_metadata_options: /* Empty */
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
            << "In 'yparse', rule 'get_metadata_options:/* Empty */' ."
            << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    $$ = 0U;
}
;
```

188. $\langle \text{get metadata options} \rangle \rightarrow \langle \text{get metadata options} \rangle \text{ OUTPUT_YY}$.

Log

[LDF 2012.12.31.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
  get_metadata_options: get_metadata_options_OUTPUT_YY
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
            << "In 'yparse', rule 'get_metadata_options: get_metadata_options_OUTPUT_YY' ."
            << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    $$ += 1;
}
;
```

189. ⟨statement⟩ → GET_YY HANDLE_YY PID_YY STRING_YY. [LDF 2012.10.15.]

Log

[LDF 2012.10.15.] Added this rule.

```

⟨Rules 11⟩ +≡
statement: GET_YY HANDLE_YY PID_YY STRING_YY
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
    *>(yyget_extra(parameter));
#endif DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] In "yyparse", rule"
        << "statement: GET_YY HANDLE_YY PID_YY STRING_YY :" << endl << "'STRING_YY' == "
        [&4] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    Response_Type response;
    response.type = Response_Type::PROCESS_PENDING_TYPE;
    param->response_deque.push_back(response);
    response.type = Response_Type::GET_HANDLE_TYPE;
    response.int_val = 0;
    response.pid_str = [&4];
    param->response_deque.push_back(response);
}
;

```

190. $\langle \text{statement} \rangle \rightarrow \text{GET_YY HANDLE_YY FILE_YY STRING_YY}$. [LDF 2012.10.15.]

Log

[LDF 2012.10.15.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
statement: GET_YY HANDLE_YY FILE_YY STRING_YY
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread_" << param->thread_ctr << "] In "yyparse", rule"
            << "statement: GET_YY HANDLE_YY FILE_YY STRING_YY :" << endl << "'STRING_YY'" <=
            [$4] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    Response_Type response;
    response.type = Response_Type::PROCESS_PENDING_TYPE;
    param->response_deque.push_back(response);
    response.type = Response_Type::GET_HANDLE_TYPE;
    response.int_val = 1;
    response.local_filename = [$4];
    param->response_deque.push_back(response);
}
;

```

191. <statement> → SEND_YY TAN_YY LIST_YY. [LDF 2012.07.19.]

!! TODO: Implement this rule, if needed. There are functions for it, but they currently don't work. I don't know whether TANs will be needed. [LDF 2013.05.23.]

Log

[LDF 2012.07.19.] Added this rule.

```

⟨Rules 11⟩ +≡
[statement: SEND_YY TAN_YY LIST_YY]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "In 'yparse', rule 'statement: SEND_YY TAN_YY LIST_YY'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    lock_cerr_mutex();
    cerr << "[Thread" << param->thread_ctr << "] " <<
        "WARNING! In 'yparse', rule 'statement: SEND_YY TAN_YY LIST_YY':"
        << endl << "This command is currently non-functional. It may not be needed."
        << endl << "Sending a message to the client and continuing." << endl;
    unlock_cerr_mutex();
    ++param->warnings_occurred;
    Response_Type response;
    response.type = Response_Type::COMMAND_ONLY_TYPE;
    temp_strm.str("");
    temp_strm << "SEND TAN LIST RESPONSE\3\The\002SEND TAN LIST\003command"
        << " is currently non-functional.\\"";
    response.command = temp_strm.str();
    temp_strm.str("");
    param->response_deque.push_back(response);
#endif 0
    response.type = Response_Type::SEND_TAN_LIST_TYPE;
#endif
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

192. ⟨statement⟩ → SENDING_YY FILE_YY ⟨filename list⟩. [LDF 2012.09.27.]

Log

[LDF 2012.09.27.] Added this rule.

```

⟨Rules 11⟩ +≡
statement: SENDING_YY FILE_YY filename_list
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
#endif DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread]" << param->thread_ctr << "] "
            << "In 'yparse', rule 'statement: SENDING_YY FILE_YY filename_list'." << endl;
        for (vector<string>::const_iterator iter = param->filename_vector.begin();
            iter != param->filename_vector.end(); ++iter) {
            cerr << "*iter==" << *iter << endl;
        }
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    status = param->receive_file();
    if (status != 0) {
        lock_cerr_mutex();
        cerr << "[Thread]" << param->thread_ctr << "] "
            << "ERROR! In 'yparse', rule 'statement: SENDING_YY FILE_YY filename_list':"
            << endl << "'Scan_Parser_Parameter_Type::receive_file' failed, returning"
            << status << "."
            << endl << "Will try to continue."
            << endl;
        unlock_cerr_mutex();
    } /* if (status != 0) */
#endif DEBUG_COMPILE
    else
        if (param->PARSER_DEBUG) {
            lock_cerr_mutex();
            cerr << "[Thread]" << param->thread_ctr << "] "
                << "In 'yparse', rule 'statement: SENDING_YY FILE_YY filename_list':"
                << endl << "'Scan_Parser_Parameter_Type::receive_file' succeeded, returning 0."
                << endl;
            unlock_cerr_mutex();
        } /* else if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->filename_vector.clear();
}
;
```

193. ⟨filename list⟩.

Log

[LDF 2012.09.27.] Added this type declaration.

⟨ Token and type declarations 128 ⟩ +≡
 [%type<int_value>filename_list]

194. ⟨filename list⟩ → STRING_YY.

Log

[LDF 2012.09.27.] Added this rule.

⟨ Rules 11 ⟩ +≡
 [filename_list:STRING_YY]
 {
 Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
 *>(yyget_extra(parameter));
 #if DEBUG_COMPILE
 if (param->PARSER_DEBUG) {
 lock_cerr_mutex();
 cerr << "[Thread" << param->thread_ctr << "] " <<
 "In 'yyparse', rule 'filename_list:STRING_YY.' " << endl;
 unlock_cerr_mutex();
 } /* if (param->PARSER_DEBUG) */
 #endif /* DEBUG_COMPILE */
 \$\$ = 0;
 param->filename_vector.clear();
 param->filename_vector.push_back(\$1);
 }
 ;

195. $\langle \text{filename list} \rangle \rightarrow \langle \text{filename list} \rangle \text{ STRING_YY}.$

Log

[LDF 2012.09.27.] Added this rule.

```

⟨Rules 11⟩ +≡
filename_list:filename_list STRING_YY
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "In 'yparse', rule 'filename_list:filename_list STRING_YY'." <<
            endl << "'STRING_YY'" << [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->filename_vector.push_back([$2]);
}
;
```

196. ⟨statement⟩ → CLIENT_YY SENDING_YY FILE_YY STRING_YY REFERENCE_YY INTEGER_YY. [LDF 2012.11.22.] ■

Log

[LDF 2012.11.22.] Added this rule.

[LDF 2012.12.14.] Added code for testing the type of the *response* retrieved from *param→response_map*. Previously, this rule was only used for files sent to be used in a “put” command. I have now adapted it for use with an “add metadata” command as well.

```

⟨Rules 11⟩ +≡
  statement:CLIENT_YY_SENDING_YY_FILE_YY_STRING_YY_REFERENCE_YY_INTEGER_YY
{
  Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
    *>(yyget_extra(parameter));
#ifndef DEBUG_COMPILE
  if (param->PARSER_DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread]" << param->thread_ctr << "] " <<
      "In 'yparse', rule 'statement:CLIENT_YY_SENDING_YY_FILE_YY' <<
      "STRING_YY_REFERENCE_YY_INTEGER_YY':" << endl << "'STRING_YY' (filename) == " <<
      [$4] << endl << "'INTEGER_YY' (reference) == " << [$6] << endl;
    unlock_cerr_mutex();
  } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
  int status = client_sending_file_rule_func(param, [$4], [$6]);
  if (status != 0) {
    lock_cerr_mutex();
    cerr << "[Thread]" << param->thread_ctr << "] " << "ERROR! In 'yparse', rule 'statement:\
      CLIENT_YY_SENDING_YY_FILE_YY' << "STRING_YY_REFERENCE_YY_INTEGER_YY':" << endl <<
      "'client_sending_file_rule_func' failed, returning " << status << "." << endl <<
      "'Will try to continue.'" << endl;
    unlock_cerr_mutex();
  } /* if (status != 0) */
#ifndef DEBUG_COMPILE
  else
    if (param->PARSER_DEBUG) {
      lock_cerr_mutex();
      cerr << "[Thread]" << param->thread_ctr << "] " <<
        "In 'yparse', rule 'statement:CLIENT_YY_SENDING_YY_FILE_YY' <<
        "STRING_YY_REFERENCE_YY_INTEGER_YY':" << endl <<
        "'client_sending_file_rule_func' succeeded, returning 0." << endl;
      unlock_cerr_mutex();
    } /* else if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
}
;
```

197. ⟨statement⟩ → CD_YY. [LDF 2012.11.27.]

Log

[LDF 2012.11.27.] Added this rule.

```
⟨Rules 11⟩ +≡
[statement:CD_YY]
{
    Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
        *>(yyget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
            << "In 'yparse', rule 'statement:CD_YY'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    Response_Type response;
    response.type = Response_Type::CD_TYPE;
    response.dirname = "";
    param->response_deque.push_back(response);
}
;
```

198. ⟨statement⟩ → CD_YY STRING_YY. [LDF 2012.07.30.]

Log

[LDF 2012.07.30.] Added this rule.

```
⟨Rules 11⟩ +≡
[statement:CD_YY STRING_YY]
{
    Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
        *>(yyget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
            << "In 'yparse', rule 'statement:CD_YY STRING_YY'." << endl
            << "'STRING_YY'" << endl
            << "$2" << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    Response_Type response;
    response.type = Response_Type::CD_TYPE;
    response.dirname = "$2";
    param->response_deque.push_back(response);
}
;
```

199. ⟨statement⟩ → ADD_YY METADATA_YY STRING_YY STRING_YY ⟨add metadata options⟩. [LDF 2012.12.14.] ■

Log

[LDF 2012.12.14.] Added this rule.
 [LDF 2012.12.31.] Added ⟨add metadata options⟩.

```

⟨Rules 11⟩ +≡
statement : ADD_YY METADATA_YY STRING_YY STRING_YY add_metadata_options
{
  Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
    *>(yyget_extra(parameter));
  bool save_PARSER_DEBUG = param->PARSER_DEBUG;
  param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
  if (param->PARSER_DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << param->thread_ctr << "] " <<
      "In 'yyparse', rule 'statement: ADD_YY METADATA_YY STRING_YY STRING_YY'" <<
      "add_metadata_options:'" << endl << "'STRING_YY' 1==" << $3 << endl <<
      "'STRING_YY' 2==" << $4 << endl << "'add_metadata_options'" <= << $5 << endl;
    unlock_cerr_mutex();
  } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
  Response_Type response;
  response.type = Response_Type::RECEIVE_METADATA_FILE_TYPE;
  response.metadata_options = $5;
  response.remote_filename = $3; /* Metadata filename */
  response.local_filename = $4; /* iRODS object */
  response.command = "ADDMETADATA";
#ifndef DEBUG_COMPILE
  if (param->PARSER_DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << param->thread_ctr << "] In 'yyparse', rule" <<
      "'statement: ADD_YY METADATA_YY STRING_YY STRING_YY'." << endl;
    response.show("response:");
    unlock_cerr_mutex();
  } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
  unsigned int ref_ctr;
  pthread_mutex_lock(&param->response_map_mutex);
  if (param->response_map.size() == 0) {
    ref_ctr = 0;
    param->response_map[0] = response;
  }
  else {
    map<unsigned int, Response_Type>::const_reverse_iterator iter =
      param->response_map.rbegin();
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
      lock_cerr_mutex();
      cerr << "iter->first==" << iter->first << endl;
      iter->second.show("iter->second:");
    }
  }
}

```

```

        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
ref_ctr = iter->first + 1;
param->response_map[ref_ctr] = response;
} /* else */
pthread_mutex_unlock(&param->response_map_mutex);
#ifndef 0 /* 1 */
cerr << "param->response_map[" << ref_ctr << "] :" << endl;
param->response_map[ref_ctr].show();
#endif
response.clear();
response.type = Response_Type::PROCESS_PENDING_TYPE;
param->delayed_response_deque.push_back(response);
response.type = Response_Type::COMMAND_ONLY_TYPE;
temp_strm.str("");
temp_strm << "SENDFILE\" " << $3 << "\"REFERENCE\" " << ref_ctr;
response.command = temp_strm.str();
#endif DEBUG_COMPILE
if (param->PARSER_DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << param->thread_ctr << "]In'yparse',rule"
        << "statement:ADD_YY_METADATA_YY_STRING_YY_STRING_YY:'"
        << endl <<
        "'temp_strm.str()'=="
        << endl << temp_strm.str() << endl;
    unlock_cerr_mutex();
} /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
temp_strm.str("");
param->delayed_response_deque.push_back(response);
param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

200. ⟨add metadata options⟩.

Log

[LDF 2012.12.31.] Added this type declaration.

⟨ Token and type declarations 128 ⟩ +≡
 %type<uint_value> add_metadata_options

201. $\langle \text{add metadata options} \rangle \rightarrow \text{Empty}$.

Log

[LDF 2012.12.31.] Added this rule.

```

⟨Rules 11⟩ +≡
[add_metadata_options:/*Empty*/]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
        << "In 'yparse', rule 'add_metadata_options:/*Empty*/'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    [$$] = 0U;
}
;
```

202. $\langle \text{add metadata options} \rangle \rightarrow \langle \text{add metadata options} \rangle \text{ FORCE_YY}$.

FORCE_YY is equivalent to FORCE_ALL_YY, i.e., “force” applies to adding and storing the metadata file.
[LDF 2013.03.06.]

Log

[LDF 2012.12.31.] Added this rule.

```

⟨Rules 11⟩ +≡
[add_metadata_options:[add_metadata_options,FORCE_YY]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
        << "In 'yparse', rule 'add_metadata_options:[add_metadata_options,FORCE_YY']."
        << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    [$$] += 3U;
}
;
```

203. <add metadata options> → <add metadata options> FORCE_ADD_YY.

Log

[LDF 2013.03.06.] Added this rule.

```

⟨Rules 11⟩ +≡
[add_metadata_options: add_metadata_options FORCE_ADD_YY]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "In 'yparse', rule 'add_metadata_options: add_metadata_options FORCE_ADD_YY'." <<
            endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    [ $$ ] += 1U;
}
;
```

204. <add metadata options> → <add metadata options> FORCE_STORE_YY.

Log

[LDF 2013.03.06.] Added this rule.

```

⟨Rules 11⟩ +≡
[add_metadata_options: add_metadata_options FORCE_STORE_YY]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " << "In 'yparse', rule 'add_metadata_option\
            s: add_metadata_options FORCE_STORE_YY'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    [ $$ ] += 6U; /* --force-store implies --store */
}
;
```

205. <add metadata options> → <add metadata options> FORCE_ALL_YY.

Log

[LDF 2013.03.06.] Added this rule.

```

⟨Rules 11⟩ +≡
[add_metadata_options: add_metadata_options FORCE_ALL_YY]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
#endif DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "In 'yparse', rule 'add_metadata_options: add_metadata_options'." <<
            endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    [ $$ ] += 3U;
}
;
```

206. <add metadata options> → <add metadata options> STORE_YY.

Log

[LDF 2013.02.28.] Added this rule.

```

⟨Rules 11⟩ +≡
[add_metadata_options: add_metadata_options STORE_YY]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
#endif DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "In 'yparse', rule 'add_metadata_options: add_metadata_options'." <<
            endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    [ $$ ] += 4U;
}
;
```

207. ⟨statement⟩ → CLIENT_YY FINISHED_YY. [LDF 2012.07.30.]

Log

[LDF 2012.07.30.] Added this rule.

```

⟨Rules 11⟩ +≡
statement: CLIENT_YY FINISHED_YY
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
        << "In 'yparse', rule 'statement:CLIENT_YY FINISHED_YY'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->client_finished = true;
}
;
```

208. ⟨statement⟩ → SLEEP_YY INTEGER_YY. [LDF 2013.04.19.]

Log

[LDF 2013.04.19.] Added this rule.

```

⟨Rules 11⟩ +≡
statement: SLEEP_YY INTEGER_YY
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread]" << param->thread_ctr << "] In 'yparse':Rule:" <<
            "statement:SLEEP_YY INTEGER_YY'." << endl << "'INTEGER_YY'==" << [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    Response_Type response;
    if (sleep_server_enabled) {
#ifndef DEBUG_COMPILE
        if (param->PARSER_DEBUG) {
            lock_cerr_mutex();
            cerr << "[Thread]" << param->thread_ctr << "] In 'yparse':Rule:" <<
                "statement:SLEEP_YY INTEGER_YY':" << endl <<
                "'sleep_server_enabled'=="'true'.Setting\up\"sleep\"response." << endl;
            unlock_cerr_mutex();
        } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
        response.type = Response_Type::SLEEP_TYPE;
        response.int_val = [$2];
        temp_strm.str("");
        temp_strm << "SLEEP_RESPONSE_0" << [$2];
        response.command = temp_strm.str();
        temp_strm.str("");
    } /* if (sleep_server_enabled) */
    else {
        lock_cerr_mutex();
        cerr << "[Thread]" << param->thread_ctr << "] WARNING! In 'yparse':Rule:" <<
            "'statement:SLEEP_YY INTEGER_YY':" << endl << "'sleep_server_enabl\
            ed'=="'false'.\" << "Not putting server program 'gwirdsif' to sleep." << endl <<
            "Will send failure notice to client." << endl;
        unlock_cerr_mutex();
        ++param->warnings_occurred;
        response.type = Response_Type::COMMAND_ONLY_TYPE;
        temp_strm.str("");
        temp_strm << "SLEEP_RESPONSE_1" << [$2];
        response.command = temp_strm.str();
        temp_strm.str("");
    } /* else */
}

```

```
param->response_deque.push_back(response);
param->PARSER_DEBUG = save_PARSER_DEBUG;
}
```

209. ⟨statement⟩ → SLEEP_YY CLIENT_YY INTEGER_YY. [LDF 2013.05.02.]

Log

[LDF 2013.05.02.] Added this rule.

```

⟨Rules 11⟩ +≡
statement: SLEEP_YY CLIENT_YY INTEGER_YY
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread]" << param->thread_ctr << "] In 'yparse':Rule:" <<
            "statement:SLEEP_YY.CLIENT_YY.INTEGER_YY." << endl << "INTEGER_YY" <=
            [$3] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    Response_Type response;
    response.type = Response_Type::COMMAND_ONLY_TYPE;
    if (sleep_client_enabled) {
#ifndef DEBUG_COMPILE
        if (param->PARSER_DEBUG) {
            lock_cerr_mutex();
            cerr << "[Thread]" << param->thread_ctr << "] In 'yparse':Rule:" <<
                "statement:SLEEP_YY.CLIENT_YY.INTEGER_YY:" << endl <<
                "'sleep_client_enabled'==true'.Setting up \"sleep\" response." << endl;
            unlock_cerr_mutex();
        } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
        temp_strm.str("");
        temp_strm << "SLEEP" << [$3];
        response.command = temp_strm.str();
        temp_strm.str("");
    } /* if (sleep_client_enabled) */
    else {
        lock_cerr_mutex();
        cerr << "[Thread]" << param->thread_ctr << "] WARNING! In 'yparse':Rule:" <<
            "statement:SLEEP_YY.CLIENT_YY.INTEGER_YY:" << endl <<
            "'sleep_client_enabled'==false'. Not sending 'SLEEP'" <<
            [$3] << "' command to client program 'gwirdcli'." << endl <<
            "Will send failure notice to client." << endl;
        unlock_cerr_mutex();
        ++param->warnings_occurred;
    }
    response.type = Response_Type::COMMAND_ONLY_TYPE;
    temp_strm.str("");
    temp_strm << "SLEEP_RESPONSE" << [$3];
    response.command = temp_strm.str();
    temp_strm.str("");
}

```

```
    } /* else */
param->response_deque.push_back(response);
param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

210. <statement> → SIGNAL_YY SERVER_YY INTEGER_YY. [LDF 2013.05.02.]

Log

[LDF 2013.05.02.] Added this rule.

```

<Rules 11> +≡
statement: SIGNAL_YY SERVER_YY INTEGER_YY
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] In 'yparse', rule"
            << "statement:SIGNAL_YY SERVER_YY INTEGER_YY:" << endl << "INTEGER_YY" <=
            [$3] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    map<int, string>::iterator iter = signal_name_map.find([$3]);
    if (iter == signal_name_map.end()) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] ERROR! In 'yparse', rule"
            << "statement:SIGNAL_YY SERVER_YY INTEGER_YY:" << endl << "Signal number"
            << [$3] << "not found in 'signal_name_map'. " << "Server not raising signal"
            << [$3] << ". " << endl << "Continuing." << endl;
        unlock_cerr_mutex();
        ++param->errors_occurred;
    } /* if */
    else if (signal_server_enabled) {
#ifndef DEBUG_COMPILE
        if (param->PARSER_DEBUG) {
            lock_cerr_mutex();
            cerr << "[Thread" << param->thread_ctr << "] In 'yparse', rule"
                << "statement:SIGNAL_YY SERVER_YY INTEGER_YY:" << endl << "Sending signal"
                << iter->second << "(" << iter->first << ")" << "to main thread." << endl;
            unlock_cerr_mutex();
        } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
        lock_cout_mutex();
        lock_cerr_mutex();
        cout << "Sending signal" << iter->second << "(" << iter->first << ")" << "to main thread."
            << endl;
        unlock_cerr_mutex();
        unlock_cout_mutex();
        pthread_kill(thread_id_map[0], iter->first);
    } /* else if (signal_server_enabled) */
    else /* !signal_server_enabled */
    {

```

```
lock_cerr_mutex();
cerr << "[Thread" << param->thread_ctr << "] " WARNING! In 'yparse', rule "
"statement:SIGNAL_YY_SERVER_YY_INTEGER_YY:" << endl <<
"signal_server_enabled'" == 'false'." << "Server not raising signal"
iter->second << "(" << iter->first << ")" . " << endl << "Continuing." << endl;
unlock_cerr_mutex();
++param->warnings_occurred;
} /* else (!signal_server_enabled) */
param->PARSER_DEBUG = save_PARSER_DEBUG;
}
```

211. $\langle \text{statement} \rangle \rightarrow \text{SIGNAL_YY SERVER_YY STRING_YY}$. [LDF 2013.05.02.]

Log

[LDF 2013.05.02.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
statement: SIGNAL_YY SERVER_YY STRING_YY
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread]" << param->thread_ctr << "] In 'yparse', rule"
            << "statement:SIGNAL_YY SERVER_YY STRING_YY." << endl << "'STRING_YY'" <=
            [$3] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    multimap<string, int>::iterator iter = signal_number_map.find(string([$3]));
    if (iter == signal_number_map.end()) {
        lock_cerr_mutex();
        cerr << "[Thread]" << param->thread_ctr << "] ERROR! In 'yparse', rule"
            << "'statement:SIGNAL_YY SERVER_YY STRING_YY':"
            << endl << "Signal" << [$3] <<
            "not found in 'signal_number_map'." << "Server not raising signal."
            << endl << "Continuing." << endl;
        unlock_cerr_mutex();
        ++param->errors_occurred;
    } /* if */
    else if (signal_server_enabled) {
#ifndef DEBUG_COMPILE
        if (param->PARSER_DEBUG) {
            lock_cerr_mutex();
            cerr << "[Thread]" << param->thread_ctr << "] "
                << "'statement:SIGNAL_YY SERVER_YY STRING_YY':"
                << endl << "Signal number" <=
                iter->second << endl << "Sending signal" << signal_name_map[iter->second] << "("
                << iter->second << ") to main thread." << endl;
            unlock_cerr_mutex();
        } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
        lock_cout_mutex();
        lock_cerr_mutex();
        cout << "Sending signal" << signal_name_map[iter->second] << "(" << iter->second <<
            ")" << " to main thread." << endl;
        unlock_cerr_mutex();
        unlock_cout_mutex();
        pthread_kill(thread_ctrl_id_map[0], iter->second);
    } /* else if (signal_server_enabled) */
    else /* !signal_server_enabled */

```

```
{  
    lock_cerr_mutex();  
    cerr << "[Thread" << param->thread_ctr << "] WARNING! In 'yyparse', rule" <<  
    "'statement:SIGNAL_YY_SERVER_YY_STRING_YY':" << endl <<  
    "'signal_server_enabled'==false'.uu" << "Server not raising signal" <<  
    signal_name_map[iter-second] << "u" << "(" << iter-second << ") ." << endl <<  
    "Continuing." << endl;  
    unlock_cerr_mutex();  
    ++param->warnings_occurred;  
} /* else (~signal_server_enabled) */  
param->PARSER_DEBUG = save_PARSER_DEBUG;  
}  
;
```

212. <statement> → SIGNAL_YY CLIENT_YY INTEGER_YY. [LDF 2013.05.02.]

Log

[LDF 2013.05.02.] Added this rule.

```

⟨Rules 11⟩ +≡
statement: SIGNAL_YY CLIENT_YY INTEGER_YY
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] In 'yparse', rule"
            << "statement:SIGNAL_YY.CLIENT_YY.INTEGER_YY." << endl << "'INTEGER_YY' == "
            << "$3" << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    map<int, string>::iterator iter = signal_name_map.find($3);
    if (iter == signal_name_map.end()) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] ERROR! In 'yparse', rule"
            << "'statement:SIGNAL_YY.CLIENT_YY.INTEGER_YY':" << endl <<
            "Signal_number" << "$3" << " not found in 'signal_name_map'."
            << "Server not sending 'SIGNAL' command to client." << endl << "Continuing." << endl;
        unlock_cerr_mutex();
        ++param->errors_occurred;
    } /* if */
    else if (signal_client_enabled) {
#ifndef DEBUG_COMPILE
        if (param->PARSER_DEBUG) {
            lock_cerr_mutex();
            cerr << "[Thread" << param->thread_ctr << "] In 'yparse', rule"
                << "'statement:SIGNAL_YY.CLIENT_YY.INTEGER_YY':" << endl << "Sending 'SIGNAL' "
                << "$3" << " command to client." << endl;
            unlock_cerr_mutex();
        } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
        Response_Type response;
        response.type = Response_Type::COMMAND_ONLY_TYPE;
        temp_strm.str("");
        temp_strm << "SIGNAL" << "$3";
        response.command = temp_strm.str();
        temp_strm.str("");
        param->response_deque.push_back(response);
    } /* else if (signal_client_enabled) */
    else /* !signal_client_enabled */
    {

```

```
lock_cerr_mutex();
cerr << "[Thread" << param->thread_ctr << "] " WARNING! In 'yparse', rule "
"statement:SIGNAL_YY,CLIENT_YY,INTEGER_YY:" << endl <<
"signal_client_enabled'" == 'false' . Not_sending << "SIGNAL"
[$3] << "' command_to client" << endl << "(signal" << iter-second << "==" <<
iter-first << ")." << endl;
unlock_cerr_mutex();
++param->warnings_occurred;
} /* else (~signal_client_enabled) */
param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

213. ⟨statement⟩ → SIGNAL_YY CLIENT_YY STRING_YY. [LDF 2013.05.02.]

Log

[LDF 2013.05.02.] Added this rule.

```

⟨Rules 11⟩ +≡
statement: SIGNAL_YY CLIENT_YY STRING_YY
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] In 'yparse', rule"
            << "statement:SIGNAL_YY_CLIENT_YY_STRING_YY:" << endl << "'STRING_YY'" <=
            [$3] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    multimap<string, int>::iterator iter = signal_number_map.find(string([$3]));
    if (iter == signal_number_map.end()) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] ERROR! In 'yparse', rule"
            << "statement:SIGNAL_YY_CLIENT_YY_STRING_YY:" << endl <<
            "Signal" << [$3] << " not found in 'signal_number_map'." << endl <<
            "Not sending 'SIGNAL' command to client." << endl << "Continuing." << endl;
        unlock_cerr_mutex();
        ++param->errors_occurred;
    } /* if */
    else if (signal_client_enabled) {
#ifndef DEBUG_COMPILE
        if (param->PARSER_DEBUG) {
            lock_cerr_mutex();
            cerr << "[Thread" << param->thread_ctr << "] In 'yparse', rule"
                << "statement:SIGNAL_YY_CLIENT_YY_STRING_YY:" << endl <<
                "iter-first" << endl << "Signal number" <=
                iter->second << endl << "iter-second" << endl << "Sending 'SIGNAL' "
                << iter->second << "' command to client." << endl;
            unlock_cerr_mutex();
        } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
        Response_Type response;
        response.type = Response_Type::COMMAND_ONLY_TYPE;
        temp_strm.str("");
        temp_strm << "SIGNAL" << iter->second;
        response.command = temp_strm.str();
        temp_strm.str("");
        param->response_deque.push_back(response);
    } /* else if (signal_client_enabled) */
    else /* !signal_client_enabled */

```

```

{
    lock_cerr_mutex();
    cerr << "[Thread" << param->thread_ctr << "] " WARNING! In 'yparse', rule "
        "statement: SIGNAL_CLIENT_STRING' :" << endl <<
        "'signal_client_enabled' == 'false'. Not sending" << "SIGNAL"
        iter->second << ' command to client" << endl << "(signal" << iter->first << " == "
        iter->second << ")" . " << endl;
    unlock_cerr_mutex();
    ++param->warnings_occurred;
} /* else (~signal_client_enabled) */
param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

214. ⟨statement⟩ → SHOW_YY CERTIFICATE_YY

and

⟨statement⟩ → SHOW_YY CERTIFICATE_YY USER_YY. [LDF 2013.05.03.]

Log

[LDF 2013.05.03.] Added these rules. They share an action. ?? I wasn't able to get this to work using a vertical-bar character (|), as described in the GNU Bison manual. The file `parser.y++` generated by `ctangle` looks correct, however. Both rules were reduced correctly, but the action was only performed for the second rule.

Instead, I've used a named CWEB section and include it under each rule.

⟨ Rules 11 ⟩ +≡

statement: SHOW_YY_CERTIFICATE_YY	⟨ show certificate user action 215 ⟩
statement: SHOW_YY_CERTIFICATE_YY_USER_YY	⟨ show certificate user action 215 ⟩

215.

```

⟨ show certificate user action 215 ⟩ ≡
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false;      /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "statement:[SHOW_YY]CERTIFICATE_YY[USER_YY]." << endl;
        unlock_cerr_mutex();
    }      /* if (param->PARSER_DEBUG) */
#endif      /* DEBUG_COMPILE */
    Response_Type response;
    response.type = Response_Type::SHOW_CERTIFICATE_TYPE;
    response.int_val = 1;      /* Show single certificate, i.e., the user's. [LDF 2013.05.15.] */
    param->response_deque.push_back(response);
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

This code is used in section 214.

216. ⟨statement⟩ → SHOW_YY CERTIFICATES_YY

and

⟨statement⟩ → SHOW_YY CERTIFICATES_YY ALL_YY. [LDF 2013.05.16.]

Log

[LDF 2013.05.16.] Added these rules. They share an action. I've used a named CWEB section and include it under each rule, as for the corresponding rule for showing only the user's certificate.

⟨ Rules 11 ⟩ +≡

statement:[SHOW_YY]CERTIFICATES_YY	⟨ show certificates all action 217 ⟩
statement:[SHOW_YY]CERTIFICATES_YY[ALL_YY]	⟨ show certificates all action 217 ⟩

217.

```
( show certificates all action 217 ) ≡
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread]" << param->thread_ctr << "] In 'yparse', rule" <<
            "'statement: SHOW_YY_CERTIFICATES_YY[ALL_YY] :'" << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    Response_Type response;
    if (param->privileges & Scan_Parser_Parameter_Type::SHOW_CERTIFICATES_PRIVILEGE) {
        response.type = Response_Type::SHOW_CERTIFICATE_TYPE;
        response.int_val = 0; /* Show all certificates. [LDF 2013.05.15.] */
#ifndef DEBUG_COMPILE
        if (param->PARSER_DEBUG) {
            lock_cerr_mutex();
            cerr << "[Thread]" << param->thread_ctr << "] In 'yparse', rule" <<
                "'statement: SHOW_YY_CERTIFICATES_YY[ALL_YY] :'" << endl <<
                "User" << param->username << "(User_ID" << param->user_id << ")" <<
                "has_the\"show_certificates\"privilege." << endl;
            unlock_cerr_mutex();
        } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    } /* if */
    else {
        lock_cerr_mutex();
        cerr << "[Thread]" << param->thread_ctr << "] WARNING! In 'yparse', rule" <<
            "'statement: SHOW_YY_CERTIFICATES_YY[ALL_YY] :'" << endl << "User" <<
            param->username << "(User_ID" << param->user_id << ")" <<
            "doesn't have the\"show_certificates\"privilege." << endl <<
            "Sending warning to client and continuing." << endl;
        unlock_cerr_mutex();
        ++param->warnings_occurred;
        response.type = Response_Type::COMMAND_ONLY_TYPE;
        response.command = "SHOW_CERTIFICATE_RESPONSE_3";
    } /* else */
    param->response_deque.push_back(response);
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

This code is used in section 216.

218. <statement> → SHOW_YY GROUPS_YY ALL_YY. [LDF 2013.06.05.]

Log

[LDF 2013.06.05.] Added this rule.

```

<Rules 11> +≡
statement: SHOW_YY GROUPS_YY ALL_YY
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread]" << param->thread_ctr << "] In 'yparse', rule"
            << "statement: SHOW_YY GROUPS_YY ALL_YY" << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    Response_Type response;
    response.type = Response_Type::COMMAND_ONLY_TYPE;
    temp_strm.str("");
    if (param->privileges & Scan_Parser_Parameter_Type::SHOW_GROUPS_PRIVILEGE) {
#ifndef DEBUG_COMPILE
        if (param->PARSER_DEBUG) {
            lock_cerr_mutex();
            cerr << "[Thread]" << param->thread_ctr << "] In 'yparse', rule"
                << "'statement: SHOW_YY GROUPS_YY ALL_YY'" << endl << "User"
                    << param->username << "(User ID)" << param->user_id << ")"
                    << "has the \"show groups\" privilege." << endl;
            unlock_cerr_mutex();
        } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
        status = Group_Type::get_all_groups(param->mysql_ptr, param->group_vector, param->thread_ctr);
        if (status != 0) {
            lock_cerr_mutex();
            cerr << "[Thread]" << param->thread_ctr << "] ERROR! In 'yparse', rule"
                << "'statement: SHOW_YY GROUPS_YY ALL_YY'" << endl <<
                "'Group_Type::get_all_groups' failed, returning" << status << "."
                << endl << "Sending error message to client and continuing." << endl;
            unlock_cerr_mutex();
            ++param->errors_occurred;
            temp_strm << "SHOW_GROUPS_RESPONSE\1\"Server-side error:\n"
                << "'Group_Type::get_all_groups' failed.\n";
        } /* if (status != 0) */
        else {
#ifndef DEBUG_COMPILE
            if (param->PARSER_DEBUG) {
                lock_cerr_mutex();
                cerr << "[Thread]" << param->thread_ctr << "] In 'yparse', rule"
                    << "'statement: SHOW_YY GROUPS_YY ALL_YY'" << endl <<

```

```

    " 'Group_Type::get_all_groups' succeeded, returning 0." << endl <<
    " param->group_vector.size(): " << param->group_vector.size() << endl;
if (param->group_vector.size() > 0) {
    cerr << " param->group_vector : " << endl;
    for (vector<Group_Type>::iterator iter = param->group_vector.begin();
          iter != param->group_vector.end(); ++iter) {
        iter->show();
    }
    cerr << endl;
} /* if */
unlock_cerr_mutex();
} /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
temp_strm << "SHOW_GROUPS_RESPONSE_0\\\"Success\\\"";
for (vector<Group_Type>::iterator iter = param->group_vector.begin();
       iter != param->group_vector.end(); ++iter) {
    temp_strm << "GROUP_ID\\\" << iter->group_id << "\\\" << "GROUP_NAME\\\" << iter->group_name <<
    " \\\" << "CREATOR_ID\\\" << iter->creator_id << "\\\" << "CREATOR_USERNAME\\\" <<
    iter->creator_username << "\\\" << "CREATED\\\" << iter->created << "\\\";
    for (map<int, pair<string, unsigned int>>::iterator iter_1 = iter->member_id_map.begin();
          iter_1 != iter->member_id_map.end(); ++iter_1) {
        temp_strm << "USER_ID\\\" << iter_1->first << "\\\" << "USER_NAME\\\"\\\" << iter_1->second.first <<
        "\\\" << "PRIVILEGES\\\" << iter_1->second.second << "\\\";
    } /* inner for */
} /* outer for */
#if DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread\\\" << param->thread_ctr << "]\\In\\\"yparse',\\rule\\\" <<
        " 'statement:\\SHOW_YY_GROUPS_YY_ALL_YY': " << endl << "temp_strm.str()\\\" ==\\\" <<
        temp_strm.str() << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    } /* else */
} /* if */
else {
    lock_cerr_mutex();
    cerr << "[Thread\\\" << param->thread_ctr << "]\\WARNING!\\In\\\"yparse',\\rule\\\" <<
    " 'statement:\\SHOW_YY_GROUPS_YY_ALL_YY': " << endl << "User\\\" <<
    param->username << "\\\"(User\\\"ID\\\" << param->user_id << "\\\" )\\\" <<
    " doesn't have the \\\"show\\\"groups\\\"privilege." << endl <<
    " Sending\\\"warning\\\"to\\\"client\\\"and\\\"continuing." << endl;
    unlock_cerr_mutex();
    ++param->warnings_occurred;
    temp_strm << "SHOW_GROUPS_RESPONSE_3\\\"Server-side\\\"error:\\\"";
} /* else */
response.command = temp_strm.str();
temp_strm.str("");
param->response_deque.push_back(response);
param->PARSER_DEBUG = save_PARSER_DEBUG;
}

```

;

219. ⟨statement⟩ → PROCESS_YY PENDING_YY and

220. ⟨statement⟩ → PROCESS_YY PENDING_YY OPERATIONS_YY. [LDF 2013.05.23.]

Log

[LDF 2013.05.23.] Added these rules. They share an action.

⟨Rules 11⟩ +≡

statement:PROCESS_YY_PENDING_YY

⟨process pending operations action 222⟩

221.

⟨Rules 11⟩ +≡

statement:PROCESS_YY_PENDING_YY_OPERATIONS_YY

⟨process pending operations action 222⟩

222. process pending operations action. [LDF 2013.05.23.]

Log

[LDF 2013.05.23.] Added this section.

⟨process pending operations action 222⟩ ≡

{

```
Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
    *>(yyget_extra(parameter));
bool save_PARSER_DEBUG = param->PARSER_DEBUG;
param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
if (param->PARSER_DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << param->thread_ctr << "] " <<
        "In 'yparse', rule "statement:PROCESS_YY_PENDING_YY[OPERATIONS] ." << endl;
    unlock_cerr_mutex();
} /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
Response_Type response;
response.type = Response_Type::PROCESS_PENDING_TYPE;
param->response_deque.push_back(response);
param->PARSER_DEBUG = save_PARSER_DEBUG;
}
```

;

This code is used in sections 220 and 221.

223. <statement> → CREATE_YY HANDLE_YY <handle option list>. [LDF 2013.05.24.]

Log

[LDF 2013.05.24.] Added this rule.

```

⟨Rules 11⟩ +≡
statement : CREATE_YY HANDLE_YY handle_option_list
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'statement:CREATE_YY	HANDLE_YY	handle_option_list'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    Response_Type response;
    response.type = Response_Type::PROCESS_PENDING_TYPE;
    param->response_deque.push_back(response);
    response.type = Response_Type::CREATE_HANDLE_TYPE;
    response.string_vector = param->string_vector;
    response.pid_str = param->pid_str;
    response.pid_prefix_str = param->pid_prefix_str;
    response.pid_suffix_str = param->pid_suffix_str;
    response.pid_institute_str = param->pid_institute_str;
    if (response.pid_prefix_str.empty()) response.pid_prefix_str = param->default_handle_prefix;
#ifndef 0 /* Not putting the institute name into the handle [LDF 2013.05.24.] */
    if (response.pid_institute_str.empty()) response.pid_institute_str = param->default_institute_name;
#endif
    lock_cerr_mutex();
    cerr << "param->default_handle_prefix==" << param->default_handle_prefix << endl <<
        "response.pid_str==" << response.pid_str << endl << "response.pid_prefix_str==" <<
        response.pid_prefix_str << endl << "response.pid_suffix_str==" << response.pid_suffix_str <<
        endl << "response.pid_institute_str==" << response.pid_institute_str << endl;
    unlock_cerr_mutex();
    param->response_deque.push_back(response);
    param->pid_str = param->pid_prefix_str = param->pid_suffix_str = param->pid_institute_str = "";
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

224. ⟨statement⟩ → ADD_YY HANDLE_VALUE_YY ⟨handle option list⟩. [LDF 2013.06.15.]

Log

[LDF 2013.06.15.] Added this rule.

```

⟨Rules 11⟩ +≡
statement: ADD_YY HANDLE_VALUE_YY handle_option_list
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'statement: ADD_YY HANDLE_VALUE_YY handle_option_list'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    Response_Type response;
    response.pid_str = param->pid_str;
    response.hvt = param->hvt;
    response.hvt.show("response.hvt:");
    param->hvt.clear();
    param->pid_str = "";
    response.type = Response_Type::ADD_HANDLE_VALUE_TYPE;
    param->response_deque.push_back(response);
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

225. ⟨handle option list⟩.

Log

[LDF 2013.05.24.] Added this type declaration.

```

⟨Token and type declarations 128⟩ +≡
```

%type <int_value> handle_option_list	%type <int_value> handle_option
--------------------------------------	---------------------------------

226. ⟨handle option list⟩ → Empty.

Log

[LDF 2013.05.24.] Added this rule.

```

⟨Rules 11⟩ +≡
[handle_option_list:/*Empty*/]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "In 'yparse', rule 'handle_option_list:/*Empty*/'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->pid_str = param->pid_prefix_str = param->pid_suffix_str = param->pid_institute_str = "";
    param->hvt.clear();
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

227. ⟨handle option list⟩ → ⟨handle option list⟩ ⟨handle option⟩.

Log

[LDF 2013.05.24.] Added this rule.

```

⟨Rules 11⟩ +≡
[handle_option_list:handle_option_list,handle_option]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "In 'yparse', rule 'handle_option_list:handle_option_list,handle_option'." <<
            endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

228. <handle option list> → STRING_YY. [LDF 2013.05.24.]

Log

[LDF 2013.06.15.] Added this rule.

```
(Rules 11) +≡
[handle_option: STRING_YY]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
    *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "In 'yparse', rule 'handle_option:STRING_YY'(PID_string)." <<
            endl << "'STRING_YY'(1): " << $1 << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->pid_str = $1;
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

229. <handle option list> → PID_YY STRING_YY. [LDF 2013.05.24.]

Log

[LDF 2013.05.24.] Added this rule.

```
(Rules 11) +≡
[handle_option: PID_YY STRING_YY]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
    *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "In 'yparse', rule 'handle_option:PID_YY STRING_YY'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->pid_str = $2;
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

230. <handle option list> → IDX_YY INTEGER_YY. [LDF 2013.06.16.]

Log

[LDF 2013.06.16.] Added this rule.

```

⟨Rules 11⟩ +≡
[handle_option:IDX_YY_INTEGER_YY]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
            "In 'yparse', rule 'handle_option:IDX_YY_INTEGER_YY'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->hvt.idx = $2;
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

231. <handle option list> → TYPE_YY STRING_YY. [LDF 2013.06.16.]

Log

[LDF 2013.06.16.] Added this rule.

```

⟨Rules 11⟩ +≡
[handle_option:TYPE_YY_STRING_YY]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
            "In 'yparse', rule 'handle_option:TYPE_YY_STRING_YY'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->hvt.type = $2;
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

232. ⟨handle option list⟩ → DATA_YY STRING_YY. [LDF 2013.06.16.]

Log

[LDF 2013.06.16.] Added this rule.

```
⟨Rules 11⟩ +≡
[handle_option:DATA_YY STRING_YY]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "In 'yparse', rule 'handle_option:DATA_YY STRING_YY'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->hvt.data_str = $2;
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

233. <statement> → DELETE_YY HANDLE_YY <delete handle option list> STRING_YY [LDF 2013.07.04.]

Log

[LDF 2013.07.04.] Added this rule.
 [LDF 2013.09.16.] Put <delete handle option list> before STRING_YY.

<Rules 11> +≡

```

statement : DELETE_YY HANDLE_YY delete_handle_option_list STRING_YY
{
  Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
    *>(yyget_extra(parameter));
  bool save_PARSER_DEBUG = param->PARSER_DEBUG;
  param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
  if (param->PARSER_DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << param->thread_ctr << "] " <<
      "'statement:DELETE_YYHANDLE_YYdelete_handle_option_listSTRING_YY'." <<
      endl << "'STRING_YY'(Handle)" << endl << "$3" << endl <<
      "'delete_handle_option_list'" << endl << hex << "$4" << "(hex)" << dec <<
      "'param->delay_value'" << endl << param->delay_value << endl;
    unlock_cerr_mutex();
  } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
  Response_Type response;
  response.pid_options = $3;
  response.pid_str = $4;
  response.delay_value = param->delay_value;
  param->delay_value = 0UL;
  response.type = Response_Type::DELETE_HANDLE_TYPE;
  param->response_deque.push_back(response);
  param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

234. <delete handle option list> and <delete handle option>. [LDF 2013.07.04.]

Log

[LDF 2013.07.04.] Added these type declarations.

<Token and type declarations 128> +≡

```
%type <uint_value> delete_handle_option_list %type <uint_value> delete_handle_option
```

235. ⟨delete handle option list⟩ → Empty. [LDF 2013.07.04.]

Log

[LDF 2013.07.04.] Added this rule.

```
<Rules 11> +≡
  delete_handle_option_list:/*Empty*/
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'delete_handle_option_list:/*Empty*/'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    $$ = 0U;
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

236. <delete handle option list> → <delete handle option list> <delete handle option> [LDF 2013.07.04.]

Log

[LDF 2013.07.04.] Added this rule.

```

⟨Rules 11⟩ +≡
  delete_handle_option_list: delete_handle_option_list delete_handle_option
  {
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
    *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
      lock_cerr_mutex();
      cerr << "[Thread" << param->thread_ctr << "] " <<
        "'delete_handle_option_list:delete_handle_option_list(delete_handle_option'." <<
        endl << "'delete_handle_option'" << hex << "$1" << "(hex)" <<
        endl << "'delete_handle_option'" << "$2" << "(hex)" << endl <<
        "Result(OR)" << endl << ("$1" | "$2") << "(hex)" << dec << endl;
      unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    $$ = $1 | $2;
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

237. <delete handle option list> → <delete handle option list> <sub-delay option> [LDF 2013.08.26.]

Log

[LDF 2013.08.26.] Added this rule.

```

⟨Rules 11⟩ +≡
  delete_handle_option_list: delete_handle_option_list sub_delay_option
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'delete_handle_option_list: delete_handle_option_list sub_delay_option'." <<
            endl << "'delete_handle_option_list'" <($1)<= " << hex << $1 << "(hex)" << endl <<
            dec << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    $$ = $1;
    if (param->delay_value > 0UL) {
        $$ &= ~1UL; /* Unset “immediate” bit. [LDF 2013.08.26.] */
    }
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

238. ⟨delete handle option⟩ → IMMEDIATE_YY. [LDF 2013.07.04.]

Log

[LDF 2013.07.04.] Added this rule.

[LDF 2013.08.26.] Now setting *param*→*delay_value* = 0_{UL}.

```

⟨Rules 11⟩ +≡
  delete_handle_option: IMMEDIATE_YY
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " << "'delete_handle_option: IMMEDIATE_YY'." <<
            endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    $$ = 1U;
    param->delay_value = 0UL;
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

239. ⟨statement⟩ → UNDELETE_YY HANDLE_YY STRING_YY. [LDF 2013.08.21.]

Log

[LDF 2013.08.21.] Added this rule.

```

⟨Rules 11⟩ +≡
statement: UNDELETE_YY HANDLE_YY STRING_YY
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] In 'yparse', rule"
            << "statement: UNDELETE_YY,HANDLE_YY,STRING_YY:'" << endl <<
            "'STRING_YY' ($3)(handle)==" << $3 << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    Response_Type response;
    response.type = Response_Type::PROCESS_PENDING_TYPE;
    param->response_deque.push_back(response);
    response.type = Response_Type::UNDELETE_HANDLE_TYPE;
    response.pid_str = $3;
    param->response_deque.push_back(response);
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

240. ⟨statement⟩ → DELETE_YY HANDLE_VALUE_YY ⟨delete handle option list⟩ STRING_YY
 ⟨statement⟩ → DELETE_YY HANDLE_VALUES_YY ⟨delete handle option list⟩ STRING_YY
 ⟨statement⟩ → DELETE_YY HANDLE_VALUE_YY ⟨delete handle option list⟩ HANDLE_VALUE_SPECIFICATION_YY
 ⟨statement⟩ → DELETE_YY HANDLE_VALUES_YY ⟨delete handle option list⟩ HANDLE_VALUE_SPECIFICATION_YY

[LDF 2013.07.17.]

Log

[LDF 2013.07.17.] Added this rule.

[LDF 2013.09.11.] Added the rules with HANDLE_VALUES_YY. These four rules share an action. Added ⟨delete handle option list⟩ to all of these rules.

[LDF 2013.09.16.] Put ⟨delete handle option list⟩ before STRING_YY or HANDLE_VALUE_SPECIFICATION_YY in all of these rules.

⟨Rules 11⟩ +≡

```
statement : DELETE_YY HANDLE_VALUE_YY delete_handle_option_list STRING_YY ⟨Delete handle
value or values 241⟩
statement : DELETE_YY HANDLE_VALUES_YY delete_handle_option_list STRING_YY ⟨Delete
handle value or values 241⟩
statement : DELETE_YY HANDLE_VALUE_YY delete_handle_option_list HANDLE_VALUE_SPECIFICATION_YY
⟨Delete handle value or values 241⟩
statement : DELETE_YY HANDLE_VALUES_YY delete_handle_option_list HANDLE_VALUE_SPECIFICATION_YY
⟨Delete handle value or values 241⟩
```

241.

Log

[LDF 2013.09.11.] Added this section.
[LDF 2013.09.16.] Put <delete handle option list> before STRING_YY or HANDLE_VALUE_SPECIFICATION_YY in all of these rules.

This code is used in section 240.

242. ⟨statement⟩ → UNDELETE_YY HANDLE_VALUE_YY STRING_YY
 ⟨statement⟩ → UNDELETE_YY HANDLE_VALUES_YY STRING_YY
 ⟨statement⟩ → UNDELETE_YY HANDLE_VALUE_YY HANDLE_VALUE_SPECIFICATION_YY
 ⟨statement⟩ → UNDELETE_YY HANDLE_VALUES_YY HANDLE_VALUE_SPECIFICATION_YY
 [LDF 2013.07.17.]

Log

[LDF 2013.07.17.] Added these rules.

⟨Rules 11⟩ +≡

```
statement : UNDELETE_YY HANDLE_VALUE_YY STRING_YY ⟨ Undelete handle value or values 243 ⟩
statement : UNDELETE_YY HANDLE_VALUES_YY STRING_YY ⟨ Undelete handle value or values 243 ⟩
statement : UNDELETE_YY HANDLE_VALUE_YY HANDLE_VALUE_SPECIFICATION_YY ⟨ Undelete handle
  value or values 243 ⟩
statement : UNDELETE_YY HANDLE_VALUES_YY HANDLE_VALUE_SPECIFICATION_YY ⟨ Undelete handle
  value or values 243 ⟩
```

243. Undelete handle value or values. [LDF 2013.09.11.]

Log

[LDF 2013.09.11.] Added this section.

⟨ Undelete handle value or values 243 ⟩ ≡

```
{
  Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
    >(yyget_extra(parameter));
  bool save_PARSER_DEBUG = param->PARSER_DEBUG;
  param->PARSER_DEBUG = true; /* false save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
  if (param->PARSER_DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << param->thread_ctr << "]"
      << "statement : UNDELETE_YY HANDLE_VALUE_YY STRING_YY" << endl
      << "or 'statement : UNDELETE_YY HANDLE_VALUES_YY STRING_YY'" << endl
      << "or 'statement : UNDELETE_YY HANDLE_VALUE_YY HANDLE_VALUE_SPECIFICATION_YY'" <<
      endl << "or 'statement : UNDELETE_YY HANDLE_VALUE_YY HANDLE_VALUES_SP\ECIFICATION_YY'" << endl
      << "STRING_YY" << "HANDLE_VALUES_SPECIFICATION_YY" << "=" << $3 << endl;
    unlock_cerr_mutex();
  } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
  Response_Type response;
  response.type = Response_Type::PROCESS_PENDING_TYPE;
  param->response_deque.push_back(response);
  response.type = Response_Type::UNDELETE_HANDLE_VALUE_TYPE;
  response.string_val = $3;
  param->response_deque.push_back(response);
  param->PARSER_DEBUG = save_PARSER_DEBUG;
}
```

;

This code is used in section 242.

244. ⟨statement⟩ → UNDELETE_YY ⟨undelete option list⟩ ⟨filename list⟩. [LDF 2013.08.15.]

Log

[LDF 2013.08.15.] Added this rule.

```

⟨Rules 11⟩ +≡
statement: UNDELETE_YY undelete_option_list filename_list
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] In 'yparse', rule"
            << "statement: UNDELETE_YY undelete_option_list filename_list':"
            << endl << "undelete_option_list' ($2) == "
            << oct << "0" << $2 << "(octal)" << dec << endl
            << "'filename_vector' == "
            << endl;
        for (vector<string>::const_iterator iter = param->filename_vector.begin();
            iter != param->filename_vector.end(); ++iter) {
            cerr << *iter << endl;
        }
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    Response_Type response;
    response.type = Response_Type::PROCESS_PENDING_TYPE;
    param->response_deque.push_back(response);
    response.type = Response_Type::UNDELETE_FILE_TYPE;
    response.string_vector = param->filename_vector;
    response.options = $2;
    param->response_deque.push_back(response);
    param->filename_vector.clear();
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

245. ⟨undelete option list⟩. [LDF 2013.08.15.]

Log

[LDF 2013.08.15.] Added this section.

```

⟨Token and type declarations 128⟩ +≡
%type<uint_value> undelete_option_list %type<uint_value> undelete_option
```

246. ⟨undelete option list⟩ → Empty [LDF 2013.08.15.]

Log

[LDF 2013.08.15.] Added this rule.

```

⟨Rules 11⟩ +≡
  undelete_option_list:/*Empty*/
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] In 'yparse', rule"
            << "undelete_option_list:/*Empty*/" << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    $$ = 0U;
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

247. ⟨undelete option list⟩ → ⟨undelete option list⟩ ⟨undelete option⟩ [LDF 2013.08.15.]

Log

[LDF 2013.08.15.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
  undelete_option_list : undelete_option_list undelete_option
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
    Response_Type response;
    response.type = Response_Type::COMMAND_ONLY_TYPE;
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] In 'yyparse', rule"
            << "undelete_option_list: undelete_option_list undelete_option" << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    if ($2 ≡ 1U ∧ $1 ∧ 2U) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] WARNING!" << endl << "In 'yyparse': Opti\
            on 'database' used and option 'database-only'" << "already set." << endl <<
            "The last option used takes precedence: Setting 'database' option" << endl <<
            "and unsetting 'database-only'." << endl;
        unlock_cerr_mutex();
        temp_strm.str();
        temp_strm << "DATABASE OPTION RESPONSE 2 \"WARNING! Multiple database op\
            tions used." << endl << "The last one takes precedence." << endl <<
            "Setting 'database' and unsetting 'database-only'.\"";
        response.command = temp_strm.str();
        temp_strm.str();
        param->response_deque.push_back(response);
        ++param->warnings_occurred;
    ;
    $$ &= ~2U;
    $$ |= 1U;
}
else if ($2 ≡ 2U ∧ $1 ∧ 1U) {
    lock_cerr_mutex();
    cerr << "[Thread" << param->thread_ctr << "] WARNING!" << endl << "In 'yyparse': Opti\
        on 'database-only' used and option 'database'" << "already set." << endl <<
        "The last option used takes precedence: Setting 'database-only' option" <<
        endl << "and unsetting 'database'." << endl;
    unlock_cerr_mutex();
    temp_strm.str();
    temp_strm << "DATABASE OPTION RESPONSE 2 \"WARNING! Multiple database op\
        tions used." << endl << "The last one takes precedence." << endl <<
        "Setting 'database-only' and unsetting 'database'."";
}

```

```

response.command = temp_strm.str();
temp_strm.str();
++param->warnings_occurred;
;
param->response_deque.push_back(response);
[$$] &= ~1U;
[$$] |= 2U;
}
else [$$] = [$1] | [$2];
param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

248. ⟨undelete option⟩ → ⟨database option⟩ [LDF 2013.08.15.]

Log

[LDF 2013.08.15.] Added this rule.

```

⟨Rules 11⟩ +≡
[undelete_option:database_option]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
    Response_Type response;
    response.type = Response_Type::COMMAND_ONLY_TYPE;
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] In 'yyparse', rule"
            << "undelete_option:database_option" << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    [$$] |= [$1];
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

249. ⟨undelete option⟩. [LDF 2013.08.15.]

Log

[LDF 2013.08.15.] Added this section.

```

⟨Token and type declarations 128⟩ +≡
%type<uint_value>database_option
```

250. <database option> → DATABASE_YY [LDF 2013.08.15.]

Log

[LDF 2013.08.15.] Added this rule.

```

⟨Rules 11⟩ +≡
[database_option:DATABASE_YY]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] In 'yyparse', rule"
            << "database_option:DATABASE_YY" << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    $$ = 1U;
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

251. <database option> → DATABASE_ONLY_YY [LDF 2013.08.15.]

Log

[LDF 2013.08.15.] Added this rule.

```

⟨Rules 11⟩ +≡
[database_option:DATABASE_ONLY_YY]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] In 'yyparse', rule"
            << "database_option:DATABASE_ONLY_YY" << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    $$ = 2U;
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

252. ⟨statement⟩ → DUMMY_STATEMENT_YY INTEGER_YY. [LDF 2013.04.05.]

Log

[LDF 2013.04.05.] Added this rule. It's used for testing.

```

⟨Rules 11⟩ +≡
statement: DUMMY_STATEMENT_YY INTEGER_YY
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'statement:DUMMY_STATEMENT_YY INTEGER_YY'." << endl << "'INTEGER_YY'" <=
            [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    lock_cout_mutex();
    lock_cerr_mutex();
    cout << "Dummy statement--" << endl << "Code number:" << [$2] << endl;
    unlock_cerr_mutex();
    unlock_cout_mutex();
    temp_strm.str("");
    temp_strm << "DUMMY_STATEMENT RESPONSE" << 0;
    Response_Type response;
    response.type = Response_Type::COMMAND_ONLY_TYPE;
    response.command = temp_strm.str();
    param->response_deque.push_back(response);
    temp_strm.str("");
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

253. ⟨statement⟩ → DUMMY_STATEMENT_YY TIME_SPECIFICATION_YY. [LDF 2013.08.19.]

Log

[LDF 2013.08.19.] Added this rule. It's used for testing.

```

⟨Rules 11⟩ +≡
statement: DUMMY_STATEMENT_YY TIME_SPECIFICATION_YY
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = true; /* false save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
            << "statement:DUMMY_STATEMENT_YY,TIME_SPECIFICATION_YY."
            << endl <<
            "TIME_SPECIFICATION_YY" <=> $2 << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    lock_cout_mutex();
    lock_cerr_mutex();
    cout << "Dummy(statement--)" << endl << "Time(specification:" <=> $2 << endl;
    unlock_cerr_mutex();
    unlock_cout_mutex();
    temp_strm.str("");
    int days = 0;
    int hours = 0;
    int minutes = 0;
    int seconds = 0;
    seconds = $2;
    days = seconds/86400;
    seconds %= 86400;
    hours = seconds/3600;
    seconds %= 3600;
    minutes = seconds/60;
    seconds %= 60;
    temp_strm << "DUMMY_STATEMENT,RESPONSE\""
        << "days" <=> "uuuu" << days << endl <<
        "hours" <=> "uuu" << hours << endl << "minutes" <=> "u" << minutes << endl << "seconds" <=> "u"
        << seconds << "\";
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
            << "statement:DUMMY_STATEMENT_YY,TIME_SPECIFICATION_YY:" << endl <<
            "days" <=> "u" << days << endl << "hours" <=> "u" << hours << endl << "minutes" <=> "u"
            << minutes << endl << "seconds" <=> "u" << seconds << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */

```

```

Response_Type response;
response.type = Response_Type::COMMAND_ONLY_TYPE;
response.command = temp_strm.str();
param->response_deque.push_back(response);
temp_strm.str("");
param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

254. ⟨statement⟩ → DUMMY_STATEMENT_YY STRING_YY. [LDF 2013.07.17.]

Log

[LDF 2013.07.17.] Added this rule. It's used for testing.

```

⟨Rules 11⟩ +≡
[statement:DUMMY_STATEMENT_YY STRING_YY]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'statement:DUMMY_STATEMENT_YY STRING_YY'." << endl << "'STRING_YY'" <=
            [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    lock_cout_mutex();
    lock_cerr_mutex();
    cout << "Dummy" << statement << endl << "String:" << [$2] << endl;
    unlock_cerr_mutex();
    unlock_cout_mutex();
    temp_strm.str("");
    temp_strm << "DUMMY_STATEMENT" << RESPONSE << "0";
    Response_Type response;
    response.type = Response_Type::COMMAND_ONLY_TYPE;
    response.command = temp_strm.str();
    param->response_deque.push_back(response);
    temp_strm.str("");
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

255. ⟨statement⟩ → DUMMY_STATEMENT_YY RESPONSE_YY INTEGER_YY. [LDF 2013.04.19.]

Log

[LDF 2013.04.19.] Added this rule.

```

⟨Rules 11⟩ +≡
statement: DUMMY_STATEMENT_YY RESPONSE_YY INTEGER_YY
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(yyget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] In 'yparse', rule"
            << "statement:DUMMY_STATEMENT_YY,RESPONSE_YY,INTEGER_YY."
            << endl << "'INTEGER_YY'" << $3 << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    lock_cout_mutex();
    lock_cerr_mutex();
    cout << "Dummy(statement,response--)" << endl << "Response,code:" << $3 << endl;
    unlock_cerr_mutex();
    unlock_cout_mutex();
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

256.

⟨Garbage 256⟩ ≡ /* Empty */

See also section 559.

This code is used in sections 257 and 560.

257. Putting parser together.

```
%{ }<Include files 3>
using namespace std;
using namespace gwrdifpk;
static stringstream temp_strm;
static int status;
<Declarations of additional functions 125>
#ifndef 0
<Garbage 256>
#endif
%}> }<Options 5>
<union declaration 127>
<Token and type declarations 128>
%%> <Rules 11>
```

258. Client Scanner. [LDF 2012.07.30.]

259.

```
<Include files 3> +≡
#include <stdlib.h>
#include <stdio.h>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <map>
#include <string>
#include <time.h>
#include <math.h>
#include <sstream>
#include <vector>
#include <deque>
#include <stack>
#include <set>
#include <gcrypt.h> /* for gcry-control */
#include <gnutls/gnutls.h>
#include <pthread.h>
#include <mysql.h>
#include <expat.h>
#if HAVE_CONFIG_H
#include <config.h>
#endif
#define NAME_LEN
#define LOCAL_HOST
#include "glblcnst.h++"
#include "glblvrbl.h++"
#include "utilfncs.h++"
#include "grouptp.h++"
#include "hndlvltp.h++"
#include "irdsavtp.h++"
#include "rspnstp.h++"
#include "irdsobtp.h++"
#include "hndltype.h++"
#include "dcmtddtp.h++"
#include "x509cert.h++"
#include "dstngnmt.h++"
#include "prsrlnt.h++"
#include "scprpmtp.h++"
```

260. Options.

```
<Options 5> +≡
[%option_header-file="scnrclnt.hxx"] [%option_bison-bridge] [%option_reentrant] [%option_prefix="zz"]
```

261. Local variables for **zzlex**. [LDF 2012.07.30.]

```
<Local variables for zzlex 261> =
  bool SCANNER_DEBUG = false; /* true */
```

This code is used in section 397.

262. Code to be executed each time **zzlex** is entered. This code must be indented or it causes an error when FLEX is run. The start condition on entry to **zzlex** can be set here. [LDF 2012.07.30.]

`<Execute on entry to zzlex 262> =`

```
    \Scan_Parse_Parameter_Type* param = static_cast<Scan_Parse_Parameter_Type*>(yyextra);
```

This code is used in section 397.

263. Rules.

264. Punctuation.

265. Whitespace.

`<Rules 11> +≡`

```
[[ :space:] \x0d]+ { }
```

266. Comments. [LDF 2012.07.30.]

`<Rules 11> +≡`

```
#.*$ { }
```

```
if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'zzlex': Comment." << endl;
    unlock_cerr_mutex();
}
/* Ignore */
}
```

267. Integer. [LDF 2012.06.27.]

Log

[LDF 2012.06.27.] Added this rule.

`<Rules 11> +≡`

```
[-]?[0-9][0-9]* { }
```

```
if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'zzlex': integer:" << yytext << endl << "Returning 'INTEGER_ZZ'." << endl;
    unlock_cerr_mutex();
}
sscanf(yytext, "%d", &yyval.int_value);
return INTEGER_ZZ; }
```

268. Unsigned Integer. [LDF 2012.10.12.]

Log

[LDF 2012.10.12.] Added this rule.

```
⟨Rules 11⟩ +≡
[0-9] [0-9]* [uU] {
if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'zzlex': unsigned_integer: " << yytext << endl <<
        "Returning 'UNSIGNED_INTEGER_ZZ'." << endl;
    unlock_cerr_mutex();
}
sscanf(yytext, "%u", &yyval.uint_value);
return UNSIGNED_INTEGER_ZZ; }
```

269. Unsigned Long Integer. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

```
⟨Rules 11⟩ +≡
[0-9] [0-9]* [uU] [lL] {
if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'zzlex': unsigned_integer: " << yytext << endl <<
        "Returning 'UNSIGNED_LONG_INTEGER_ZZ'." << endl;
    unlock_cerr_mutex();
}
sscanf(yytext, "%lu", &yyval.ulint_value);
return UNSIGNED_LONG_INTEGER_ZZ; }
```

270. END. [LDF 2012.07.30.]

```
⟨Rules 11⟩ +≡
[END|end] {
    return END_ZZ; }
```

271. PWD. [LDF 2012.07.30.]

```
⟨Rules 11⟩ +≡
[PWD|pwd] {
    return PWD_ZZ; }
```

272. LS. [LDF 2012.09.06.]

Log

[LDF 2012.09.06.] Added this rule.

```
⟨Rules 11⟩ +≡
[LS|ls] {
    return LS_ZZ; }
```

273. GET. [LDF 2012.09.26.]

Log

[LDF 2012.09.26.] Added this rule.

$\langle \text{Rules } 11 \rangle + \equiv$

`GET|get{|} return GET_ZZ; []`

274. PUT. [LDF 2012.09.26.]

Log

[LDF 2012.09.26.] Added this rule.

$\langle \text{Rules } 11 \rangle + \equiv$

`PUT|put{|} return PUT_ZZ; []`

275. RECEIVE. [LDF 2012.09.27.]

Log

[LDF 2012.09.27.] Added this rule.

$\langle \text{Rules } 11 \rangle + \equiv$

`RECEIVE|receive{|} return RECEIVE_ZZ; []`

276. SEND. [LDF 2012.09.27.]

Log

[LDF 2012.09.27.] Added this rule.

$\langle \text{Rules } 11 \rangle + \equiv$

`SEND|send{|} return SEND_ZZ; []`

277. TAN. [LDF 2013.06.01.]

Log

[LDF 2013.06.01.] Added this rule.

$\langle \text{Rules } 11 \rangle + \equiv$

`TAN|tan{|} return TAN_ZZ; []`

278. LIST. [LDF 2013.06.01.]

Log

[LDF 2013.06.01.] Added this rule.

$\langle \text{Rules } 11 \rangle + \equiv$

`LIST|list{|} return LIST_ZZ; []`

279. SENDING. [LDF 2012.12.13.]

Log

[LDF 2012.12.13.] Added this rule.

$\langle \text{Rules } 11 \rangle + \equiv$
SENDING | sending ↳ { return SENDING_ZZ; }

280. ADD. [LDF 2012.12.14.]

Log

[LDF 2012.12.14.] Added this rule.

$\langle \text{Rules } 11 \rangle + \equiv$
ADD | add ↳ { return ADD_ZZ; }

281. STORE. [LDF 2013.03.07.]

Log

[LDF 2013.03.07.] Added this rule.

$\langle \text{Rules } 11 \rangle + \equiv$
STORE | store ↳ { return STORE_ZZ; }

282. FILE. [LDF 2012.10.12.]

Log

[LDF 2012.10.12.] Added this rule.

$\langle \text{Rules } 11 \rangle + \equiv$
FILE | file ↳ { return FILE_ZZ; }

283. CLIENT_SIDE_FILENAME. [LDF 2012.11.19.]

Log

[LDF 2012.11.19.] Added this rule.

$\langle \text{Rules } 11 \rangle + \equiv$
CLIENT_SIDE_FILENAME | client_side_filename ↳ { return CLIENT_SIDE_FILENAME_ZZ; }

284. CLIENT_SIDE_FILENAME. [LDF 2012.11.19.]

Log

[LDF 2012.11.19.] Added this rule.

$\langle \text{Rules } 11 \rangle + \equiv$
OVERWRITE | overwrite ↳ { return OVERWRITE_ZZ; }

285. HANDLE. [LDF 2012.10.12.]

Log

[LDF 2012.10.12.] Added this rule.

$\langle \text{Rules } 11 \rangle +\equiv$

$\boxed{\backslash+?(HANDLE|handle)\u{ }\{ \text{return HANDLE_ZZ; } \}}$

286. HANDLES. [LDF 2012.10.12.]

Log

[LDF 2012.10.12.] Added this rule.

$\langle \text{Rules } 11 \rangle +\equiv$

$\boxed{HANDLES|handles}\u{ }\{ \text{return HANDLES_ZZ; } \}$

287. HANDLE_VALUE. [LDF 2012.10.12.]

Log

[LDF 2012.10.12.] Added this rule.

$\langle \text{Rules } 11 \rangle +\equiv$

$\boxed{\backslash+?(HANDLE_VALUE|handle_value)\u{ }\{ \text{return HANDLE_VALUE_ZZ; } \}}$

288. HANDLE_VALUES. [LDF 2012.10.12.]

Log

[LDF 2012.10.12.] Added this rule.

$\langle \text{Rules } 11 \rangle +\equiv$

$\boxed{HANDLE_VALUES|handle_values}\u{ }\{ \text{return HANDLE_VALUES_ZZ; } \}$

289. IDX. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

$\langle \text{Rules } 11 \rangle +\equiv$

$\boxed{IDX|idx}\u{ }\{ \text{return IDX_ZZ; } \}$

290. TYPE. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

$\langle \text{Rules } 11 \rangle +\equiv$

$\boxed{TYPE|type}\u{ }\{ \text{return TYPE_ZZ; } \}$

291. DATA. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

$\langle \text{Rules } 11 \rangle +\equiv$

`DATA|data_{ return DATA_ZZ; }`

292. DATA_LENGTH. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

$\langle \text{Rules } 11 \rangle +\equiv$

`DATA_LENGTH|data_length_{ return DATA_LENGTH_ZZ; }`

293. DATA_HEXL_ENCODED. [LDF 2013.04.26.]

Log

[LDF 2013.04.26.] Added this rule.

$\langle \text{Rules } 11 \rangle +\equiv$

`DATA_HEXL_ENCODED|data_hexl_encoded_{ return DATA_HEXL_ENCODED_ZZ; }`

294. TTL_TYPE. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

$\langle \text{Rules } 11 \rangle +\equiv$

`TTL_TYPE|ttl_type_{ return TTL_TYPE_ZZ; }`

295. TTL. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

$\langle \text{Rules } 11 \rangle +\equiv$

`TTL|ttl_{ return TTL_ZZ; }`

296. TIMESTAMP. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

$\langle \text{Rules } 11 \rangle +\equiv$

`TIMESTAMP|timestamp_{ return TIMESTAMP_ZZ; }`

297. REFS. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

$\langle \text{Rules } 11 \rangle + \equiv$

`REFS|refs_{ return REFS_ZZ; }`

298. REFS_LENGTH. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

$\langle \text{Rules } 11 \rangle + \equiv$

`REFS_LENGTH|refs_length_{ return REFS_LENGTH_ZZ; }`

299. REFS_HEXL_ENCODED. [LDF 2013.04.28.]

Log

[LDF 2013.04.28.] Added this rule.

$\langle \text{Rules } 11 \rangle + \equiv$

`REFS_HEXL_ENCODED|refs_hexl_encoded_{ return REFS_HEXL_ENCODED_ZZ; }`

300. ADMIN_READ. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

$\langle \text{Rules } 11 \rangle + \equiv$

`ADMIN_READ|admin_read_{ return ADMIN_READ_ZZ; }`

301. ADMIN_WRITE. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

$\langle \text{Rules } 11 \rangle + \equiv$

`ADMIN_WRITE|admin_write_{ return ADMIN_WRITE_ZZ; }`

302. PUB_READ. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

$\langle \text{Rules } 11 \rangle + \equiv$

`PUB_READ|pub_read_{ return PUB_READ_ZZ; }`

303. PUB_WRITE. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

$\langle \text{Rules } 11 \rangle +\equiv$

`PUB_WRITE|pub_write_{ return PUB_WRITE_ZZ; }`

304. HANDLE_ID. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

$\langle \text{Rules } 11 \rangle +\equiv$

`HANDLE_ID|handle_id_{ return HANDLE_ID_ZZ; }`

305. HANDLE_VALUE_ID. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

$\langle \text{Rules } 11 \rangle +\equiv$

`HANDLE_VALUE_ID|handle_value_id_{ return HANDLE_VALUE_ID_ZZ; }`

306. IRODS_OBJECT_ID. [LDF 2013.08.26.]

Log

[LDF 2013.08.26.] Added this rule.

$\langle \text{Rules } 11 \rangle +\equiv$

`IRODS_OBJECT_ID|irods_object_id_{ return IRODS_OBJECT_ID_ZZ; }`

307. CREATE. [LDF 2013.05.24.]

Log

[LDF 2013.05.24.] Added this rule.

$\langle \text{Rules } 11 \rangle +\equiv$

`CREATE|create_{ return CREATE_ZZ; }`

308. CREATED. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

$\langle \text{Rules } 11 \rangle +\equiv$

`CREATED|created_{ return CREATED_ZZ; }`

309. DELETED. [LDF 2013.07.15.]

Log

[LDF 2013.07.15.] Added this rule.

$\langle \text{Rules } 11 \rangle + \equiv$

`DELETED|deleted{|} return DELETED_ZZ; []`

310. MARKED_FOR_DELETION. [LDF 2013.08.22.]

Log

[LDF 2013.07.15.] [LDF 2013.08.22.]

$\langle \text{Rules } 11 \rangle + \equiv$

`MARKED_FOR_DELETION|marked_for_deletion{|} return MARKED_FOR_DELETION_ZZ; []`

311. LAST_MODIFIED. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

$\langle \text{Rules } 11 \rangle + \equiv$

`LAST_MODIFIED|last_modified{|} return LAST_MODIFIED_ZZ; []`

312. DELETE_FROM_DATABASE_TIMESTAMP. [LDF 2013.08.26.]

Log

[LDF 2013.08.26.] Added this rule.

$\langle \text{Rules } 11 \rangle + \equiv$

`DELETE_FROM_DATABASE_TIMESTAMP|delete_from_database_timestamp{|} return
DELETE_FROM_DATABASE_TIMESTAMP_ZZ; []`

313. CREATED_BY_USER. [LDF 2013.02.28.]

Log

[LDF 2013.02.28.] Added this rule.

$\langle \text{Rules } 11 \rangle + \equiv$

`CREATED_BY_USER|created_by_user{|} return CREATED_BY_USER_ZZ; []`

314. RESPONSE. [LDF 2012.09.06.]

Log

[LDF 2012.09.06.] Added this rule.

$\langle \text{Rules } 11 \rangle + \equiv$

`RESPONSE|response{|} return RESPONSE_ZZ; []`

315. FAILED. [LDF 2012.09.10.]

Log

[LDF 2012.09.10.] Added this rule.

$\langle \text{Rules } 11 \rangle + \equiv$
FAILED | failed { return FAILED_ZZ; }

316. SUCCEEDED. [LDF 2012.09.10.]

Log

[LDF 2012.09.10.] Added this rule.

$\langle \text{Rules } 11 \rangle + \equiv$
SUCCEEDED | succeeded { return SUCCEEDED_ZZ; }

317. METADATA. [LDF 2012.10.12.]

Log

[LDF 2012.10.12.] Added this rule.

$\langle \text{Rules } 11 \rangle + \equiv$
METADATA | metadata { return METADATA_ZZ; }

318. ATTRIBUTE. [LDF 2012.10.12.]

Log

[LDF 2012.10.12.] Added this rule.

$\langle \text{Rules } 11 \rangle + \equiv$
ATTRIBUTE | attribute { return ATTRIBUTE_ZZ; }

319. VALUE. [LDF 2012.10.12.]

Log

[LDF 2012.10.12.] Added this rule.

$\langle \text{Rules } 11 \rangle + \equiv$
VALUE | value { return VALUE_ZZ; }

320. UNITS. [LDF 2012.10.12.]

Log

[LDF 2012.10.12.] Added this rule.

$\langle \text{Rules } 11 \rangle + \equiv$
UNITS | units { return UNITS_ZZ; }

321. TIME_SET. [LDF 2012.10.12.]

Log

[LDF 2012.10.12.] Added this rule.

$\langle \text{Rules } 11 \rangle +\equiv$

`TIME_SET|time_set{|} return TIME_SET_ZZ; {}}`

322. SERVER. [LDF 2012.07.30.]

Log

[LDF 2012.07.30.] Added this rule.

$\langle \text{Rules } 11 \rangle +\equiv$

`SERVER|server{|} return SERVER_ZZ; {}}`

323. LOCAL_FILENAME. [LDF 2012.11.21.]

Log

[LDF 2012.11.21.] Added this rule.

$\langle \text{Rules } 11 \rangle +\equiv$

`LOCAL_FILENAME|local_filename{|} return LOCAL_FILENAME_ZZ; {}}`

324. REMOTE_FILENAME. [LDF 2012.11.21.]

Log

[LDF 2012.11.21.] Added this rule.

$\langle \text{Rules } 11 \rangle +\equiv$

`REMOTE_FILENAME|remote_filename{|} return REMOTE_FILENAME_ZZ; {}}`

325. PID. [LDF 2012.11.21.]

Log

[LDF 2012.11.21.] Added this rule.

$\langle \text{Rules } 11 \rangle +\equiv$

`\+?(PID|pid){|} return PID_ZZ; {}}`

326. PIDS. [LDF 2012.11.21.]

Log

[LDF 2012.11.21.] Added this rule.

$\langle \text{Rules } 11 \rangle +\equiv$

`PIDS|pids{|} return PIDS_ZZ; {}}`

327. REFERENCE. [LDF 2012.11.21.]

Log

[LDF 2012.11.21.] Added this rule.

$\langle \text{Rules } 11 \rangle + \equiv$

`REFERENCE|reference_{ return REFERENCE_ZZ; }`

328. CD. [LDF 2012.11.23.]

Log

[LDF 2012.11.23.] Added this rule.

$\langle \text{Rules } 11 \rangle + \equiv$

`CD|cd_{ return CD_ZZ; }`

329. MKDIR. [LDF 2012.11.23.]

Log

[LDF 2012.11.23.] Added this rule.

$\langle \text{Rules } 11 \rangle + \equiv$

`MKDIR|mkdir_{ return MKDIR_ZZ; }`

330. RM. [LDF 2012.11.23.]

Log

[LDF 2012.11.23.] Added this rule.

$\langle \text{Rules } 11 \rangle + \equiv$

`RM|rm_{ return RM_ZZ; }`

331. FINISHED. [LDF 2012.07.30.]

Log

[LDF 2012.07.30.] Added this rule.

$\langle \text{Rules } 11 \rangle + \equiv$

`FINISHED|finished_{ return FINISHED_ZZ; }`

332. END_SERVER. [LDF 2013.04.03.]**Log**

[LDF 2013.04.03.] Added this rule.

```
<Rules 11> +≡
[END_SERVER|end_server]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return END_SERVER_ZZ; []
}
```

333. SLEEP. [LDF 2013.04.19.]**Log**

[LDF 2013.04.19.] Added this rule.

```
<Rules 11> +≡
[SLEEP|sleep]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return SLEEP_ZZ; []
}
```

334. SIGNAL. [LDF 2013.05.02.]**Log**

[LDF 2013.05.02.] Added this rule.

```
<Rules 11> +≡
[SIGNAL|signal]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return SIGNAL_ZZ; []
}
```

335. SHOW. [LDF 2013.05.03.]**Log**

[LDF 2013.05.03.] Added this rule.

```
<Rules 11> +≡
[SHOW|show]{
#endif DEBUG_COMPILE
if (SCANNER_DEBUG) {
lock_cerr_mutex();
cerr << "In `zzlex': " << yytext << endl;
unlock_cerr_mutex();
} /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
return SHOW_ZZ; []
```

336. CERTIFICATE. [LDF 2013.05.03.]**Log**

[LDF 2013.05.03.] Added this rule.

```
<Rules 11> +≡
[CERTIFICATE|certificate]{
#endif DEBUG_COMPILE
if (SCANNER_DEBUG) {
lock_cerr_mutex();
cerr << "In `zzlex': " << yytext << endl;
unlock_cerr_mutex();
} /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
return CERTIFICATE_ZZ; []
```

337. CERTIFICATES. [LDF 2013.05.03.]**Log**

[LDF 2013.05.03.] Added this rule.

```
<Rules 11> +≡
[CERTIFICATES|certificates]{
#endif DEBUG_COMPILE
if (SCANNER_DEBUG) {
lock_cerr_mutex();
cerr << "In `zzlex': " << yytext << endl;
unlock_cerr_mutex();
} /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
return CERTIFICATES_ZZ; []
```

338. FOR. [LDF 2013.05.03.]**Log**

[LDF 2013.05.03.] Added this rule.

```
(Rules 11) +≡
[FOR|for_]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return FOR_ZZ; []
}
```

339. ALL. [LDF 2013.05.03.]**Log**

[LDF 2013.05.03.] Added this rule.

```
(Rules 11) +≡
[ALL|all_]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return ALL_ZZ; []
}
```

340. DISTINGUISHED_NAME. [LDF 2013.05.14.]**Log**

[LDF 2013.05.14.] Added this rule.

```
(Rules 11) +≡
[DISTINGUISHED_NAME|distinguished_name_]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return DISTINGUISHED_NAME_ZZ; []
}
```

341. AUTHENTICATION. [LDF 2013.05.10.]**Log**

[LDF 2013.05.10.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
[AUTHENTICATION|authentication_{
#define DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return AUTHENTICATION_ZZ; []
}

```

342. ERROR. [LDF 2013.05.10.]**Log**

[LDF 2013.05.10.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
[ERROR|error_{
#define DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return ERROR_ZZ; []
}

```

343. CERTIFICATE_ID. [LDF 2013.05.15.]**Log**

[LDF 2013.05.15.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
[CERTIFICATE_ID|certificate_id_{
#define DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return CERTIFICATE_ID_ZZ; []
}

```

344. USER_ID. [LDF 2013.05.15.]**Log**

[LDF 2013.05.15.] Added this rule.

```
<Rules 11> +≡
[USER_ID|user_id]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return USER_ID_ZZ; []
}
```

345. ISSUER_CERT_ID. [LDF 2013.05.15.]**Log**

[LDF 2013.05.15.] Added this rule.

```
<Rules 11> +≡
[ISSUER_CERT_ID|issuer_cert_id]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return ISSUER_CERT_ID_ZZ; []
}
```

346. USER_NAME. [LDF 2013.05.15.]**Log**

[LDF 2013.05.15.] Added this rule.

```
<Rules 11> +≡
[USER_NAME|user_name]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return USER_NAME_ZZ; []
}
```

347. ORGANIZATION. [LDF 2013.05.15.]**Log**

[LDF 2013.05.15.] Added this rule.

```
<Rules 11> +≡
[ORGANIZATION|organization_{
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return ORGANIZATION_ZZ; []
}
```

348. ORGANIZATIONAL_UNIT_NAME. [LDF 2013.05.15.]**Log**

[LDF 2013.05.15.] Added this rule.

```
<Rules 11> +≡
[ORGANIZATIONAL_UNIT_NAME|organizational_unit_name_{
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return ORGANIZATIONAL_UNIT_NAME_ZZ; []
}
```

349. COMMON_NAME. [LDF 2013.05.15.]**Log**

[LDF 2013.05.15.] Added this rule.

```
<Rules 11> +≡
[COMMON_NAME|common_name_{
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return COMMON_NAME_ZZ; []
}
```

350. COUNTRY_NAME. [LDF 2013.05.15.]**Log**

[LDF 2013.05.15.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
[ COUNTRY_NAME | country_name ]
#if DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return COUNTRY_NAME_ZZ; []

```

351. LOCALITY_NAME. [LDF 2013.05.15.]**Log**

[LDF 2013.05.15.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
[ LOCALITY_NAME | locality_name ]
#if DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return LOCALITY_NAME_ZZ; []

```

352. STATE_OR_PROVINCE_NAME. [LDF 2013.05.15.]**Log**

[LDF 2013.05.15.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
[ STATE_OR_PROVINCE_NAME | state_or_province_name ]
#if DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return STATE_OR_PROVINCE_NAME_ZZ; []

```

353. SERIAL_NUMBER. [LDF 2013.05.15.]**Log**

[LDF 2013.05.15.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
[SERIAL_NUMBER|serial_number]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return SERIAL_NUMBER_ZZ; []

```

354. VALIDITY_NOT_BEFORE. [LDF 2013.05.15.]**Log**

[LDF 2013.05.15.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
[VALIDITY_NOT_BEFORE|validity_not_before]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return VALIDITY_NOT_BEFORE_ZZ; []

```

355. VALIDITY_NOT_AFTER. [LDF 2013.05.15.]**Log**

[LDF 2013.05.15.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
[VALIDITY_NOT_AFTER|validity_not_after]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return VALIDITY_NOT_AFTER_ZZ; []

```

356. IS_CA. [LDF 2013.05.15.]**Log**

[LDF 2013.05.15.] Added this rule.

```
<Rules 11> +≡
[IS_CA|is_ca_{
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return IS_CA_ZZ; []
}
```

357. IS_PROXY. [LDF 2013.05.15.]**Log**

[LDF 2013.05.15.] Added this rule.

```
<Rules 11> +≡
[IS_PROXY|is_proxy_{
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return IS_PROXY_ZZ; []
}
```

358. PRIVILEGES. [LDF 2013.05.22.]**Log**

[LDF 2013.05.22.] Added this rule.

```
<Rules 11> +≡
[PRIVILEGES|privileges_{
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return PRIVILEGES_ZZ; []
}
```

359. IRODS_CURRENT_DIR. [LDF 2013.05.22.]

Log

[LDF 2013.05.22.] Added this rule.

```
<Rules 11> +≡
[IRODS_CURRENT_DIR|irods_current_dir]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return IRODS_CURRENT_DIR_ZZ; []
```

360. IRODS_HOMEDIR. [LDF 2013.05.22.]

Log

[LDF 2013.05.22.] Added this rule.

```
<Rules 11> +≡
[IRODS_HOMEDIR|irods_homedir]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return IRODS_HOMEDIR_ZZ; []
```

361. IRODS_ZONE. [LDF 2013.05.22.]

Log

[LDF 2013.05.22.] Added this rule.

```
<Rules 11> +≡
[IRODS_ZONE|irods_zone]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return IRODS_ZONE_ZZ; []
```

362. IRODS_DEFAULT_RESOURCE. [LDF 2013.05.22.]**Log**

[LDF 2013.05.22.] Added this rule.

```
<Rules 11> +≡
[IRODS_DEFAULT_RESOURCE|irods_default_resource]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return IRODS_DEFAULT_RESOURCE_ZZ; []
```

363. IRODS_OBJECT. [LDF 2013.08.07.]**Log**

[LDF 2013.08.07.] Added this rule.

```
<Rules 11> +≡
[IRODS_OBJECT|irods_object]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return IRODS_OBJECT_ZZ; []
```

364. DEFAULT_HANDLE_PREFIX. [LDF 2013.05.22.]**Log**

[LDF 2013.05.22.] Added this rule.

```
<Rules 11> +≡
[DEFAULT_HANDLE_PREFIX|default_handle_prefix]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return DEFAULT_HANDLE_PREFIX_ZZ; []
```

365. DEFAULT_HANDLE_PREFIX_ID. [LDF 2013.05.22.]**Log**

[LDF 2013.05.22.] Added this rule.

```
( Rules 11 ) +≡
[DEFAULT_HANDLE_PREFIX_ID|default_handle_prefix_id]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return DEFAULT_HANDLE_PREFIX_ID_ZZ; []
}
```

366. DEFAULT_INSTITUTE_NAME. [LDF 2013.05.22.]**Log**

[LDF 2013.05.22.] Added this rule.

```
( Rules 11 ) +≡
[DEFAULT_INSTITUTE_NAME|default_institute_name]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return DEFAULT_INSTITUTE_NAME_ZZ; []
}
```

367. DEFAULT_INSTITUTE_ID. [LDF 2013.05.22.]**Log**

[LDF 2013.05.22.] Added this rule.

```
( Rules 11 ) +≡
[DEFAULT_INSTITUTE_ID|default_institute_id]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return DEFAULT_INSTITUTE_ID_ZZ; []
}
```

368. GET_USER. [LDF 2013.05.16.]**Log**

[LDF 2013.05.16.] Added this rule.

```
<Rules 11> +≡
[GET_USER|get_user]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In `zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return GET_USER_ZZ; []
}
```

369. GET_USER_INFO. [LDF 2013.05.17.]**Log**

[LDF 2013.05.17.] Added this rule.

```
<Rules 11> +≡
[GET_USER_INFO|get_user_info]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In `zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    yyval->int_value = GET_USER_INFO_ZZ;
    return GET_USER_INFO_ZZ; []
}
```

370. HEXL_ENCODED_STRING. [LDF 2013.04.26.]

The hexadecimal encoded string is delimited by two ASCII RS, i.e., “Record Separator” characters “\x1e” = 30. [LDF 2013.04.28.]

Log

[LDF 2013.04.26.] Added this rule.

[LDF 2013.04.28.] Now allocating a **char** buffer on the heap and setting *yyval-pointer_value* to point to it. This makes it possible to have hexadecimal encoded strings longer than 1024 characters. Also removing delimiter characters from beginning and end of the encoded string.

```
<Rules 11> +≡
[ \x1e[0-9a-fA-F]*\x1e ]
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': Rule for hexadecimal encoded string:" << endl <<
            " 'yytext' == " << yytext << endl << " 'strlen(yytext)' == " << strlen(yytext) << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    string s = yytext;
    s.erase(0, 1);
    s.erase(s.length() - 1);
    char *buffer = new char[s.length() + 1];
    memset(buffer, 0, s.length() + 1);
    memcpy(buffer, s.c_str(), s.length());
    yyval.pointer_value = static_cast<void*>(buffer);
    return HEXL_ENCODED_STRING_ZZ; []

```

371. WHOAMI. [LDF 2013.05.19.]

Log

[LDF 2013.05.19.] Added this rule.

```
<Rules 11> +≡
[ WHOAMI | whoami ]
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    yyval.int_value = WHOAMI_ZZ;
    return WHOAMI_ZZ; []

```

372. PROCESS. [LDF 2013.05.23.]**Log**

[LDF 2013.05.23.] Added this rule.

```
<Rules 11> +≡
[PROCESS|process_]{}
#if DEBUG_COMPILE
if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'zzlex': " << yytext << endl;
    unlock_cerr_mutex();
} /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
return PROCESS_ZZ; []
```

373. PENDING. [LDF 2013.05.23.]**Log**

[LDF 2013.05.23.] Added this rule.

```
<Rules 11> +≡
[PENDING|pending_]{}
#if DEBUG_COMPILE
if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'zzlex': " << yytext << endl;
    unlock_cerr_mutex();
} /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
return PENDING_ZZ; []
```

374. OPERATIONS. [LDF 2013.05.23.]**Log**

[LDF 2013.05.23.] Added this rule.

```
<Rules 11> +≡
[OPERATIONS|operations_]{}
#if DEBUG_COMPILE
if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'zzlex': " << yytext << endl;
    unlock_cerr_mutex();
} /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
return OPERATIONS_ZZ; []
```

375. GROUP. [LDF 2013.06.04.]

Log

[LDF 2013.06.04.] Added this rule.

```
<Rules 11> +≡
[GROUP|group_]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return GROUP_ZZ; []

```

376. GROUPS. [LDF 2013.06.04.]

Log

[LDF 2013.06.04.] Added this rule.

```
<Rules 11> +≡
[GROUPS|groups_]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return GROUPS_ZZ; []

```

377. GROUP_ID. [LDF 2013.06.05.]

Log

[LDF 2013.06.05.] Added this rule.

```
<Rules 11> +≡
[GROUP_ID|group_id_]{}
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return GROUP_ID_ZZ; []

```

378. GROUP_NAME. [LDF 2013.06.05.]**Log**

[LDF 2013.06.05.] Added this rule.

```
<Rules 11> +≡
[GROUP_NAME|group_name_{

#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return GROUP_NAME_ZZ; []
}
```

379. CREATOR_ID. [LDF 2013.06.05.]**Log**

[LDF 2013.06.05.] Added this rule.

```
<Rules 11> +≡
[CREATOR_ID|creator_id_{

#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return CREATOR_ID_ZZ; []
}
```

380. CREATOR_USERNAME. [LDF 2013.06.05.]**Log**

[LDF 2013.06.05.] Added this rule.

```
<Rules 11> +≡
[CREATOR_USERNAME|creator_username_{

#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return CREATOR_USERNAME_ZZ; []
}
```

381. DELETE. [LDF 2013.07.04.]

Log

[LDF 2013.07.04.] Added this rule.

```
<Rules 11> +≡
  [DELETE|delete]{
#ifndef DEBUG_COMPILE
  if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'yylex': " << yytext << endl;
    unlock_cerr_mutex();
  } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
  return DELETE_ZZ; []
```

382. UNDELETE. [LDF 2013.07.04.]

Log

[LDF 2013.07.04.] Added this rule.

```
<Rules 11> +≡
  [UNDELETE|undelete]{
#ifndef DEBUG_COMPILE
  if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'yylex': " << yytext << endl;
    unlock_cerr_mutex();
  } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
  return UNDELETE_ZZ; []
```

383. DELETION. [LDF 2013.07.04.]

Log

[LDF 2013.07.04.] Added this rule.

```
<Rules 11> +≡
  [DELETION|deletion]{
#ifndef DEBUG_COMPILE
  if (SCANNER_DEBUG) {
    lock_cerr_mutex();
    cerr << "In 'yylex': " << yytext << endl;
    unlock_cerr_mutex();
  } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
  return DELETION_ZZ; []
```

384. MARK. [LDF 2013.07.04.]**Log**

[LDF 2013.07.04.] Added this rule.

```
<Rules 11> +≡
[MARK|mark_{
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In `yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return MARK_ZZ; []
}
```

385. UNMARK. [LDF 2013.07.04.]**Log**

[LDF 2013.07.04.] Added this rule.

```
<Rules 11> +≡
[UNMARK|unmark_{
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In `yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return UNMARK_ZZ; []
}
```

386. DATABASE. [LDF 2013.08.12.]**Log**

[LDF 2013.08.12.] Added this rule.

```
<Rules 11> +≡
[DATABASE|database_{
#ifndef DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In `yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return DATABASE_ZZ; []
}
```

387. DELAY. [LDF 2013.08.08.]**Log**

[LDF 2013.08.08.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
[DELAY|delay]{
#endif DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return DELAY_ZZ; []
}

```

388. OPTION. [LDF 2013.08.08.]**Log**

[LDF 2013.08.08.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
[OPTION|option]{
#endif DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'yylex': " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return OPTION_ZZ; []
}

```

389. Handle value specification. [LDF 2013.08.30.]**Log**

[LDF 2013.08.30.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
[[[:digit:]]{5}]/[[:alnum:]\-_]+[:alnum:]\-_+{
    bool save_scanner_debug = SCANNER_DEBUG;
    SCANNER_DEBUG = true; /* false */
#endif DEBUG_COMPILE
    if (SCANNER_DEBUG) {
        lock_cerr_mutex();
        cerr << "In 'zzlex': handle_value_specification: " << yytext << endl;
        unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
    strcpy(yyval->string_value, yytext);
    SCANNER_DEBUG = save_scanner_debug;
    return HANDLE_VALUE_SPECIFICATION_ZZ; []
}

```

390. DUMMY_STATEMENT. [LDF 2013.04.05.]

Log

[LDF 2013.04.05.] Added this rule.

```
<Rules 11> +≡
  DUMMY_STATEMENT | dummy_statement ↴{  
#if DEBUG_COMPILE
    if (SCANNER_DEBUG) {
      lock_cerr_mutex();
      cerr ≪ "In 'zzlex': " ≪ yytext ≪ endl;
      unlock_cerr_mutex();
    } /* if (SCANNER_DEBUG) */
#endif /* DEBUG_COMPILE */
  return DUMMY_STATEMENT_ZZ; }
```

391. End-of-file (EOF).

```
<Rules 11> +≡
  <<EOF>> ↴{ return END_ZZ; }
```

392. Delimited string: "....". [LDF 2012.07.30.]**Log**

[LDF 2012.09.27.] Now allowing all characters except for " in a delimited string.

[LDF 2013.04.05.] Added code for converting ASCII STX and ETX (“Start of Text” and “End of Text”, respectively) to double quotation marks. Please note that two different characters are both converted to the same character, thus losing some information, albeit a slight amount. It would be possible to convert them to different characters or strings, e.g., ‘ and ’ or “ and ”.

```
<Rules 11> +≡
["[^"]*"[{"  

    if (SCANNER_DEBUG) {  

        lock_cerr_mutex();  

        cerr << "Delimited_string." << yytext << endl;  

        unlock_cerr_mutex();  

    }  

    string temp_str = yytext;  

    string::size_type s = temp_str.size();  

    temp_str.erase(0, 1);  

    temp_str.erase(s - 2, s - 1);  

    size_t pos = string::npos;  

    do {  

        pos = temp_str.find('\002'); /* ASCII STX (“Start of Text”) character */  

        if (pos != string::npos) temp_str.replace(pos, 1, '\042'); /* Double quotation mark */  

    } while (pos != string::npos);  

    pos = string::npos;  

    do {  

        pos = temp_str.find('\003'); /* ASCII ETX (“End of Text”) character */  

        if (pos != string::npos) temp_str.replace(pos, 1, '\042'); /* Double quotation mark */  

    } while (pos != string::npos);  

    do {  

        pos = temp_str.find("\n");  

        if (pos != string::npos) temp_str.replace(pos, 2, "\n");  

    } while (pos != string::npos);  

    strcpy(yyval->string_value, temp_str.c_str());  

    return STRING_ZZ; []
```

393. Undelimited string. [LDF 2012.07.30.]

```
<Rules 11> +≡
[[alnum:] ./; :~@() +? =~$] [[alnum:] ./; :~- =~+@() ? =~$]* [{"strcpy(yyval->string_value, yytext);  

if (SCANNER_DEBUG) {  

    lock_cerr_mutex();  

    cerr << "In 'zzlex': Undelimited_string == " << yytext << endl <<  

    "Returning 'STRING_ZZ'." << endl;  

    unlock_cerr_mutex();  

}  

return STRING_ZZ; []}]
```

394. Other characters. [LDF 2012.07.30.]

```
< Rules 11 > +≡
  [lock_cerr_mutex();]
  cerr ≪ "In 'zzlex': Other character: " ≪ yytext ≪ " . Continuing." ≪ endl;
  unlock_cerr_mutex(); /* Do nothing. */
  [j]
```

395. Additional functions.

```
< zzwrap definition 395 > ≡
int zzwrap(yyscan_t parameter)
{
    return 1;
}
```

This code is used in section 397.

396.

```
< zzerror definition 396 > ≡
int zzerror(void *v, char const *s)
{
    return 0;
}
```

This code is used in section 397.

397. Putting client scanner together.

```
%{  
    < Include files 3 >  
    using namespace std;  
    using namespace gwrdifpk; [%] < Options 5 >  
    %% [%] < Local variables for zzlex 261 >  
    [%] < Execute on entry to zzlex 262 >  
    < Rules 11 >  
    %% < zzwrap definition 395 >  
    < zzerror definition 396 >
```

398. Client Parser. [LDF 2012.07.30.]

399. Include files.

```
<Include files 3> +≡
#include <stdlib.h>
#include <stdio.h>
#include <stdarg.h>
#include <limits.h>
#include <signal.h>
#include <algorithm>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <iterator>
#include <time.h>
#include <math.h>
#include <sstream>
#include <map>
#include <vector>
#include <deque>
#include <stack>
#include <set>
#include <pthread.h>
#include <expat.h>
#if HAVE_CONFIG_H
#include "config.h"
#endif
#include <gcrypt.h> /* for gcry-control */
#include <gnutls/gnutls.h>
#include <gnutls/x509.h>
#include <mysql.h>
#undef NAME_LEN
#undef LOCAL_HOST
#include "rspercds.h++"
#include "glblcnst.h++"
#include "glblvrbl.h++"
#include "utilfncs.h++"
#include "grouptp.h++"
#include "hndlvltp.h++"
#include "irdsavtp.h++"
#include "pidfncs.h++"
#include "tanfncs.h++"
#include "prsrlnt.h++"
#include "scnrlnt.h++"
#include "rspnstp.h++"
#include "irdsobtp.h++"
#include "hndltype.h++"
#include "dcmtddtp.h++"
#include "x509cert.h++"
#include "dstngnmt.h++"
#include "scprpmtp.h++"
#include "usrinftp.h++"
```

400. Declarations of additional functions.

```
(Declarations of additional functions 125) +≡
int zzlex(YYSTYPE *lvalp, yyscan_t parameter);
int zzwrap(void);
int zzerror(void *v, char const *s);
```

401. Options.

```
(Options 5) +≡
%verbose
%pure-parser
%parse-param {yyscan_t parameter}
%name-prefix="zz"
%lex-param {yyscan_t parameter}
%debug
```

402. union declaration.

Log

[LDF 2012.09.28.] Added **unsigned int uint_value**.
 [LDF 2013.04.28.] Added **void *pointer_value**.

```
(union declaration 127) +≡
```

```
%union {
    int int_value;
    unsigned int uint_value;
    unsigned long int ulint_value;
    float float_value;
    char string_value[1024];
    void *pointer_value; }
```

403. Token and type declarations. [LDF 2012.07.30.]

Log

- [LDF 2012.07.30.] Added token declarations for SERVER_ZZ and FINISHED_ZZ.
[LDF 2012.09.10.] Added token declarations for FAILED_ZZ and SUCCEEDED_ZZ.
[LDF 2012.09.26.] Added token declaration for PUT_ZZ.
[LDF 2012.09.27.] Added token declaration for RECEIVE_ZZ.
[LDF 2012.10.10.] Added token declaration for PIDS_ZZ.
[LDF 2012.10.12.] Added token declarations for UNSIGNED_INTEGER_ZZ, ATTRIBUTE_ZZ, VALUE_ZZ, UNITS_ZZ and TIME_SET_ZZ.
[LDF 2012.10.12.] Added the token declarations for HANDLE_ZZ, HANDLES_ZZ, HANDLE_VALUE_ZZ and HANDLE_VALUES_ZZ.
[LDF 2012.10.16.] Added token declarations for IDX_ZZ, TYPE_ZZ, DATA_ZZ, DATA_LENGTH_ZZ, TTL_TYPE_ZZ, TTL_ZZ, TIMESTAMP_ZZ, REFS_ZZ, REFS_LENGTH_ZZ, ADMIN_READ_ZZ, ADMIN_WRITE_ZZ, PUB_READ_ZZ, PUB_WRITE_ZZ, HANDLE_ID_ZZ, HANDLE_VALUE_ID_ZZ, CREATED_ZZ and LAST_MODIFIED_ZZ.
[LDF 2012.10.16.] Added token declaration for UNSIGNED_LONG_INTEGER_ZZ.
[LDF 2012.11.19.] Added token declarations for CLIENT_SIDE_FILENAME_ZZ and OVERWRITE_ZZ.
[LDF 2012.11.21.] Added token declarations for LOCAL_FILENAME_ZZ, REMOTE_FILENAME_ZZ, FLAG_ZZ and REFERENCE_ZZ.
[LDF 2012.12.13.] Added token declaration for SENDING_ZZ.
[LDF 2012.12.14.] Added token declaration for ADD_ZZ.
[LDF 2012.12.31.] Added token declaration for OUTPUT_ZZ.
[LDF 2013.02.28.] Added token declaration for CREATED_BY_USER_ZZ
[LDF 2013.03.07.] Added token declaration for STORE_ZZ.
[LDF 2013.04.03.] Added token declaration for END_SERVER_ZZ.
[LDF 2013.04.05.] Added token declaration for DUMMY_STATEMENT_ZZ.
[LDF 2013.04.19.] Added token declaration for SLEEP_ZZ.
[LDF 2013.04.26.] Added token declarations for HEXL_ENCODED_STRING_ZZ and DATA_HEXL_ENCODED_ZZ.
[LDF 2013.04.28.] Changed the type of HEXL_ENCODED_STRING_ZZ from *char_value* to *pointer_value*. Added token declaration for REFS_HEXL_ENCODED_ZZ.
[LDF 2013.05.02.] Added token declaration for SIGNAL_ZZ.
[LDF 2013.05.03.] Added token declarations for CERTIFICATE_ZZ, CERTIFICATES_ZZ, FOR_ZZ and ALL_ZZ.
[LDF 2013.05.10.] Added token declarations for AUTHENTICATION_ZZ and ERROR_ZZ.
[LDF 2013.05.15.] Added the following token declarations: CERTIFICATE_ID_ZZ
- USER_ID_ZZ
ISSUER_CERT_ID_ZZ
USER_NAME_ZZ
ORGANIZATION_ZZ
ORGANIZATIONAL_UNIT_NAME_ZZ
COMMON_NAME_ZZ
COUNTRY_NAME_ZZ
LOCALITY_NAME_ZZ
STATE_OR_PROVINCE_NAME_ZZ
SERIAL_NUMBER_ZZ
VALIDITY_NOT_BEFORE_ZZ
VALIDITY_NOT_AFTER_ZZ
IS_CA_ZZ
IS_PROXY_ZZ
- [LDF 2013.05.16.] Added token declaration for GET_USER_ZZ.
[LDF 2013.05.17.] Added token declaration for GET_USER_INFO_ZZ.
[LDF 2013.05.19.] Added token declaration for WHOAMI_ZZ.

[LDF 2013.05.22.] Added token declarations for PRIVILEGES_ZZ, IRODS_HOME_DIR_ZZ, IRODS_CURRENT_DIR_ZZ, IRODS_ZONE_ZZ, IRODS_DEFAULT_RESOURCE_ZZ, DEFAULT_HANDLE_PREFIX_ID_ZZ, DEFAULT_HANDLE_PREFIX_ZZ, DEFAULT_INSTITUTE_ID_ZZ and DEFAULT_INSTITUTE_NAME_ZZ.

[LDF 2013.05.23.] Added token declarations for PROCESS_ZZ, PENDING_ZZ and OPERATIONS_ZZ.

[LDF 2013.06.04.] Added token declarations for GROUP_ZZ and GROUPS_ZZ.

[LDF 2013.06.05.] Added token declarations for GROUP_ID_ZZ, GROUP_NAME_ZZ, CREATOR_ID_ZZ, and CREATOR_USERNAME_ZZ.

[LDF 2013.07.04.] Added token declarations for UNDELETE_ZZ, DELETION_ZZ, MARK_ZZ and UNMARK_ZZ.

[LDF 2013.07.15.] Added token declaration for DELETED_ZZ.

[LDF 2013.08.07.] Added token declaration for IRODS_OBJECT_ZZ.

[LDF 2013.08.08.] Added token declarations for DELAY_ZZ and OPTION_ZZ.

[LDF 2013.08.12.] Added token declaration for DATABASE_ZZ.

[LDF 2013.08.22.] Added token declaration for MARKED_FOR_DELETION_ZZ.

[LDF 2013.08.26.] Added token declarations for DELETE_FROM_DATABASE_TIMESTAMP_ZZ and IRODS_OBJECT_ID_ZZ.

[LDF 2013.08.30.] Added token declaration for HANDLE_VALUE_SPECIFICATION_ZZ.

< Token and type declarations 128 > +≡

```
% token < int_value > INTEGER_ZZ
% token < uint_value > UNSIGNED_INTEGER_ZZ
% token < uint_value > UNSIGNED_LONG_INTEGER_ZZ
% token < int_value > END_ZZ
% token < int_value > END_SERVER_ZZ
% token < int_value > CERTIFICATE_ZZ
% token < int_value > CERTIFICATES_ZZ
% token < int_value > CERTIFICATE_ID_ZZ
% token < int_value > USER_ID_ZZ
% token < int_value > ISSUER_CERT_ID_ZZ
% token < int_value > USER_NAME_ZZ
% token < int_value > GET_USER_ZZ
% token < int_value > GET_USER_INFO_ZZ
% token < int_value > WHOAMI_ZZ
% token < int_value > PROCESS_ZZ
% token < int_value > PENDING_ZZ
% token < int_value > OPERATIONS_ZZ
% token < int_value > ORGANIZATION_ZZ
% token < int_value > ORGANIZATIONAL_UNIT_NAME_ZZ
% token < int_value > COMMON_NAME_ZZ
% token < int_value > COUNTRY_NAME_ZZ
% token < int_value > LOCALITY_NAME_ZZ
% token < int_value > STATE_OR_PROVINCE_NAME_ZZ
% token < int_value > SERIAL_NUMBER_ZZ
% token < int_value > VALIDITY_NOT_BEFORE_ZZ
% token < int_value > VALIDITY_NOT_AFTER_ZZ
% token < int_value > IS_CA_ZZ
% token < int_value > IS_PROXY_ZZ
% token < int_value > FOR_ZZ
% token < int_value > ALL_ZZ
% token < int_value > SIGNAL_ZZ
% token < string_value > HANDLE_VALUE_SPECIFICATION_ZZ
```

404.

```
< Token and type declarations 128 > +≡
% token < int_value > SLEEP_ZZ
% token < int_value > DUMMY_STATEMENT_ZZ
% token < int_value > ADD_ZZ
% token < int_value > STORE_ZZ
% token < int_value > FILE_ZZ
% token < string_value > CLIENT_SIDE_FILENAME_ZZ
% token < int_value > OVERWRITE_ZZ
% token < int_value > DIRECTORY_ZZ
% token < int_value > LOCAL_ZZ
% token < int_value > REMOTE_ZZ
% token < string_value > LOCAL_FILENAME_ZZ
% token < string_value > REMOTE_FILENAME_ZZ
% token < string_value > FLAG_ZZ
% token < int_value > REFERENCE_ZZ
% token < int_value > MKDIR_ZZ
% token < int_value > RM_ZZ
% token < int_value > CREATE_ZZ
% token < int_value > DELETE_ZZ
% token < int_value > UNDELETE_ZZ
% token < int_value > DELETION_ZZ
% token < int_value > MARK_ZZ
% token < int_value > UNMARK_ZZ
% token < int_value > COPY_ZZ
% token < int_value > MOVE_ZZ
% token < int_value > RENAME_ZZ
% token < int_value > CHMOD_ZZ
% token < int_value > CHOWN_ZZ
% token < int_value > SHOW_ZZ
% token < int_value > USER_ZZ
% token < int_value > INFO_ZZ
% token < int_value > PWD_ZZ
% token < int_value > CD_ZZ
```

405.

```
< Token and type declarations 128 > +≡
% token < int_value > METADATA_ZZ
% token < int_value > OUTPUT_ZZ
% token < int_value > URI_ZZ
% token < int_value > PASSWORD_ZZ
% token < int_value > SU_ZZ
% token < int_value > SET_ZZ
% token < int_value > GET_ZZ
% token < int_value > PUT_ZZ
% token < int_value > SESSION_ZZ
% token < int_value > LS_ZZ
% token < int_value > REPOSITORY_ZZ
% token < int_value > RESPONSE_ZZ
% token < int_value > CONNECT_ZZ
% token < int_value > DISCONNECT_ZZ
% token < int_value > CONNECTION_ZZ
% token < int_value > RESET_ZZ
% token < int_value > WRITE_ZZ
% token < int_value > READ_ZZ
% token < int_value > TO_ZZ
% token < int_value > DISTINGUISHED_NAME_ZZ
% token < string_value > STRING_ZZ
% token < pointer_value > HEXL_ENCODED_STRING_ZZ
% token < int_value > SEND_ZZ
% token < int_value > SENDING_ZZ
% token < int_value > TAN_ZZ
% token < int_value > LIST_ZZ
% token < int_value > HANDLE_ZZ
% token < int_value > HANDLES_ZZ
% token < int_value > HANDLE_VALUE_ZZ
% token < int_value > HANDLE_VALUES_ZZ
```

406.

```
< Token and type declarations 128 > +≡
% token < int_value > IDX_ZZ
% token < int_value > TYPE_ZZ
% token < int_value > DATA_ZZ
% token < int_value > DATA_LENGTH_ZZ
% token < int_value > DATA_HEXL_ENCODED_ZZ
% token < int_value > TTL_TYPE_ZZ
% token < int_value > TTL_ZZ
% token < int_value > TIMESTAMP_ZZ
% token < int_value > REFS_ZZ
% token < int_value > REFS_LENGTH_ZZ
% token < int_value > REFS_HEXL_ENCODED_ZZ
% token < int_value > ADMIN_READ_ZZ
% token < int_value > ADMIN_WRITE_ZZ
% token < int_value > PUB_READ_ZZ
% token < int_value > PUB_WRITE_ZZ
% token < int_value > HANDLE_ID_ZZ
% token < int_value > HANDLE_VALUE_ID_ZZ
% token < int_value > IRODS_OBJECT_ID_ZZ
% token < int_value > CREATED_ZZ
% token < int_value > LAST_MODIFIED_ZZ
% token < int_value > CREATED_BY_USER_ZZ
% token < int_value > DELETE_FROM_DATABASE_TIMESTAMP_ZZ
% token < int_value > DELETED_ZZ
% token < int_value > MARKED_FOR_DELETION_ZZ
% token < int_value > PID_ZZ
% token < int_value > PIDS_ZZ
% token < int_value > RECEIVE_ZZ
% token < int_value > FAILED_ZZ
% token < int_value > SUCCEEDED_ZZ
% token < int_value > ATTRIBUTE_ZZ
% token < int_value > VALUE_ZZ
% token < int_value > UNITS_ZZ
% token < int_value > TIME_SET_ZZ
% token < int_value > PRIVILEGES_ZZ
% token < int_value > IRODS_HOMEDIR_ZZ
% token < int_value > IRODS_CURRENT_DIR_ZZ
% token < int_value > IRODS_ZONE_ZZ
% token < int_value > IRODS_DEFAULT_RESOURCE_ZZ
% token < int_value > IRODS_OBJECT_ZZ
% token < int_value > DEFAULT_HANDLE_PREFIX_ID_ZZ
% token < int_value > DEFAULT_HANDLE_PREFIX_ZZ
% token < int_value > DEFAULT_INSTITUTE_ID_ZZ
% token < int_value > DEFAULT_INSTITUTE_NAME_ZZ
% token < int_value > AUTHENTICATION_ZZ
% token < int_value > ERROR_ZZ
% token < int_value > GROUP_ZZ
% token < int_value > GROUPS_ZZ
% token < int_value > GROUP_ID_ZZ
% token < int_value > GROUP_NAME_ZZ
% token < int_value > CREATOR_ID_ZZ
```

```
% token < int_value > CREATOR_USERNAME_ZZ
% token < int_value > DATABASE_ZZ
% token < int_value > DELAY_ZZ
% token < int_value > OPTION_ZZ
% token < int_value > SERVER_ZZ
% token < int_value > FINISHED_ZZ
```

407. Rules.

$\langle \text{Rules } 11 \rangle +\equiv$

408. Program. [LDF 2012.07.30.]

$\langle \text{Rules } 11 \rangle +\equiv$

```
program: statement_list END_ZZ
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
#if DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread_" << param->thread_ctr << "] " program:statement_list END_ZZ" << endl <<
            "Exiting 'zzparse' with return value 0." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    return 0;
}
;
```

409. $\langle \text{statement list} \rangle$. [LDF 2012.07.30.]**410.** $\langle \text{statement list} \rangle \longrightarrow \text{EMPTY}$. This rule ensures that an empty file won't cause an error. [LDF 2012.07.30.] ■

$\langle \text{Rules } 11 \rangle +\equiv$

```
statement_list: /* Empty */
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
#if DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread_" << param->thread_ctr << "] " statement_list:/* Empty */" << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
}
;
```

411. ⟨statement list⟩ → ⟨statement⟩. [LDF 2012.07.30.]

⟨Rules 11⟩ +≡

```

statement_list : statement_list statement
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
#endif DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " statement_list : statement
            << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
}
;
```

412. ⟨statement⟩ → PWD_ZZ RESPONSE_ZZ STRING_ZZ. [LDF 2012.09.06.]

Log

[LDF 2012.09.06.] Added RESPONSE_ZZ.

⟨Rules 11⟩ +≡

```

statement : PWD_ZZ RESPONSE_ZZ STRING_ZZ
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
#endif DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " 'statement : PWD_ZZ RESPONSE_ZZ STRING_ZZ' . "
            << endl << 'STRING_ZZ' == " << [$3] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    string temp_str = [$3];
    lock_cout_mutex();
    lock_cerr_mutex();
    cout << "pwd-->" << endl << [$3] << endl << endl;
    unlock_cerr_mutex();
    unlock_cout_mutex();
}
;
```

413. ⟨statement⟩ → PWD_ZZ FAILED_ZZ INTEGER_ZZ. [LDF 2012.09.10.]

Log

[LDF 2012.09.10.] Added this rule.

```

⟨Rules 11⟩ +≡
statement : PWD_ZZ FAILED_ZZ INTEGER_ZZ
{
    Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
        *>(zzget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] 'statement:PWD_ZZ FAILED_ZZ INTEGER_ZZ'." <<
            endl << "'INTEGER_ZZ'" <= < $3 << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    lock_cout_mutex();
    lock_cerr_mutex();
    cout << "ERROR! In 'zzparse', rule 'statement:PWD_ZZ FAILED_ZZ INTEGER_ZZ':"
        endl << "Server-side error: 'pwd' failed."
        "Exiting function unsuccessfully with return value 1." << endl;
    unlock_cerr_mutex();
    unlock_cout_mutex();
    return 1;
}
;

```

414. ⟨statement⟩ → LS_ZZ RESPONSE_ZZ INTEGER_ZZ STRING_ZZ. [LDF 2012.09.06.]

Log

[LDF 2012.09.06.] Added this rule.

```

⟨Rules 11⟩ +≡
statement: LS_ZZ RESPONSE_ZZ INTEGER_ZZ STRING_ZZ
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] In 'zzparse', rule" << endl <<
            "'statement: LS_ZZ RESPONSE_ZZ INTEGER_ZZ STRING_ZZ'." << endl <<
            "'INTEGER_ZZ'" <= " " << $3 << endl << "'STRING_ZZ'" <= " " << $4 << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
/* !! TODO: LDF 2012.11.28. Add code for evaluating result code. */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] In 'zzparse', rule" << endl <<
            "'statement: LS_ZZ RESPONSE_ZZ INTEGER_ZZ STRING_ZZ';" << endl <<
            "'STRING_ZZ'" <= " " << $4 << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    lock_cout_mutex();
    lock_cerr_mutex();
    cout << "ls-->" << endl << $4 << endl;
    unlock_cerr_mutex();
    unlock_cout_mutex();
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

415. ⟨statement⟩ → PUT_ZZ RESPONSE_ZZ STRING_ZZ INTEGER_ZZ STRING_ZZ. [LDF 2012.09.27.]

Log

[LDF 2012.09.27.] Added this rule.

```

⟨Rules 11⟩ +≡
statement : PUT_ZZ RESPONSE_ZZ STRING_ZZ INTEGER_ZZ STRING_ZZ
{
    Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
        *>(zzget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "statement:PUT_ZZ_RESPONSE_ZZ_STRING_ZZ_INTEGER_ZZ_STRING_ZZ" <<
            endl << "'STRING_ZZ'" << $3 << endl << "'INTEGER_ZZ'" << $4 << endl <<
            "'STRING_ZZ'" << $5 << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    lock_cout_mutex();
    lock_cerr_mutex();
    cout << "put-->" << endl << "Filename:" << $3 << endl << "Exit_status:" << $4 <<
        endl << "Response:" << $5 << endl;
    if ($4 == 2) cout << "Prefix not found." << endl;
    else if ($4 == 3) cout << "Prefix disabled." << endl;
    cout << endl;
    unlock_cerr_mutex();
    unlock_cout_mutex();
}
;
```

416. ⟨statement⟩ → GET_ZZ FILE_ZZ RESPONSE_ZZ STRING_ZZ INTEGER_ZZ STRING_ZZ ⟨client-side filename optional⟩. [LDF 2012.10.02.]

The response code INTEGER_ZZ will be 0 in the case that the “get” operation succeeded and the server is sending the file, an odd number > 0 in error cases, and an even number > 0 if just a message is being sent. [LDF 2013.04.05.]

Log

[LDF 2012.10.02.] Added this rule.

[LDF 2012.10.12.] Added FILE_ZZ. Without it, this rule was confusing, because I’ve added more “GET” rules.

⟨Rules 11⟩ +≡

```

statement : GET_ZZ FILE_ZZ RESPONSE_ZZ STRING_ZZ INTEGER_ZZ STRING_ZZ [client_side_filename_optional]
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "statement: GET_ZZ FILE_ZZ RESPONSE_ZZ STRING_ZZ INTEGER_ZZ STRING_ZZ " <<
            "client_side_filename_optional_overwrite_optional'." <<
            endl << "'STRING_ZZ'" << (server-side_filename) << $4 <<
            endl << "'INTEGER_ZZ'" << (response) << $5 << endl <<
            "'STRING_ZZ'" << (server-side_command_output) << $6 <<
            endl << "'client_side_filename_optional'" << $7 << endl <<
            "'overwrite_optional'" << $8 << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    lock_cout_mutex();
    lock_cerr_mutex();
    cout << "get--" << endl;
    if ($5 > 0 & $5 % 2 == 0) /* Response is a message */
    {
        cout << "Local_filename:" << $4 << endl << "Response_code:" << $5 << endl <<
            "Response:" << endl;
    } /* if */
    else {
        cout << "Remote_filename:" << $4 << endl << "Local_filename:" <<;
        if (strlen($7) > 0) cout << $7;
        else cout << $4;
        cout << endl << "Exit_status:" << $5 << endl << "Overwrite:" <<;
        if ($8) cout << "True";
        else cout << "False";
        cout << endl;
    } /* else */
    cout << endl;
    unlock_cerr_mutex();

```

```

unlock_cout_mutex();
if ([\$5] ≡ 0) /* Exit status of server-side "get" command was zero (success) */
{
    string local_filename;
    string temp_filename;
    status = param->receive_file([\$4], [\$7], [\$8], &local_filename, &temp_filename);
    if (status ≠ 0) {
        lock_cerr_mutex();
        cerr ≪ "[Thread" ≪ param->thread_ctr ≪ "] ERROR! In 'zzparse', rule" ≪ endl ≪
            "statement: GET_ZZ_FILE_ZZ_RESPONSE_ZZ_STRING_ZZ_INTEGER_ZZ_STRING_ZZ'" ≪
            endl ≪ "'Scan_Parse_Parameter_Type::receive_file' failed, returning" ≪
            status ≪ "." ≪ endl ≪ "Failed to receive file" ≪ [\$4] ≪ "'";
        if (strlen([\$7]) > 0) cerr ≪ " or store it in" ≪ [\$7] ≪ "";
        cerr ≪ "." ≪ endl;
        if (status ≡ 1) cerr ≪ "'access' error for local filename" ≪ endl;
        else if (status ≡ 2)
            cerr ≪ "Local file already exists and 'overwrite' == false" ≪ endl;
        else if (status ≡ 3) cerr ≪ "'mkdir' error." ≪ endl;
        else if (status ≡ 4) cerr ≪ "'access' error for local path." ≪ endl;
        else if (status ≡ 5) cerr ≪ "'ofstream::open' error for local file." ≪ endl;
        else cerr ≪ "Unknown error." ≪ endl;
        if (temp_filename.empty())
            cerr ≪ "Failed to store file contents in temporary file." ≪ endl;
        else cerr ≪ "Stored file contents in file:" ≪ temp_filename ≪ "" ≪ endl;
        cerr ≪ "Will try to continue." ≪ endl;
        unlock_cerr_mutex();
        ++param->errors_occurred;
    } /* if (status ≠ 0) */
    else /* status ≡ 0 */
    {
        lock_cout_mutex();
        lock_cerr_mutex();
        cout ≪ "Received remote file" ≪ [\$4] ≪ "." ≪ endl ≪ "Stored in local file" ≪ "
            local_filename ≪ '.' ≪ endl ≪ endl;
        unlock_cerr_mutex();
        unlock_cout_mutex();
    } /* else (status ≡ 0) */
} /* if ([\$5] ≡ 0) */

```

417.

Log

[LDF 2013.04.05.] Added this section.

```

⟨ Rules 11 ⟩ +≡
else
    if ([\$5] ≡ 2) { /* Do nothing. [LDF 2013.04.05.] */
    }

```

418.

```

⟨Rules 11⟩ +≡
else      /* Exit status of server-side “get” command was non-zero (failure) */
{
    lock_cout_mutex();
    lock_cerr_mutex();
    cout << "Server-side‘get’_command_failed,_returning" << $5 << "."
        << endl <<
    "Not_calling‘receive_file’._" << "Will_try_to_continue." << endl;
    unlock_cerr_mutex();
    unlock_cout_mutex();
}      /* else */
param~PARSER_DEBUG = save_PARSER_DEBUG; } ;

```

419. ⟨client-side filename optional⟩. [LDF 2012.11.19.]**Log**

[LDF 2012.11.19.] Added this type declaration.

```

⟨Token and type declarations 128⟩ +≡
[%type<string_value>_client_side_filename_optional]

```

420. ⟨client-side filename optional⟩ → Empty.**Log**

[LDF 2012.11.19.] Added this rule.

[LDF 2013.04.04.] BUG FIX: Now setting the value of this rule, i.e., \$\$\$, to the empty string.

```

⟨Rules 11⟩ +≡
[client_side_filename_optional:_/*_Empty_*/]
{
    Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
        *>(zzget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param~PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread_" << param~thread_ctr << "]_"
            << "In_‘zzparse’,_rule_‘client_side_filename_optional:_/*_Empty_*/' ."
            << endl;
        unlock_cerr_mutex();
    }      /* if (param~PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    strcpy($$, "");
}
;
```

421. ⟨client-side filename optional⟩ → CLIENT_SIDE_FILENAME_ZZ STRING_ZZ. [LDF 2012.11.19.]

Log

[LDF 2012.11.19.] Added this rule.

```

⟨Rules 11⟩ +≡
  client_side_filename_optional: CLIENT_SIDE_FILENAME_ZZ STRING_ZZ
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "In 'zzparse', rule 'client_side_filename_optional:' <<
            "CLIENT_SIDE_FILENAME_ZZ STRING_ZZ'." << endl << "'STRING_ZZ' == " << [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    strcpy([$$, [$2]);
}
;
```

422. ⟨overwrite optional⟩. [LDF 2012.11.19.]

Log

[LDF 2012.11.19.] Added this type declaration.

```

⟨Token and type declarations 128⟩ +≡
  [%type<int_value>overwrite_optional]
```

423. ⟨overwrite optional⟩ → Empty.

Log

[LDF 2012.11.19.] Added this rule.

```

⟨Rules 11⟩ +≡
  overwrite_optional:/*Empty*/
{
    Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
        *>(zzget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
            << "In 'zzparse', rule 'overwrite_optional:/*Empty*/'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    $$ = 0;
}
;
```

424. ⟨overwrite optional⟩ → OVERWRITE_ZZ. [LDF 2012.11.19.]

Log

[LDF 2012.11.19.] Added this rule.

```

⟨Rules 11⟩ +≡
  overwrite_optional:OVERWRITE_ZZ
{
    Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
        *>(zzget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
            << "In 'zzparse', rule 'overwrite_optional:'"
            << "OVERWRITE_ZZ'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    $$ = 1;
}
;
```

425. ⟨statement⟩ → GET_ZZ PIDS_ZZ RESPONSE_ZZ STRING_ZZ INTEGER_ZZ INTEGER_ZZ STRING_ZZ.
[LDF 2012.10.10.]

Log

[LDF 2012.10.10.] Added this rule.
[LDF 2012.12.31.] Added UNSIGNED_INTEGER_ZZ.

```

⟨Rules 11⟩ +≡
[statement: GET_ZZ PIDS_ZZ RESPONSE_ZZ STRING_ZZ INTEGER_ZZ INTEGER_ZZ UNSIGNED_INTEGER_ZZ] STRING_ZZ
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] In 'zzparse', rule" <<
            endl << "statement: GET_ZZ PIDS_ZZ RESPONSE_ZZ STRING_ZZ" <<
            "INTEGER_ZZ INTEGER_ZZ UNSIGNED_INTEGER_ZZ STRING_ZZ;" << endl <<
            "'STRING_ZZ'(4)==uuuuuuuuuuuu" << $4 << endl << "'INTEGER_ZZ'(5)==uuuuuuuuuuuu" <<
            $5 << endl << "'INTEGER_ZZ'(6)==uuuuuuuuuu" << $6 << endl <<
            "'UNSIGNED_INTEGER_ZZ'(7)==u" << $7 << endl << "'STRING_ZZ'(8)==uuuuuuuuuu" <<
            $8 << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    lock_cout_mutex();
    lock_cerr_mutex();
    cout << "get_pids-->" << endl << "Filename:uuuuuuuu" << $4 << endl << "Exit_status:uuuu" <<
        $5 << endl << "Number_of_PIDS:uu" << $6 << endl << "Response:uuuuuuuu" << $7 << endl;
    if ($6 > 0) cout << "PIDs:uuuuuuuuuuuu" << $8 << endl;
    cout << endl;
    unlock_cerr_mutex();
    unlock_cout_mutex();
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

426. ⟨statement⟩ → GET_ZZ METADATA_ZZ RESPONSE_ZZ STRING_ZZ INTEGER_ZZ INTEGER_ZZ UNSIGNED_INTEGER_ZZ [avu list]. [LDF 2012.10.12.]

Log

[LDF 2012.10.12.] Added this rule.
 [LDF 2012.12.31.] Added UNSIGNED_INTEGER_ZZ.

```

⟨Rules 11⟩ +≡
[statement : GET_ZZ METADATA_ZZ RESPONSE_ZZ STRING_ZZ INTEGER_ZZ INTEGER_ZZ] UNSIGNED_INTEGER_ZZ avu list
{
  Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type*>(zzget_extra(parameter));
  bool save_PARSER_DEBUG = param->PARSER_DEBUG;
  param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
  if (param->PARSER_DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread]" << param->thread_ctr << "]" <<
      "statement: GET_ZZ METADATA_ZZ RESPONSE_ZZ STRING_ZZ" <<
      "INTEGER_ZZ INTEGER_ZZ UNSIGNED_INTEGER_ZZ avu list'" << endl <<
      "STRING_ZZ" <= "[$4]" << endl << "INTEGER_ZZ" <= "[$5]" << endl <<
      "INTEGER_ZZ" <= "[$6]" << endl << "UNSIGNED_INTEGER_ZZ" <= "[$7]" << endl;
    unlock_cerr_mutex();
  } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
  lock_cout_mutex();
  lock_cerr_mutex();
  cout << "get_metadata--" << endl << "Filename: " << "[$4]" << endl <<
    "Exit_status: " << "[$5]" << endl << "Number_of_AVUs: " << "[$6]" << endl << endl;
  unlock_cerr_mutex();
  unlock_cout_mutex();
  stringstream temp_strm;
  if (param & param->irods_object) {
    lock_cout_mutex();
    lock_cerr_mutex();
    if (param->irods_object->avu_vector.size() == 0)
      cout << "No user-defined metadata (AVUs) to display" << endl << endl;
    else {
      param->irods_object->path = "[$4]";
      param->irods_object->vector.push_back(*param->irods_object);
      delete param->irods_object;
      param->irods_object = 0;
      param->irods_object->vector.back().show("", &temp_strm);
      cout << temp_strm.str();
    }
    unlock_cerr_mutex();
    unlock_cout_mutex();
  }
  else {
    lock_cerr_mutex();
  }
}

```

```
cerr << "[Thread]" << param->thread_ctr << "]" <<
    "In 'zzparse', rule 'statement:GET_ZZ_METADATA_ZZ_RESPONSE_ZZ_STRING_ZZ' <<
    "INTEGER_ZZ_INTEGER_ZZ_UNSIGNED_INTEGER_ZZ_avu_list':" << endl <<
    "'param' is NULL or 'param->irods_object' is NULL." << endl <<
    "Can't display user-defined metadata (AVUs) for iRODS object" << "''" << [$4] <<
    "'.'" << endl << "Will try to continue." << endl;
    unlock_cerr_mutex();
}
param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

427. ⟨statement⟩ → GET_ZZ METADATA_ZZ RESPONSE_ZZ STRING_ZZ INTEGER_ZZ INTEGER_ZZ UNSIGNED_INTEGER_ZZ STRING_ZZ. [LDF 2012.10.12.]

Log

[LDF 2012.10.12.] Added this rule. Currently, it is only matched if there was a server-side error, but this may change in the future.

[LDF 2012.12.31.] Added UNSIGNED_INTEGER_ZZ.

```

⟨ Rules 11 ⟩ +≡
statement : GET_ZZ METADATA_ZZ RESPONSE_ZZ STRING_ZZ INTEGER_ZZ INTEGER_ZZ UNSIGNED_INTEGER_ZZ STRING_ZZ
{
    Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "statement: GET_ZZ METADATA_ZZ RESPONSE_ZZ STRING_ZZ " <<
            "INTEGER_ZZ INTEGER_ZZ UNSIGNED_INTEGER_ZZ STRING_ZZ'." <<
            endl << "'STRING_ZZ'" <(4)<(filename or PID)==uuu" << $4 <<
            endl << "'INTEGER_ZZ'" <(5)<(error code)==uuuuuuu" << $5 <<
            endl << "'INTEGER_ZZ'" <(6)<(number of AVUS)==uuu" << $6 <<
            endl << "'UNSIGNED_INTEGER_ZZ'" <(7)<(options)==u" << $7 << endl <<
            "'STRING_ZZ'" <(8)<(response)==uuuuuuuuu" << $8 << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    lock_cout_mutex();
    lock_cerr_mutex();
    cout << "get_metadata--" << endl << "Filename or PID: " << $4 << endl <<
        "Exit status: " << $5 << endl << "Number of AVUs: " << $6 << endl <<
        "Options: " << hex << $7 << "(hexadecimal)" << dec << endl <<
        "Response: " << $8 << endl << endl;
    unlock_cerr_mutex();
    unlock_cout_mutex();
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

428. ⟨avu list⟩ and ⟨avu⟩. [LDF 2012.10.12.]

Log

[LDF 2012.10.12.] Added these type declarations.

```

⟨ Token and type declarations 128 ⟩ +≡
% type < int_value > avu_list % type < int_value > avu

```

429. $\langle \text{avu list} \rangle \rightarrow \text{Empty}$. [LDF 2012.10.12.]

Log

[LDF 2012.10.12.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
[avu_list:/*Empty*/
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
#endif DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
        << "In 'zzparse', rule 'avu_list:/*Empty*/'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    if (param->irods_object) {
        param->irods_object->clear();
    }
    else param->irods_object = new Irods_Object_Type;
    $$ = 0;
}
;
```

430. $\langle \text{avu list} \rangle \rightarrow \langle \text{avu list} \rangle \langle \text{avu} \rangle$. [LDF 2012.10.12.]

Log

[LDF 2012.10.12.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
[avu_list:avu_list_avu
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
#endif DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
        << "In 'zzparse', rule 'avu_list:avu_list_avu.'"
        << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
}
```

431. ⟨avu⟩ → ATTRIBUTE_ZZ STRING_ZZ VALUE_ZZ STRING_ZZ UNITS_ZZ STRING_ZZ TIME_SET_ZZ UNSIGNED_INTEGER_ZZ
[LDF 2012.10.12.]

Log

[LDF 2012.10.12.] Added this rule.

```

⟨Rules 11⟩ +≡
[avu:ATTRIBUTE_ZZ STRING_ZZ VALUE_ZZ STRING_ZZ UNITS_ZZ STRING_ZZ] [TIME_SET_ZZ] UNSIGNED_INTEGER_ZZ
{
    Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
        *>(zzget_extra(parameter));
#endif DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "In 'zzparse', rule " avu:ATTRIBUTE_ZZ STRING_ZZ VALUE_ZZ" << endl <<
            "STRING_ZZ UNITS_ZZ STRING_ZZ TIME_SET_ZZ UNSIGNED_INTEGER_ZZ'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    Irods_AVU_Type irods_avu($2, $4, $6, $8);
    if (param & param->irods_object) {
        param->irods_object->avu_vector.push_back(irods_avu);
    }
    else {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "ERROR! In 'zzparse', rule " avu:ATTRIBUTE_ZZ STRING_ZZ VALUE_ZZ" <<
            endl << "STRING_ZZ UNITS_ZZ STRING_ZZ TIME_SET_ZZ UNSIGNED_INTEGER_ZZ' :" <<
            endl << "'param'==0 or 'param->irods_object'==0." <<
            endl << "Can't push 'Irods_AVU_Type irods_avu' onto " <<
            "'param->irods_object.avu_vector'." << endl << "Will try to continue." << endl;
        unlock_cerr_mutex();
    }
}
;
```

432. ⟨statement⟩ → SERVER_ZZ FINISHED_ZZ. [LDF 2012.07.30.]

Log

[LDF 2012.07.30.] Added this rule.

```
(Rules 11) +≡
statement: SERVER_ZZ FINISHED_ZZ
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
#endif DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread]" << param->thread_ctr << "] " <<
            "In 'zzparse', rule" << statement: SERVER_ZZ FINISHED_ZZ . " << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->server_finished = true;
}
;
```

433. ⟨statement⟩ → GET_ZZ HANDLE_ZZ RESPONSE_ZZ INTEGER_ZZ ⟨handle item list⟩. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

[LDF 2013.04.26.] Now calling **Handle_Value_Type**::*write_to_database*. This writes *param->handle_value* to the client-side *gwirdcli.handles* database table.

```
(Rules 11) +≡
statement: GET_ZZ HANDLE_ZZ RESPONSE_ZZ INTEGER_ZZ handle_item_list {
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#endif DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread]" << param->thread_ctr << "] In 'zzparse', Rule" <<
            endl << "statement: GET_ZZ HANDLE_ZZ RESPONSE_ZZ INTEGER_ZZ " <<
            "handle_item_list ." << endl << "INTEGER_ZZ" == " " << [$4] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
```

434. See **Scan_Parse_Parameter_Type** :: *get_handle* in **scprpmtp.web** for error codes. [LDF 2012.11.14.]

```

⟨ Rules 11 ⟩ +≡
if ([\$4] ≠ 0) {
    lock_cout_mutex();
    lock_cerr_mutex();
    cout ≪ "get_handle-->" ≪ endl ≪ "Server-side_error:" ≪ endl ≪ "Response_code:uuu" ≪
        [\$4] ≪ endl ≪ "handle:uuuuuuuuuu" ≪ param→handle_value.handle ≪ endl;
    if ([\$4] ≡ 1 ∨ [\$4] ≡ 2 ∨ [\$4] ≡ 4) {
        cout ≪ "Database_error";
        ++param→errors_occurred;
    }
    else if ([\$4] ≡ 3) {
        cout ≪ "Database_query_returned_0_rows";
        ++param→warnings_occurred;
    }
    else if ([\$4] ≡ 5) {
        cout ≪ "'Handle_Type::set' failed";
        ++param→errors_occurred;
    }
    else if ([\$4] > 5) {
        cout ≪ "Other_error.";
        ++param→errors_occurred;
    }
    cout ≪ endl;
    unlock_cerr_mutex();
    unlock_cout_mutex();
} /* if */

```

435.

Log

[LDF 2013.02.08.] Added code for converting *param-handle_value.timestamp*, *param-handle_value.created* and *param-handle_value.last_modified* to timestamps.

[LDF 2013.02.28.] Added code to account for *param-handle_value.created_by_user_id* and *param-handle_value.created_by*.

```

⟨Rules 11⟩ +≡
else /* Success */
{ int field_width = strlen("delete_from_database_timestamp:uu");
lock_cerr_mutex();
lock_cout_mutex();
cout << "get_handle-->" << endl << setw(field_width) << left << "Response_code:" <<
    $4 << endl << setw(field_width) << left << "filename:" << param-handle_value.filename <<
    endl << setw(field_width) << left << "handle:" << param-handle_value.handle << endl <<
    setw(field_width) << left << "idx:" << param-handle_value.idx << endl << setw(field_width) << left <<
    "type:" << param-handle_value.type << endl << setw(field_width) << left << "data_length:" <<
    param-handle_value.data_length << endl;
if (param-handle_value.data & param-handle_value.data_length > 0) {
    cout << setw(field_width) << left << "data:" <<
    fwrite(param-handle_value.data, 1, param-handle_value.data_length, stdout);
}
else if (param-handle_value.data == 0 & param-handle_value.data_length > 0)
    cout << setw(field_width) << left << "data:" << "(binary)";
else cout << setw(field_width) << left << "data:" << "NULL";
cout << endl << setw(field_width) << left << "ttl_type:" << param-handle_value.ttl_type << endl <<
    setw(field_width) << left << "ttl:" << param-handle_value.ttl << endl << setw(field_width) << left <<
    "timestamp:" << param-handle_value.timestamp;
if (param-handle_value.timestamp != 0)
    cout << " " << "(" << convert_seconds(param-handle_value.timestamp, true) << ")";
cout << endl << setw(field_width) << left << "refs_length:" << param-handle_value.refs_length << endl;
if (param-handle_value.refs & param-handle_value.refs_length > 0) {
    cout << "refs:" <<
    fwrite(param-handle_value.refs, 1, param-handle_value.refs_length, stdout);
}
else if (param-handle_value.refs == 0 & param-handle_value.refs_length > 0)
    cout << setw(field_width) << left << "refs:" << "(binary)";
else cout << setw(field_width) << left << "refs:" << "NULL";
cout << endl << setw(field_width) << left << "admin_read:" << param-handle_value.admin_read <<
    endl << setw(field_width) << left << "admin_write:" << param-handle_value.admin_write <<
    endl << setw(field_width) << left << "pub_read:" << param-handle_value.pub_read << endl <<
    setw(field_width) << left << "pub_write:" << param-handle_value.pub_write << endl <<
    setw(field_width) << left << "handle_id:" << param-handle_value.handle_id << endl <<
    setw(field_width) << left << "handle_value_id:" << param-handle_value.handle_value_id << endl <<
    setw(field_width) << left << "irods_object_id:" << param-handle_value.irods_object_id << endl <<
    setw(field_width) << left << "created:" << param-handle_value.created;
if (param-handle_value.created != 0)
    cout << " " << "(" << convert_seconds(param-handle_value.created, true) << ")";
cout << endl << setw(field_width) << left << "last_modified:" << param-handle_value.last_modified;
if (param-handle_value.last_modified != 0)
    cout << " " << "(" << convert_seconds(param-handle_value.last_modified, true) << ")";

```

```

cout << endl << setw(field_width) << left << "delete_from_database_timestamp:" <<
    param->handle_value.delete_from_database_timestamp;
if (param->handle_value.delete_from_database_timestamp != 0)
    cout << " " << "(" << convert_seconds(param->handle_value.delete_from_database_timestamp,
    true) << ")";
cout << endl << setw(field_width) << left << "created_by_user:" <<
    param->handle_value.created_by_user_id << " " << param->handle_value.created_by_user_name << endl <<
    setw(field_width) << left << "marked_for_deletion:" << param->handle_value.marked_for_deletion <<
    endl << endl;
unlock_cout_mutex();
unlock_cerr_mutex();
#endif DEBUG_COMPILE
if (param->PARSER_DEBUG) {
    cerr << "param->handle_value.data==" << param->handle_value.data << endl <<
        "param->handle_value.data_length==" << param->handle_value.data_length << endl;
    cerr << "param->handle_value==" << endl;
    param->handle_value.show();
} /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */

```

436.

```

⟨Rules 11⟩ +≡
status = param->handle_value.write_to_database(param->mysql_ptr, "gwirdcli", true);
if (status != 0) {
    lock_cerr_mutex();
    cerr << "[Thread" << param->thread_ctr << "] ERROR! In 'zzparse', Rule" << endl <<
        "'statement: GET_ZZ_HANDLE_ZZ_RESPONSE_ZZ_INTEGER_ZZ'" << "handle_item_list':" <<
        endl << "'Handle_Value_Type::write_to_database' failed, returning" << status <<
        "." << endl << "Failed to write handle value to 'gwirdcli.handles' table." << endl <<
        "Will try to continue." << endl;
    unlock_cerr_mutex();
    ++param->errors_occurred;
} /* if (status != 0) */
#endif DEBUG_COMPILE
else
if (param->PARSER_DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << param->thread_ctr << "] In 'zzparse', Rule" << endl <<
        "'statement: GET_ZZ_HANDLE_ZZ_RESPONSE_ZZ_INTEGER_ZZ'" << "handle_item_list':" <<
        endl << "'Handle_Value_Type::write_to_database' succeeded, returning 0." << endl;
    unlock_cerr_mutex();
} /* else if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
} /* else Success */
param->handle_value.clear(); /* !! TODO: LDF 2013.04.26. I may want to copy the value of
    param->handle_value to somewhere before clearing it. */
param->PARSER_DEBUG = save_Parser_DEBUG; } ;

```

437. ⟨handle item list⟩ and ⟨handle item⟩. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added these type declarations.

⟨ Token and type declarations 128 ⟩ +≡

% type < int_value > handle_item_list % type < int_value > handle_item

438. ⟨handle item list⟩ → Empty. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

⟨ Rules 11 ⟩ +≡

```

[handle_item_list : /*Empty*/
{
    Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
        *>(zzget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
        << "In 'zzparse', rule 'handle_item_list : /*Empty*/' ."
        << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->handle_value.clear();
    $$ = 0;
}
;
```

439. ⟨handle item list⟩ → ⟨handle item list⟩ ⟨handle item⟩. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

```

⟨Rules 11⟩ +≡
[handle_item_list: handle_item_list handle_item]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
            << "In 'zzparse', rule 'handle_item_list: handle_item_list handle_item.'"
            << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
}
;
```

440. ⟨handle item⟩ → FILE_ZZ STRING_ZZ. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

```

⟨Rules 11⟩ +≡
[handle_item: FILE_ZZ STRING_ZZ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
            << "In 'zzparse', rule 'handle_item: FILE_ZZ STRING_ZZ.'"
            << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->handle_value.filename = [$2];
}
;
```

441. <handle item> —> HANDLE_ZZ STRING_ZZ. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

```

⟨Rules 11⟩ +≡
[handle_item:HANDLE_ZZ_STRING_ZZ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
        << "In 'zzparse', rule 'handle_item:HANDLE_ZZ_STRING_ZZ'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->handle_value.handle = [$2];
}
;
```

442. <handle item> —> IDX_ZZ INTEGER_ZZ. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

```

⟨Rules 11⟩ +≡
[handle_item:IDX_ZZ_INTEGER_ZZ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
        << "In 'zzparse', rule 'handle_item:IDX_ZZ_INTEGER_ZZ'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->handle_value.idx = [$2];
}
;
```

443. $\langle \text{handle item} \rangle \longrightarrow \text{TYPE_ZZ STRING_ZZ}$. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

```
< Rules 11 > +≡
[handle_item: TYPE_ZZ_STRING_ZZ] { Scan_Parser_Parameter_Type
    *param = static_cast<Scan_Parser_Parameter_Type*>(zzget_extra(parameter)); #
    if DEBUG_COMPILE
        if (param->PARSER_DEBUG) {
            lock_cerr_mutex();
            cerr << "[Thread_" << param->thread_ctr << "] " <<
                "In_`zzparse', rule_`handle_item:TYPE_ZZ_STRING_ZZ'." << endl;
            unlock_cerr_mutex();
        } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->handle_value.type = [$2]; }
```

444. $\langle \text{handle item} \rangle \longrightarrow \text{DATA_ZZ STRING_ZZ}$. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

```
< Rules 11 > +≡
[handle_item: DATA_ZZ_STRING_ZZ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type*>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#endif /* DEBUG_COMPILE */
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread_" << param->thread_ctr << "] " <<
            "In_`zzparse', rule_`handle_item:DATA_ZZ_STRING_ZZ'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->handle_value.data = new char[strlen([$2]) + 1];
    memset(param->handle_value.data, 0, strlen([$2]) + 1);
    memcpy(param->handle_value.data, [$2], strlen([$2]));
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

445. ⟨handle item⟩ → DATA_LENGTH_ZZ INTEGER_ZZ. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

```

⟨Rules 11⟩ +≡
handle_item:DATA_LENGTH_ZZ INTEGER_ZZ { Scan_Parser_Parameter_Type
    *param = static_cast<Scan_Parser_Parameter_Type *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
    #
    if DEBUG_COMPILE
        if (param->PARSER_DEBUG) {
            lock_cerr_mutex();
            cerr << "[Thread" << param->thread_ctr << "] " <<
                "In 'zzparse', rule 'handle_item:DATA_LENGTH_ZZ INTEGER_ZZ'." << endl;
            unlock_cerr_mutex();
        } /* if (param->PARSER_DEBUG) */
    #endif /* DEBUG_COMPILE */
    param->handle_value.data_length = $2;
    param->PARSER_DEBUG = save_PARSER_DEBUG; } ;

```

446. ⟨handle item⟩ → DATA_HEXL_ENCODED_ZZ HEXL_ENCODED_STRING_ZZ. [LDF 2013.04.26.]

Log

[LDF 2013.04.26.] Added this rule.

[LDF 2013.04.28.] Rewrote this rule to account for the change in the type of HEXL_ENCODED_STRING_ZZ from `char[1024]string_value` to `void *pointer_value` (see union declaration above). This makes it possible to have hexadecimal encoded strings of length > 1024.

```

⟨Rules 11⟩ +≡
[handle_item:DATA_HEXL_ENCODED_ZZ,HEXL_ENCODED_STRING_ZZ]
{
    Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " << "In 'zzparse', rule 'handle_item:DATA_H\
EXL_ENCODED_ZZ,HEXL_ENCODED_STRING_ZZ'." << endl;
        cerr << "'HEXL_ENCODED_STRING_ZZ'" <= << static_cast<char *>([$2]) << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    string temp_str = static_cast<char *>([$2]);
    delete[] static_cast<char *>([$2]);
    [$2] = 0;
    string dest;
    unsigned int dest_length;
    status = hexl_decode(temp_str, dest, dest_length);
    if (status != 0) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " << "ERROR! In 'zzparse', " <<
            "rule 'handle_item:DATA_HEXL_ENCODED_ZZ,HEXL_ENCODED_STRING_ZZ': " <<
            endl << "'hexl_decode' failed, returning " << status << "." << endl <<
            "Failed to decode hexadecimal-encoded string: " << endl << temp_str << endl <<
            "Leaving data field of handle unset. Will try to continue. " << endl;
        unlock_cerr_mutex();
        ++param->errors_occurred;
    } /* if (status != 0) */
    else /* status == 0 */
    {
#ifndef DEBUG_COMPILE
        if (param->PARSER_DEBUG) {
            cerr << "[Thread" << param->thread_ctr << "] " << "In 'zzparse', " <<
                "rule 'handle_item:DATA_HEXL_ENCODED_ZZ,HEXL_ENCODED_STRING_ZZ': " <<
                endl << "'hexl_decode' succeeded, returning 0." << endl <<
                "Decoded hexadecimal-encoded string successfully." <<
                endl << "'dest_length'" <= << dest_length << endl <<

```

```

    "param->handle_value.data_length == " << param->handle_value.data_length << endl <<
    "'buffer'" == " << endl;
    fwrite(dest.c_str(), 1, dest_length, stderr);
    cerr << endl << "Setting 'data' field of handle to 'buffer'." << endl;
    unlock_cerr_mutex();
} /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */

if (param->handle_value.data_length > 0 & param->handle_value.data_length != dest_length) {
    cerr << "[Thread" << param->thread_ctr << "] " << "WARNING! In 'zzparse', "
    "rule 'handle_item: DATA_HEXL_ENCODED_ZZ_HEXL_ENCODED_STRING_ZZ': " << endl <<
    "'param->handle_value.data_length' == " << param->handle_value.data_length <<
    endl << "'dest_length' == " << dest_length << endl <<
    "'param->handle_value.data_length' is not 0 and not equal to 'dest_length'." <<
    endl << "This shouldn't happen. Resetting 'param->handle_value.data_length' to "
    "to 'dest_length' == " << dest_length << "." << endl << "Will try to continue." <<
    endl;
    param->handle_value.data_length = dest_length;
    ++param->warnings_occurred;
} /* if */
#endif DEBUG_COMPILE
else
if (param->PARSER_DEBUG & param->handle_value.data_length > 0 & param->handle_value.data_length == dest_length) {
    lock_cerr_mutex();
    cerr << "[Thread" << param->thread_ctr << "] " << "In 'zzparse', "
    "rule 'handle_item: DATA_HEXL_ENCODED_ZZ_HEXL_ENCODED_STRING_ZZ': " << endl <<
    "'param->handle_value.data_length' == 'dest_length' == " << dest_length <<
    endl << "This is as it should be." << endl;
    unlock_cerr_mutex();
} /* else if */
else if (param->PARSER_DEBUG & param->handle_value.data_length == 0) {
    lock_cerr_mutex();
    cerr << "[Thread" << param->thread_ctr << "] " << "In 'zzparse', "
    "rule 'handle_item: DATA_HEXL_ENCODED_ZZ_HEXL_ENCODED_STRING_ZZ': " << endl <<
    "'param->handle_value.data_length' == 0. It may be set later." << endl;
    unlock_cerr_mutex();
} /* else if */
#endif /* DEBUG_COMPILE */
param->handle_value.data = new char[dest_length + 1];
memset(param->handle_value.data, 0, dest_length + 1);
memcpy(param->handle_value.data, dest.c_str(), dest_length);
} /* else (status == 0) */
param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

447. <handle item> → TTL_TYPE_ZZ INTEGER_ZZ. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

```

⟨Rules 11⟩ +≡
[handle_item:TTL_TYPE_ZZ_INTEGER_ZZ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
    *>(zzget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
        << "In 'zzparse', rule 'handle_item:TTL_TYPE_ZZ_INTEGER_ZZ'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->handle_value.ttl_type = [$2];
}
;
```

448. <handle item> → TTL_ZZ INTEGER_ZZ. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

```

⟨Rules 11⟩ +≡
[handle_item:TTL_ZZ_INTEGER_ZZ]{ Scan_Parser_Parameter_Type
    *param = static_cast<Scan_Parser_Parameter_Type *>(zzget_extra(parameter)); #
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
        << "In 'zzparse', rule 'handle_item:TTL_ZZ_INTEGER_ZZ'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->handle_value.ttl = [$2]; }
```

449. <handle item> → TIMESTAMP_ZZ UNSIGNED_INTEGER_ZZ. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

```

⟨Rules 11⟩ +≡
[handle_item: TIMESTAMP_ZZ UNSIGNED_INTEGER_ZZ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
            << "In 'zzparse', rule 'handle_item: TIMESTAMP_ZZ UNSIGNED_INTEGER_ZZ'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->handle_value.timestamp = [$2];
}
;
```

450. <handle item> → REFS_ZZ STRING_ZZ. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

```

⟨Rules 11⟩ +≡
[handle_item: REFS_ZZ STRING_ZZ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
            << "In 'zzparse', rule 'handle_item: REFS_ZZ STRING_ZZ'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->handle_value.refs = new char[strlen([$2]) + 1];
    memset(param->handle_value.refs, 0, strlen([$2]) + 1);
    strcpy(param->handle_value.refs, [$2]);
}
;
```

451. ⟨handle item⟩ → REFS_LENGTH_ZZ INTEGER_ZZ. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

```
⟨Rules 11⟩ +≡
[handle_item:REFS_LENGTH_ZZ INTEGER_ZZ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "In 'zzparse', rule " << handle_item:REFS_LENGTH_ZZ INTEGER_ZZ . " << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->handle_value.refs_length = [$2];
}
;
```

452. ⟨handle item⟩ → REFS_HEXL_ENCODED_ZZ HEXL_ENCODED_STRING_ZZ. [LDF 2013.04.26.]

Log

[LDF 2013.04.28.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
[handle_item:_REFS_HEXL_ENCODED_ZZ:_HEXL_ENCODED_STRING_ZZ_]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
#endif DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread_" << param->thread_ctr << "] " << "In '_zzparse', rule '_handle_item:_REFS_H\
EXL_ENCODED_ZZ:_HEXL_ENCODED_STRING_ZZ'." << endl;
        cerr << "'HEXL_ENCODED_STRING_ZZ'" << static_cast<char *>([$2]) << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    string temp_str = static_cast<char *>([$2]);
    delete[] static_cast<char *>([$2]);
    string dest;
    unsigned int dest_length;
    status = hexl_decode(temp_str, dest, dest_length);
    if (status != 0) {
        lock_cerr_mutex();
        cerr << "[Thread_" << param->thread_ctr << "] " << "ERROR! In '_zzparse', "
            "rule '_handle_item:_REFS_HEXL_ENCODED_ZZ:_HEXL_ENCODED_STRING_ZZ':"
            " " << endl << "'hexl_decode'_failed, returning" << status << "."
            " " << endl << "'Failed_to_decode_hexadecimal-encoded_string:'"
            " " << endl << temp_str << endl <<
            "'Leaving_refs_field_of_handle_unset. Will_try_to_continue.'"
            " " << endl;
        unlock_cerr_mutex();
        ++param->errors_occurred;
    } /* if (status != 0) */
    else /* status == 0 */
    {
#endif DEBUG_COMPILE
        if (param->PARSER_DEBUG) {
            cerr << "[Thread_" << param->thread_ctr << "] " << "In '_zzparse', "
                "rule '_handle_item:_REFS_HEXL_ENCODED_ZZ:_HEXL_ENCODED_STRING_ZZ':"
                " " << endl << "'hexl_decode'_succeeded, returning 0."
                " " << endl << "'Decoded_hexadecimal-encoded_string_successfully.'"
                " " << endl << "'dest_length'" << dest_length << endl <<
                "'param->handle_value.refs_length'" << param->handle_value.refs_length << endl <<
                "'buffer'" << endl;
            fwrite(dest.c_str(), 1, dest_length, stderr);
            cerr << endl << "'Setting_refs_field_of_handle_to_buffer.'"
            " " << endl;
            unlock_cerr_mutex();
        } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
        if (param->handle_value.refs_length > 0 & param->handle_value.refs_length != dest_length) {

```

```

cerr << "[Thread]" << param->thread_ctr << "] " << "WARNING! In 'zzparse', "
    "rule 'handle_item:_REFS_HEXL_ENCODED_ZZ_HEXL_ENCODED_STRING_ZZ': " << endl <<
    "'param->handle_value.refs_length' == " << param->handle_value.refs_length <<
    endl << "'dest_length' == " << dest_length << endl <<
    "'param->handle_value.refs_length' is not 0 and not equal to 'dest_length'. " <<
    endl << "This shouldn't happen. Resetting 'param->handle_value.refs_length' to "
    "to 'dest_length' == " << dest_length << ". " << endl << "Will try to continue. " <<
    endl;
param->handle_value.refs_length = dest_length;
++param->warnings_occurred;
} /* if */
#endif DEBUG_COMPILE
else
    if (param->PARSER_DEBUG & param->handle_value.refs_length > 0 & param->handle_value.refs_length ==
        dest_length) {
        lock_cerr_mutex();
        cerr << "[Thread]" << param->thread_ctr << "] " << "In 'zzparse', "
            "rule 'handle_item:_REFS_HEXL_ENCODED_ZZ_HEXL_ENCODED_STRING_ZZ': " << endl <<
            "'param->handle_value.refs_length' == 'dest_length' == " << dest_length <<
            endl << "This is as it should be. " << endl;
        unlock_cerr_mutex();
    } /* else if */
else if (param->PARSER_DEBUG & param->handle_value.refs_length == 0) {
    lock_cerr_mutex();
    cerr << "[Thread]" << param->thread_ctr << "] " << "In 'zzparse', "
        "rule 'handle_item:_REFS_HEXL_ENCODED_ZZ_HEXL_ENCODED_STRING_ZZ': " << endl <<
        "'param->handle_value.refs_length' == 0. It may be set later. " << endl;
    unlock_cerr_mutex();
} /* else if */
#endif /* DEBUG_COMPILE */
param->handle_value.refs = new char[dest_length];
memset(param->handle_value.refs, 0, dest_length);
memcpy(param->handle_value.refs, dest.c_str(), dest.length);
} /* else (status == 0) */
}
;

```

453. <handle item> —> ADMIN_READ_ZZ INTEGER_ZZ. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

```
<Rules 11> +≡
[handle_item:ADMIN_READ_ZZ_INTEGER_ZZ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
            << "In 'zzparse', rule 'handle_item:ADMIN_READ_ZZ_INTEGER_ZZ'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->handle_value.admin_read = [$2];
}
;
```

454. <handle item> —> ADMIN_WRITE_ZZ INTEGER_ZZ. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

```
<Rules 11> +≡
[handle_item:ADMIN_WRITE_ZZ_INTEGER_ZZ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
            << "In 'zzparse', rule 'handle_item:ADMIN_WRITE_ZZ_INTEGER_ZZ'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->handle_value.admin_write = [$2];
}
;
```

455. <handle item> → PUB_READ_ZZ INTEGER_ZZ. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

```

⟨Rules 11⟩ +≡
[handle_item:PUB_READ_ZZ_INTEGER_ZZ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
    *>(zzget_extra(parameter));
#endif DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "In 'zzparse', rule 'handle_item:PUB_READ_ZZ_INTEGER_ZZ'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->handle_value.pub_read = [$2];
}
;
```

456. <handle item> → PUB_WRITE_ZZ INTEGER_ZZ. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

```

⟨Rules 11⟩ +≡
[handle_item:PUB_WRITE_ZZ_INTEGER_ZZ] { Scan_Parser_Parameter_Type
    *param = static_cast<Scan_Parser_Parameter_Type *>(zzget_extra(parameter)); #
    if DEBUG_COMPILE
        if (param->PARSER_DEBUG) {
            lock_cerr_mutex();
            cerr << "[Thread" << param->thread_ctr << "] " <<
                "In 'zzparse', rule 'handle_item:PUB_WRITE_ZZ_INTEGER_ZZ'." << endl;
            unlock_cerr_mutex();
        } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->handle_value.pub_write = [$2]; } ;
```

457. <handle item> → HANDLE_ID_ZZ UNSIGNED_LONG_INTEGER_ZZ. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

```

⟨Rules 11⟩ +≡
[handle_item:HANDLE_ID_ZZ_UNSIGNED_LONG_INTEGER_ZZ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
            << "In 'zzparse', rule 'handle_item:HANDLE_ID_ZZ_UNSIGNED_LONG_INTEGER_ZZ'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->handle_value.handle_id = [$2];
}
;
```

458. <handle item> → HANDLE_VALUE_ID_ZZ UNSIGNED_LONG_INTEGER_ZZ. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

```

⟨Rules 11⟩ +≡
[handle_item:HANDLE_VALUE_ID_ZZ_UNSIGNED_LONG_INTEGER_ZZ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
            << "In 'zzparse', rule 'handle_item:HANDLE\
_VALUE_ID_ZZ_UNSIGNED_LONG_INTEGER_ZZ'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->handle_value.handle_value_id = [$2];
}
;
```

459. <handle item> → IRODS_OBJECT_ID_ZZ UNSIGNED_LONG_INTEGER_ZZ. [LDF 2013.08.26.]

Log

[LDF 2013.08.26.] Added this rule.

```
<Rules 11> +≡
[handle_item:IRODS_OBJECT_ID_ZZUNSIGNED_LONG_INTEGER_ZZ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
    *>(zzget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " << "In 'zzparse', rule 'handle_item:IRODS_\
OBJECT_ID_ZZUNSIGNED_LONG_INTEGER_ZZ'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->handle_value.irods_object_id = [$2];
}
;
```

460. <handle item> → CREATED_ZZ UNSIGNED_INTEGER_ZZ. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

```
<Rules 11> +≡
[handle_item:CREATED_ZZUNSIGNED_INTEGER_ZZ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
    *>(zzget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
        "In 'zzparse', rule 'handle_item:CREATED_ZZUNSIGNED_INTEGER_ZZ'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->handle_value.created = [$2];
}
;
```

461. <handle item> → LAST_MODIFIED_ZZ UNSIGNED_INTEGER_ZZ. [LDF 2012.10.16.]

Log

[LDF 2012.10.16.] Added this rule.

```

⟨Rules 11⟩ +≡
[handle_item:LAST_MODIFIED_ZZ_UNSIGNED_INTEGER_ZZ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
    *>(zzget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "In 'zzparse', rule 'handle_item:LAST_MODIFIED_ZZ_UNSIGNED_INTEGER_ZZ'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->handle_value.last_modified = [$2];
}
;
```

462. <handle item> → DELETE_FROM_DATABASE_TIMESTAMP_ZZ UNSIGNED_INTEGER_ZZ. [LDF 2013.08.26.]

Log

[LDF 2013.08.26.] Added this rule.

```

⟨Rules 11⟩ +≡
[handle_item:DELETE_FROM_DATABASE_TIMESTAMP_ZZ_UNSIGNED_INTEGER_ZZ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
    *>(zzget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "In 'zzparse', rule 'handle_item:' <<
            "DELETE_FROM_DATABASE_TIMESTAMP_ZZ_UNSIGNED_INTEGER_ZZ'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->handle_value.delete_from_database_timestamp = [$2];
}
;
```

463. <handle item> → CREATED_BY_USER_ZZ UNSIGNED_INTEGER_ZZ STRING_ZZ. [LDF 2013.02.28.]

Log

[LDF 2013.02.28.] Added this rule.

```
(Rules 11) +≡
[handle_item: CREATED_BY_USER_ZZ_UNSIGNED_INTEGER_ZZ_STRING_ZZ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
            << "In 'zzparse', rule 'handle_item: CREATED_BY_USER_ZZ'"
            << "UNSIGNED_INTEGER_ZZ_STRING_ZZ'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->handle_value.created_by_user_id = [$2];
    param->handle_value.created_by_user_name = [$3];
}
;
```

464. <handle item> → MARKED_FOR_DELETION_ZZ INTEGER_ZZ. [LDF 2013.07.15.]

Log

[LDF 2013.07.15.] Added this rule.

[LDF 2013.08.22.] Changed DELETED_ZZ to MARKED_FOR_DELETION_ZZ.

```
(Rules 11) +≡
[handle_item: MARKED_FOR_DELETION_ZZ_INTEGER_ZZ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
            << "In 'zzparse', rule 'handle_item: MARKED_FOR_DELETION_ZZ_INTEGER_ZZ'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->handle_value.marked_for_deletion = static_cast<bool>([$2]);
}
;
```

465. ⟨statement⟩ → SEND_ZZ FILE_ZZ STRING_ZZ REFERENCE_ZZ INTEGER_ZZ. [LDF 2012.11.21.]

Log

[LDF 2012.11.21.] Added this rule.

```

⟨Rules 11⟩ +≡
statement: SEND_ZZ FILE_ZZ STRING_ZZ REFERENCE_ZZ INTEGER_ZZ
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
    *>(zzget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "In 'zzparse', rule 'statement:SEND_ZZ(FILE_ZZ|STRING_ZZ|"
            "REFERENCE_ZZ|INTEGER_ZZ)." << endl << "'STRING_ZZ'|filename" << "$3" <<
            endl << "'INTEGER_ZZ'|reference_for_server" << "$5" << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    Response_Type response;
    response.type = Response_Type::SEND_FILE_TYPE;
    response.local_filename = "$3";
    temp_strm.str("");
    temp_strm << "CLIENT|SENDING|FILE|" << "$3" << "\"|REFERENCE" << "$5";
    response.command = temp_strm.str();
    param->response_deque.push_back(response);
    temp_strm.str("");
#endif DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "In 'zzparse', rule 'statement:SEND_ZZ(FILE_ZZ|STRING_ZZ|"
            "REFERENCE_ZZ|INTEGER_ZZ):'" << endl;
        response.show();
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
}
;
```

466. <statement> → CD_ZZ RESPONSE_ZZ INTEGER_ZZ STRING_ZZ STRING_ZZ STRING_ZZ. [LDF 2012.11.23.] ■

Integer: Status code. 0 for success, non-zero for failure.

First string: Requested directory. May be a single directory or a path containing more than one directory.

Second string: Previous working directory. Will be complete path. Third string: Current working directory.

Will be complete path. Fourth string: If *icommands* ≡ true, the output of the *cd* command. !! PLEASE

NOTE: This is the only case that has been implemented as of this date. [LDF 2012.11.23.]

Log

[LDF 2012.11.23.] Added this rule.

```

⟨Rules 11⟩ +≡
[statement : CD_ZZ RESPONSE_ZZ INTEGER_ZZ STRING_ZZ STRING_ZZ STRING_ZZ STRING_ZZ]
{
    Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
        *>(zzget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " << "statement:CD_ZZ\R\
            ESPONSE_ZZ\INTEGER_ZZ" << "STRING_ZZ\STRING_ZZ\STRING_ZZ\STRING_ZZ:" <<
            endl << "INTEGER_ZZ"(status_code)== " << $3 << endl <<
            "STRING_ZZ"(requested_directory)== " << $4 << endl <<
            "STRING_ZZ"(previous_working_directory)== " << $5 << endl <<
            "STRING_ZZ"(current_working_directory)== " << $6 << endl <<
            "STRING_ZZ"(command_output_or_response_from_server)== " << $7 << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    string temp_str = $7;
    lock_cout_mutex();
    lock_cerr_mutex();
    cout << "cd-->" << endl << "Exit status:|||||||||||||||||" <<
        $3 << endl << "Requested directory:|||||||||||||" <<
        $4 << endl << "Previous iRODS working directory:||" << $5 <<
        endl << "Current iRODS working directory:||" << $6 << endl <<
        "Response:|||||||||||||" << temp_str << endl;
    if (temp_str.empty() ∨ temp_str[temp_str.size() - 1] ≠ '\n') cout << endl;
    if ($3 ≡ 4) {
        cout << "ERROR! 'cd' command failed. " << "Server-side current\
            iRODS working directory is in an undefined state." << endl <<
            "Try resetting using 'cd' with no argument, or with a valid directory\"
            "as the argument." << endl;
        ++param->errors_occurred;
    }
    else if ($3 ≠ 0) {
        cout << "ERROR! 'cd' command failed. " << "Server-side current iRODS working direc\
            tory unchanged." << endl << "Will try to continue." << endl;
        ++param->errors_occurred;
    }
    else {

```

```

param→irods_current_dir = [$6];
cout << "'cd' command succeeded." Server-side current iRODS working \
    directory changed to " << " " << param→irods_current_dir << "." << endl;
}
cout << endl << endl;
unlock_cerr_mutex();
unlock_cout_mutex();
}
;

```

467. ⟨statement⟩ → MKDIR_ZZ RESPONSE_ZZ INTEGER_ZZ STRING_ZZ STRING_ZZ STRING_ZZ. [LDF 2012.11.29.] ■

Integer: Status code. 0 for success, non-zero for failure.

[LDF 2012.11.29.]

Log

[LDF 2012.11.29.] Added this rule.

```

⟨Rules 11⟩ +≡
[statement : MKDIR_ZZ RESPONSE_ZZ INTEGER_ZZ STRING_ZZ STRING_ZZ STRING_ZZ]
{
    Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
        *>(zzget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param→PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param→thread_ctr << "]"
            << "statement: MKDIR_ZZ RESPONSE_ZZ INTEGER_ZZ STRING_ZZ STRING_ZZ STRING_ZZ :"
            << endl << "'INTEGER_ZZ'(status_code)=="
            << [$3] << endl << "'STRING_ZZ'=="
            << [$4] << endl << "'STRING_ZZ'=="
            << [$6] << endl;
        unlock_cerr_mutex();
    } /* if (param→PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    lock_cout_mutex();
    lock_cerr_mutex();
    cout << "mkdir-->" << endl << "Exit status: "
        << [$3] << endl <<
        "Requested directories: "
        << [$4] << endl << "Command output: ";
    if (strlen([$5]) > 0) cout << [$5] << endl;
    else cout << "(None)" << endl;
    cout << "Server message: "
        << [$6] << endl;
    unlock_cerr_mutex();
    unlock_cout_mutex();
}
;

```

468. <statement> → RM_ZZ RESPONSE_ZZ INTEGER_ZZ STRING_ZZ STRING_ZZ STRING_ZZ. [LDF 2012.11.30.] ■
 Integer: Status code. 0 for success, non-zero for failure.
 [LDF 2012.11.30.]

Log

[LDF 2012.11.30.] Added this rule.

```
<Rules 11> +≡
[statement:RM_ZZ_RESPONSE_ZZ_INTEGER_ZZ_STRING_ZZ_STRING_ZZ_STRING_ZZ]
{
  Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
    *>(zzget_extra(parameter));
#if DEBUG_COMPILE
  if (param->PARSER_DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << param->thread_ctr << "] " <<
      "'statement:RM_ZZ_RESPONSE_ZZ_INTEGER_ZZ_STRING_ZZ_STRING_ZZ_STRING_ZZ':"
      endl << "'INTEGER_ZZ'(status_code)==" << [$3] << endl << "'STRING_ZZ'=="
      [$4] << endl << "'STRING_ZZ'==" << [$4] << endl << "'STRING_ZZ'==" << [$6] << endl;
    unlock_cerr_mutex();
  } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
  lock_cout_mutex();
  lock_cerr_mutex();
  cout << "rm-->" << endl << "Exit_status:XXXXXXXXXXXXXXXXXXXXXX" <<
    [$3] << endl << "Files_and_directories_in_delete_request:uu" << [$4] << endl <<
    "Command_output:XXXXXXXXXXXXXXXXXXXXXX";
  if (strlen([$5]) > 0) cout << [$5] << endl;
  else cout << "(None)" << endl;
  cout << "Server_message:XXXXXXXXXXXXXXXXXXXXXX" << [$6] << endl << endl;
  unlock_cout_mutex();
  unlock_cerr_mutex();
  if ([$3] != 0) ++param->errors_occurred;
}
;
```

469. ⟨statement⟩ → SERVER_ZZ SENDING_ZZ METADATA_ZZ FILE_ZZ STRING_ZZ UNSIGNED_INTEGER_ZZ.
[LDF 2012.12.13.]

Log

[LDF 2012.12.13.] Added this rule.
[LDF 2012.12.31.] Added UNSIGNED_INTEGER_ZZ.

```

⟨Rules 11⟩ +≡
[statement: SERVER_ZZ SENDING_ZZ METADATA_ZZ FILE_ZZ STRING_ZZ UNSIGNED_INTEGER_ZZ] {
    Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type*>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] In 'zzparse', " << endl <<
            "rule 'statement: SERVER_ZZ SENDING_ZZ METADATA_ZZ FILE_ZZ STRING_ZZ'" <<
            "UNSIGNED_INTEGER_ZZ' :" << endl << "'STRING_ZZ'" == " " << $5 << endl <<
            "'UNSIGNED_INTEGER_ZZ'" == " " << $6 << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    string temp_filename;
    status = param->receive_file("", "", true, 0, &temp_filename);
    if (status != 0) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] ERROR! In 'zzparse', rule" <<
            "statement: SERVER_ZZ SENDING_ZZ METADATA_ZZ FILE_ZZ" <<
            "STRING_ZZ UNSIGNED_INTEGER_ZZ' :" << endl <<
            "'Scan_Parser_Parameter_Type::receive_file' failed, returning" <<
            status << "." << endl << "Failed to receive or store metadata file." << endl <<
            "Will try to continue." << endl;
        unlock_cerr_mutex();
        ++param->errors_occurred;
    } /* if (status != 0) */
    else /* status == 0 */
    {
        lock_cout_mutex();
        lock_cerr_mutex();
        cout << "Received metadata for iRODS object " << $5 << "." << endl <<
            "Stored in temporary file: " << temp_filename << endl << endl;
        unlock_cerr_mutex();
        unlock_cout_mutex();
    } /* else (status == 0) */
}

```

470.

Log

[LDF 2012.12.31.] Added this section.

```

⟨Rules 11⟩ +≡
if ( [$6] & 1U)      /* output option set [LDF 2012.12.31.] */
{
    lock_cout_mutex();
    lock_cerr_mutex();
    cout << "Outputting_file" << temp_filename << ":" << endl;
    temp_strm << "cat" << temp_filename;
    errno = 0;
    status = system(temp_strm.str().c_str());
    cout << endl;
    unlock_cerr_mutex();
    unlock_cout_mutex();
    temp_strm.str("");
    if (status != 0) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]_ERROR!_In_`zzparse'," << endl <<
            "rule_`statement:_SERVER_ZZ_SENDING_ZZ_METADATA_ZZ_FILE_ZZ_STRING_ZZ" <<
            "UNSIGNED_INTEGER_ZZ:" << endl << "'system'_failed,_returning" << status << ":" <<
            endl << strerror(errno) << endl;
        if (WIFEXITED(status)) {
            cerr << "WEXITSTATUS(" << status << ")_==_WEXITSTATUS(status)" << endl;
        }
        else {
            cerr << "Process_didn't_exit." << endl;
        }
        unlock_cerr_mutex();
    }      /* if (status != 0) */
#endif DEBUG_COMPILE
else
{
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]_In_`zzparse'," << endl <<
            "rule_`statement:_SERVER_ZZ_SENDING_ZZ_METADATA_ZZ_FILE_ZZ_STRING_ZZ" <<
            "UNSIGNED_INTEGER_ZZ:" << endl << "'system'_succeeded." << endl;
        unlock_cerr_mutex();
    }      /* else if (param->PARSER_DEBUG) */
#endif      /* DEBUG_COMPILE */
}      /* if ([$6] & 1U) */
#endif DEBUG_COMPILE
else
{
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]_In_`zzparse'," << endl <<
            "rule_`statement:_SERVER_ZZ_SENDING_ZZ_METADATA_ZZ_FILE_ZZ_STRING_ZZ" <<
            "UNSIGNED_INTEGER_ZZ:" << endl << "Not_outputting_file" << temp_filename << ":" <<
            endl;
        unlock_cerr_mutex();
    }
}

```

```
    } /* else if (param->PARSER_DEBUG) */  
#endif /* DEBUG_COMPILE */
```

471.

$\langle \text{Rules } 11 \rangle +\equiv$
 $\quad param\text{-PARSER_DEBUG} = save\text{-PARSER_DEBUG}; \}$;

472. <statement> → ADD_ZZ METADATA_ZZ RESPONSE_ZZ INTEGER_ZZ STRING_ZZ STRING_ZZ STRING_ZZ.
[LDF 2012.12.14.]

Integer: Status code. 0 for success, non-zero for failure.
[LDF 2012.12.14.]

Log

[LDF 2012.12.14.] Added this rule.

```

(Rules 11) +≡
statement : ADD_ZZ METADATA_ZZ RESPONSE_ZZ INTEGER_ZZ STRING_ZZ STRING_ZZ STRING_ZZ
{
    Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
        *>(zzget_extra(parameter));
#endif DEBUG_COMPILE
if (param->PARSER_DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread_" << param->thread_ctr << "] " <<
        "'statement:ADD_ZZ METADATA_ZZ RESPONSE_ZZ INTEGER_ZZ' " <<
        "STRING_ZZ STRING_ZZ STRING_ZZ'" : " << endl << "'INTEGER_ZZ'(status_code)" == " " <<
        $4 << endl << "'STRING_ZZ'" == " " << $5 << endl << "'STRING_ZZ'" == " " << $6 << endl <<
        "'STRING_ZZ'" == " " << $7 << endl;
    unlock_cerr_mutex();
} /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
lock_cout_mutex();
lock_cerr_mutex();
cout << "add_metadata-->" << endl << "Exit status: " << endl << $4 << endl << "Metadata file" << endl << $5 << endl <<
    "iRODS_object" << endl << $6 << endl << "Server message:" ;
if (strlen($7) > 0) cout << endl << "uuu" << $7 << endl;
else cout << "uu(None)" << endl;
cout << endl;
unlock_cerr_mutex();
unlock_cout_mutex();
}
;
```

473. <statement> → STORE_ZZ METADATA_ZZ RESPONSE_ZZ INTEGER_ZZ STRING_ZZ STRING_ZZ STRING_ZZ [LDF 2013.03.07.]

Integer: Status code. 0 for success, non-zero for failure.
[LDF 2013.03.07.]

Log

[LDF 2013.03.07.] Added this rule.

```
<Rules 11> +≡
  statement: STORE_ZZ METADATA_ZZ RESPONSE_ZZ INTEGER_ZZ STRING_ZZ STRING_ZZ STRING_ZZ
{
  Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
    *>(zzget_extra(parameter));
#ifndef DEBUG_COMPILE
  if (param->PARSER_DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << param->thread_ctr << "] " <<
      "statement: STORE_ZZ METADATA_ZZ RESPONSE_ZZ INTEGER_ZZ " <<
      "STRING_ZZ STRING_ZZ STRING_ZZ :" << endl << "'INTEGER_ZZ'" << (status_code) == " <<
      [$4] << endl << "'STRING_ZZ'" == " << [$5] << endl << "'STRING_ZZ'" == " << [$6] << endl <<
      "'STRING_ZZ'" == " << [$7] << endl;
    unlock_cerr_mutex();
  } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
  lock_cout_mutex();
  lock_cerr_mutex();
  cout << "store_metadata-->" << endl << "Exit status: " << [$4] << endl <<
    [$4] << endl << "Dublin Core metadata/iRODS_object_file: " << [$5] << endl <<
    "iRODS_object_referred_to: " << [$6] << endl << "Server message:";
  if (strlen([$7]) > 0) cout << endl << " " << [$7] << endl;
  else cout << " (None)" << endl;
  cout << endl;
  unlock_cerr_mutex();
  unlock_cout_mutex();
}
;
```

474. ⟨statement⟩ → END_SERVER_ZZ RESPONSE_ZZ INTEGER_ZZ. [LDF 2013.04.03.]

Log

[LDF 2013.04.03.] Added this rule.

```

⟨Rules 11⟩ +≡
statement: END_SERVER_ZZ RESPONSE_ZZ INTEGER_ZZ
{
    Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
        *>(zzget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
            << "statement:END_SERVER_ZZ_RESPONSE_ZZ_INTEGER_ZZ."
            << endl <<
            "'INTEGER_ZZ'" << [$3] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    lock_cout_mutex();
    lock_cerr_mutex();
    cout << "end_server_response--" << endl << "Response_code:" << [$3];
    if ([\$3] == 0) cout << "(Success)";
    else if ([\$3] == 1) cout << endl << "ERROR! 'END_SERVER' command not enabled.";
    else cout << endl << "ERROR! 'END_SERVER' command failed";
    cout << endl;
    unlock_cerr_mutex();
    unlock_cout_mutex();
    if ([\$3] == 0) {
        lock_cout_mutex();
        lock_cerr_mutex();
        cerr << "Server and client will both exit."
            << "Exiting 'zzparse' successfully with return value 2." << endl;
        unlock_cerr_mutex();
        unlock_cout_mutex();
        return 2;
    } /* if ([\$3] == 0) */
}
;
```

475. ⟨statement⟩ → SLEEP_ZZ INTEGER_ZZ. [LDF 2013.05.02.]

Log

[LDF 2013.05.02.] Added this rule.

```

⟨Rules 11⟩ +≡
statement: SLEEP_ZZ INTEGER_ZZ
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] In 'zzlex', rule"
            << "statement:SLEEP_ZZ,INTEGER_ZZ." << endl << 'INTEGER_ZZ' <=
            [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    if (sleep_client_enabled) {
#ifndef DEBUG_COMPILE
        if (param->PARSER_DEBUG) {
            lock_cerr_mutex();
            cerr << "[Thread" << param->thread_ctr << "] In 'zzlex', rule"
                << "statement:SLEEP_ZZ,INTEGER_ZZ:" << endl <<
                "'sleep_client_enabled'" <= 'true'. Will go to sleep for
                [$2] << endl;
            unlock_cerr_mutex();
        } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
        lock_cout_mutex();
        lock_cerr_mutex();
        cout << "Sleep-->" << endl << "Sleep value:" << [$2] << endl <<
            "Going to sleep for" << endl << [$2] << "seconds..." << flush;
        unlock_cerr_mutex();
        unlock_cout_mutex();
        status = sleep([$2]);
        lock_cout_mutex();
        lock_cerr_mutex();
        cout << endl;
        if (status == 0) cout << "Woke up after" << [$2] << "seconds." << endl;
        else cout << "Woke up after" << [$2] << " - " << status << " == "
            << ($2 - status) << endl << "Unslept seconds:" << [$2] << endl;
        unlock_cerr_mutex();
        unlock_cout_mutex();
    } /* if (sleep_client_enabled) */
    else /* !sleep_client_enabled */
    {
        lock_cerr_mutex();

```

```
cerr << "[Thread" << param->thread_ctr << "] " WARNING! In 'zzlex', rule " <<
"statement:SLEEP_ZZ_INTEGER_ZZ:" << endl <<
"sleep_client_enabled'" == 'false'. Not going to sleep for " <<
[$2] << " seconds." << endl << "Continuing." << endl;
unlock_cerr_mutex();
++param->warnings_occurred;
} /* else ( $\neg$ sleep_client_enabled) */
param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

476. ⟨statement⟩ → SLEEP_ZZ RESPONSE_ZZ INTEGER_ZZ INTEGER_ZZ. [LDF 2013.04.19.]

Log

[LDF 2013.04.19.] Added this rule.

```

⟨Rules 11⟩ +≡
statement: SLEEP_ZZ RESPONSE_ZZ INTEGER_ZZ INTEGER_ZZ
{
    Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
        *>(zzget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
            << "statement:SLEEP_ZZ_RESPONSE_ZZ_INTEGER_ZZ_INTEGER_ZZ."
            << endl << "INTEGER_ZZ" << $3 << endl << "INTEGER_ZZ" << $4 << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    lock_cout_mutex();
    lock_cerr_mutex();
    cout << "Sleep_response--" << endl << "Response_code:" << $3 << endl <<
        "Sleep_value:" << $4 << endl;
    if ($3 == 0) cout << "Server_thread_will_sleep_for"
        << $4 << "seconds." << endl;
    else if ($3 == 1) {
        cout << "ERROR! Server_not_put_to_sleep. Continuing."
            << endl;
        ++param->errors_occurred;
    }
    else if ($3 == 3) {
        cout << "WARNING! sleep_client_enabled' == 'false' on the server-side."
            << endl <<
            "Server_not_sending 'SLEEP'" << $4 << "' command to client."
            << endl;
        ++param->warnings_occurred;
    }
    else {
        cout << "ERROR! Unknown_server-side_error." << endl;
        ++param->errors_occurred;
    }
    unlock_cerr_mutex();
    unlock_cout_mutex();
}
;
```

477. ⟨statement⟩ → SIGNAL_ZZ INTEGER_ZZ. [LDF 2013.05.02.]

We don't bother checking that **[\$2]** is a valid signal number here, because the server already does this in the rules that tell it to send this command back to the client. [LDF 2013.05.02.]

Log

[LDF 2013.05.02.] Added this rule.

```

⟨Rules 11⟩ +≡
[statement: SIGNAL_ZZ INTEGER_ZZ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] In 'zzparse', rule"
            << "statement: SIGNAL_ZZ INTEGER_ZZ'." << endl << "'INTEGER_ZZ'" << " " << [$2] << endl;
        if (signal_client_enabled) cerr << "'signal_client_enabled'" << "true'." << endl <<
            "Calling 'pthread_kill(pthread_self(), " << [$2] << ")" << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    if (signal_client_enabled) {
        lock_cout_mutex();
        lock_cerr_mutex();
        cout << "Sending signal" << signal_name_map[$2] << "(" << [$2] << ") to self." << endl;
        unlock_cerr_mutex();
        unlock_cout_mutex();
        pthread_kill(pthread_self(), [$2]);
    }
    else {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] WARNING! In 'zzparse', rule"
            << "statement: SIGNAL_ZZ INTEGER_ZZ':" << endl <<
            "'signal_client_enabled'" << "false'." << "Not sending signal" <<
            signal_name_map[$2] << "(" << [$2] << ")" << " to self." << endl << "Continuing." << endl;
        unlock_cerr_mutex();
        ++param->warnings_occurred;
    } /* else */
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

478. ⟨statement⟩ → SHOW_ZZ CERTIFICATE_ZZ RESPONSE_ZZ INTEGER_ZZ ⟨certificate item list⟩. [LDF 2013.05.03.] ■

Log

[LDF 2013.05.03.] Added this rule.

```

⟨Rules 11⟩ +≡
statement: SHOW_ZZ CERTIFICATE_ZZ RESPONSE_ZZ INTEGER_ZZ certificate_item_list
{
    Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " << 'statement:' << SHOW_ZZ << CERTIFICATE_ZZ << RESP \
            ONSE_ZZ << INTEGER_ZZ << certificate_item_list'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    temp_strm.str("");
    if ([§4] ≡ 0 ∧ param->cert_ptr ≠ 0) param->cert_ptr->show("X.509_Certificate:", &temp_strm, false);
    lock_cerr_mutex();
    lock_cout_mutex();
    cout << "Certificate_response-->" << endl << "Response_code:" << [§4] << endl;
    if ([§4] ≡ 3) {
        cout << "WARNING! User not authorized to show all certificates." << endl;
        ++param->warnings_occurred;
    }
    else if ([§4] ≠ 0) {
        cout << "ERROR! Server failed to retrieve certificate(s)." << endl;
        ++param->errors_occurred;
    }
    if (¬temp_strm.str().empty()) cout << temp_strm.str();
    cout << endl;
    unlock_cout_mutex();
    unlock_cerr_mutex();
    temp_strm.str("");
    delete param->cert_ptr;
    param->cert_ptr = 0;
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

479. ⟨certificate item list⟩.

480. ⟨certificate item list⟩ → Empty. [LDF 2013.05.03.]

Log

[LDF 2013.05.03.] Added this rule.

```

⟨Rules 11⟩ +≡
certificate_item_list:/*Empty*/
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " << 'certificate_item_list:/*Empty*/' . " <<
            endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    if (param->cert_ptr) param->cert_ptr->clear();
    else param->cert_ptr = new X509_Cert_Type;
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

481. ⟨certificate item list⟩ → ⟨certificate item list⟩ ⟨certificate item⟩. [LDF 2013.05.15.]

Log

[LDF 2013.05.15.] Added this rule.

```

⟨Rules 11⟩ +≡
certificate_item_list:certificate_item_list_certificate_item
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'certificate_item_list:certificate_item_list_certificate_item'" . " << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

482. ⟨certificate item⟩. [LDF 2013.05.16.]

Log

[LDF 2013.05.16.] Added this section.

483. ⟨certificate item list⟩ → CERTIFICATE_ID_ZZ UNSIGNED_INTEGER_ZZ. [LDF 2013.05.15.]

Log

[LDF 2013.05.15.] Added this rule.

⟨Rules 11⟩ +≡

```

certificate_item :_CERTIFICATE_ID_ZZ_UNSIGNED_INTEGER_ZZ_
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread_" << param->thread_ctr << "]_" <<
            "'certificate_item:_CERTIFICATE_ID_ZZ_UNSIGNED_INTEGER_ZZ'." << endl <<
            "'UNSIGNED_INTEGER_ZZ'" <= <= [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->cert_ptr->certificate_id = [$2];
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

484. ⟨certificate item list⟩ → USER_ID_ZZ UNSIGNED_INTEGER_ZZ. [LDF 2013.05.15.]

Log

[LDF 2013.05.15.] Added this rule.

```

⟨Rules 11⟩ +≡
[certificate_item:_USER_ID_ZZ_UNSIGNED_INTEGER_ZZ_]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread_" << param->thread_ctr << "]_" <<
            "'certificate_item:_USER_ID_ZZ_UNSIGNED_INTEGER_ZZ'." << endl <<
            "'UNSIGNED_INTEGER_ZZ'"_==_" << [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->cert_ptr->user_id = [$2];
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

485. ⟨certificate item list⟩ → USER_NAME_ZZ STRING_ZZ. [LDF 2013.05.15.]

Log

[LDF 2013.05.15.] Added this rule.

```

⟨Rules 11⟩ +≡
[certificate_item:_USER_NAME_ZZ_STRING_ZZ_]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread_" << param->thread_ctr << "]_" <<
            "'certificate_item:_USER_NAME_ZZ_STRING_ZZ'." << endl <<
            "'STRING_ZZ'"_==_" << [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->cert_ptr->user_name = [$2];
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

486. <certificate item list> → ISSUER_CERT_ID_ZZ UNSIGNED_INTEGER_ZZ. [LDF 2013.05.16.]

Log

[LDF 2013.05.16.] Added this rule.

```

⟨Rules 11⟩ +≡
[certificate_item:_ISSUER_CERT_ID_ZZ_UNSIGNED_INTEGER_ZZ_]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread_" << param->thread_ctr << "]_" <<
            "'certificate_item:_ISSUER_CERT_ID_ZZ_UNSIGNED_INTEGER_ZZ'." << endl <<
            "'UNSIGNED_INTEGER_ZZ'" == " " << [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->cert_ptr->issuer_cert_id = [$2];
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

487. <certificate item list> → IS_CA_ZZ INTEGER_ZZ. [LDF 2013.05.16.]

Log

[LDF 2013.05.16.] Added this rule.

```

⟨Rules 11⟩ +≡
[certificate_item:_IS_CA_ZZ_INTEGER_ZZ_]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread_" << param->thread_ctr << "]_" <<
            "'certificate_item:_IS_CA_ZZ_INTEGER_ZZ'." << endl <<
            "'INTEGER_ZZ'" == " " << [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->cert_ptr->is_ca = static_cast<bool>([$2]);
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

488. ⟨certificate item list⟩ → IS_PROXY_ZZ INTEGER_ZZ. [LDF 2013.05.16.]

Log

[LDF 2013.05.16.] Added this rule.

```

⟨Rules 11⟩ +≡
[certificate_item: IS_PROXY_ZZ INTEGER_ZZ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'certificate_item: IS_PROXY_ZZ INTEGER_ZZ'." << endl << "'INTEGER_ZZ'" <=
            [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->cert_ptr->is_proxy = static_cast<bool>([$2]);
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

489. ⟨certificate item list⟩ → ORGANIZATION_ZZ STRING_ZZ. [LDF 2013.05.16.]

Log

[LDF 2013.05.16.] Added this rule.

```

⟨Rules 11⟩ +≡
[certificate_item: ORGANIZATION_ZZ STRING_ZZ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'certificate_item: ORGANIZATION_ZZ STRING_ZZ'." << endl << "'STRING_ZZ'" <=
            [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->cert_ptr->organization = [$2];
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

490. <certificate item list> → ORGANIZATIONAL_UNIT_NAME_ZZ STRING_ZZ. [LDF 2013.05.16.]

Log

[LDF 2013.05.16.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
[certificate_item:_ORGANIZATIONAL_UNIT_NAME_ZZ_STRING_ZZ_]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread_" << param->thread_ctr << "]_" <<
            "'certificate_item:_ORGANIZATIONAL_UNIT_NAME_ZZ_STRING_ZZ'." << endl <<
            "'STRING_ZZ'" <= " " << [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->cert_ptr->organizationalUnitName = [$2];
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

491. <certificate item list> → COMMON_NAME_ZZ STRING_ZZ. [LDF 2013.05.16.]

Log

[LDF 2013.05.16.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
[certificate_item:_COMMON_NAME_ZZ_STRING_ZZ_]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread_" << param->thread_ctr << "]_" <<
            "'certificate_item:_COMMON_NAME_ZZ_STRING_ZZ'." << endl <<
            "'STRING_ZZ'" <= " " << [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->cert_ptr->commonName = [$2];
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

492. ⟨certificate item list⟩ → COUNTRY_NAME_ZZ STRING_ZZ. [LDF 2013.05.16.]

Log

[LDF 2013.05.16.] Added this rule.

```

⟨Rules 11⟩ +≡
[certificate_item : COUNTRY_NAME_ZZ STRING_ZZ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'certificate_item:' COUNTRY_NAME_ZZ STRING_ZZ'." << endl << "'STRING_ZZ'" <=
            [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->cert_ptr->countryName = [$2];
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

493. ⟨certificate item list⟩ → LOCALITY_NAME_ZZ STRING_ZZ. [LDF 2013.05.16.]

Log

[LDF 2013.05.16.] Added this rule.

```

⟨Rules 11⟩ +≡
[certificate_item : LOCALITY_NAME_ZZ STRING_ZZ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'certificate_item:' LOCALITY_NAME_ZZ STRING_ZZ'." << endl << "'STRING_ZZ'" <=
            [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->cert_ptr->localityName = [$2];
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

494. ⟨certificate item list⟩ → STATE_OR_PROVINCE_NAME_ZZ STRING_ZZ. [LDF 2013.05.16.]

Log

[LDF 2013.05.16.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
[certificate_item:_STATE_OR_PROVINCE_NAME_ZZ_STRING_ZZ_]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread_" << param->thread_ctr << "]_" <<
            "'certificate_item:_STATE_OR_PROVINCE_NAME_ZZ_STRING_ZZ'." << endl <<
            "'STRING_ZZ'" <= <= [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->cert_ptr->stateOrProvinceName = [$2];
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

495. ⟨certificate item list⟩ → SERIAL_NUMBER_ZZ UNSIGNED_LONG_INTEGER_ZZ. [LDF 2013.05.16.]

Log

[LDF 2013.05.16.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
[certificate_item:_SERIAL_NUMBER_ZZ_UNSIGNED_LONG_INTEGER_ZZ_]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread_" << param->thread_ctr << "]_" <<
            "'certificate_item:_SERIAL_NUMBER_ZZ_UNSIGNED_LONG_INTEGER_ZZ'." <<
            endl << "'UNSIGNED_LONG_INTEGER_ZZ'" <= <= [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->cert_ptr->serialNumber = [$2];
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

496. ⟨certificate item list⟩ → VALIDITY_NOT_BEFORE_ZZ UNSIGNED_LONG_INTEGER_ZZ. [LDF 2013.05.16.]

Log

[LDF 2013.05.16.] Added this rule.

```

⟨Rules 11⟩ +≡
[certificate_item:VALIDITY_NOT_BEFORE_ZZ_UNSIGNED_LONG_INTEGER_ZZ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'certificate_item:VALIDITY_NOT_BEFORE_ZZ_UNSIGNED_LONG_INTEGER_ZZ'." << endl <<
            "'UNSIGNED_LONG_INTEGER_ZZ'" <=> [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->cert_ptr->Validity_notBefore = [$2];
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

497. ⟨certificate item list⟩ → VALIDITY_NOT_AFTER_ZZ UNSIGNED_LONG_INTEGER_ZZ. [LDF 2013.05.16.]

Log

[LDF 2013.05.16.] Added this rule.

```

⟨Rules 11⟩ +≡
[certificate_item:VALIDITY_NOT_AFTER_ZZ_UNSIGNED_LONG_INTEGER_ZZ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'certificate_item:VALIDITY_NOT_AFTER_ZZ_UNSIGNED_LONG_INTEGER_ZZ'." << endl <<
            "'UNSIGNED_LONG_INTEGER_ZZ'" <=> [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->cert_ptr->Validity_notAfter = [$2];
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

498. <user info item> → VALIDITY_NOT_BEFORE_ZZ UNSIGNED_LONG_INTEGER_ZZ. [LDF 2013.05.22.]

Log

[LDF 2013.05.22.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
[ user_info_item : VALIDITY_NOT_BEFORE_ZZ UNSIGNED_LONG_INTEGER_ZZ ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'user_info_item:VALIDITY_NOT_BEFORE_ZZ_UNSIGNED_LONG_INTEGER_ZZ'." << endl <<
            "'UNSIGNED_LONG_INTEGER_ZZ'" <= << [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->user_info_ptr->certificate.Validity_notBefore = [$2];
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

499. <user info item> → VALIDITY_NOT_AFTER_ZZ UNSIGNED_LONG_INTEGER_ZZ. [LDF 2013.05.22.]

Log

[LDF 2013.05.22.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
[ user_info_item : VALIDITY_NOT_AFTER_ZZ UNSIGNED_LONG_INTEGER_ZZ ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'user_info_item:VALIDITY_NOT_AFTER_ZZ_UNSIGNED_LONG_INTEGER_ZZ'." <<
            endl << "'UNSIGNED_LONG_INTEGER_ZZ'" <= << [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->user_info_ptr->certificate.Validity_notAfter = [$2];
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

500. ⟨statement⟩ → SHOW_ZZ GROUPS_ZZ RESPONSE_ZZ INTEGER_ZZ STRING_ZZ ⟨group item list⟩. [LDF 2013.06.05.] █

Log

[LDF 2013.06.05.] Added this rule.

```

⟨Rules 11⟩ +≡
statement: SHOW_ZZ GROUPS_ZZ RESPONSE_ZZ INTEGER_ZZ STRING_ZZ group_item_list
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "statement: SHOW_ZZ GROUPS_ZZ RESPONSE_ZZ INTEGER_ZZ STRING_ZZ " <<
            "group_item_list" . " << endl << 'INTEGER_ZZ' (4) == " << $4 << endl <<
            "STRING_ZZ" (5) == " << $5 << endl << "param->group_vector.size()" == " <<
            param->group_vector.size() << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    temp_strm.str("");
    lock_cerr_mutex();
    lock_cout_mutex();
    cout << "Show_groups_response-->" << endl << "Response_code:" << $4 << endl;
    if ($4 == 3) {
        cout << "WARNING! User not authorized to show all groups." << endl;
        ++param->warnings_occurred;
    }
    else if ($4 != 0) {
        cout << "ERROR! Server failed to retrieve groups(s)." << endl;
        ++param->errors_occurred;
    }
    else if (param->group_vector.size() == 0) cout << "No group info to show." << endl;
    else {
        cout << "Group info for " << param->group_vector.size() << " groups: " << endl;
        for (vector<Group_Type>::iterator iter = param->group_vector.begin();
             iter != param->group_vector.end(); ++iter) {
            iter->show("", &temp_strm);
        }
    }
    cout << temp_strm.str();
    unlock_cout_mutex();
    unlock_cerr_mutex();
    temp_strm.str("");
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

501. *group-item* and *group-item-list*. [LDF 2013.06.05.]

Log

[LDF 2013.06.05.] Added this section.

< Token and type declarations 128 > +≡

%type<int_value> group_item	%type<int_value> group_item_list
-----------------------------	----------------------------------

502. *<group item list>* → *Empty*. [LDF 2013.06.05.]

Log

[LDF 2013.06.05.] Added this rule.

< Rules 11 > +≡

group_item_list : /* Empty */

```

{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false;      /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] In 'zzparse', rule"
            << "'group_item_list : /* Empty */'." << endl;
        unlock_cerr_mutex();
    }      /* if (param->PARSER_DEBUG) */
#endif      /* DEBUG_COMPILE */
    param->group_vector.clear();
    param->group_vector.push_back(Group_Type());
}
;
```

503. <group item list> → <group item list> <group item> [LDF 2013.06.05.]

Log

[LDF 2013.06.05.] Added this rule.

```
<Rules 11> +≡
[group_item_list:✉group_item_list✉group_item]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]In'zzparse',rule"
            << "group_item_list:✉group_item_list✉group_item'." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
}
;
```

504. <group item> → GROUP_ID_ZZ INTEGER_ZZ. [LDF 2013.06.05.]

Log

[LDF 2013.06.05.] Added this rule.

```
<Rules 11> +≡
[group_item:✉GROUP_ID_ZZ✉INTEGER_ZZ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]In'zzparse',rule"
            << "group_item:✉GROUP_ID_ZZ✉INTEGER_ZZ'." << endl << "'INTEGER'(2)==" << $2 <<
            endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    if (!(param->group_vector.back().group_id == 0 || param->group_vector.back().group_id == $2)) {
        param->group_vector.push_back(Group_Type());
    }
    param->group_vector.back().group_id = $2;
}
;
```

505. <group item> → GROUP_NAME_ZZ STRING_ZZ. [LDF 2013.06.05.]

Log

[LDF 2013.06.05.] Added this rule.

```

⟨Rules 11⟩ +≡
[group_item:GROUP_NAME_ZZ_STRING_ZZ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread]" << param->thread_ctr << "]In'zzparse',rule" <<
            "'group_item:GROUP_NAME_ZZ_STRING_ZZ'." << endl << "'STRING'(2)=='" << [$2] <<
            endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->group_vector.back().group_name = [$2];
}
;
```

506. <group item> → CREATOR_ID_ZZ INTEGER_ZZ. [LDF 2013.06.05.]

Log

[LDF 2013.06.05.] Added this rule.

```

⟨Rules 11⟩ +≡
[group_item:CREATOR_ID_ZZ_INTEGER_ZZ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread]" << param->thread_ctr << "]In'zzparse',rule" <<
            "'group_item:CREATOR_ID_ZZ_INTEGER_ZZ'." << endl << "'INTEGER'(2)=='" << [$2] <<
            endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->group_vector.back().creator_id = [$2];
}
;
```

507. <group item> → CREATOR_USERNAME_ZZ STRING_ZZ. [LDF 2013.06.05.]

Log

[LDF 2013.06.05.] Added this rule.

```
<Rules 11> +≡
[group_item:_CREATOR_USERNAME_ZZ_STRING_ZZ_]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread_" << param->thread_ctr << "] In 'zzparse', rule"
            << "'group_item:_CREATOR_USERNAME_ZZ_STRING_ZZ'." << endl << "'STRING'"_2==_
            [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->group_vector.back().creator_username = [$2];
}
;
```

508. <group item> → CREATED_ZZ UNSIGNED_INTEGER_ZZ. [LDF 2013.06.05.]

Log

[LDF 2013.06.05.] Added this rule.

```
<Rules 11> +≡
[group_item:_CREATED_ZZ_UNSIGNED_INTEGER_ZZ_]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread_" << param->thread_ctr << "] In 'zzparse', rule"
            << "'group_item:_CREATED_ZZ_UNSIGNED_INTEGER_ZZ'." << endl << "'UNSIGNED_INTEGER'"_2==_
            [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->group_vector.back().created = static_cast<time_t>([$2]);
}
;
```

509. <group item> → USER_ID_ZZ INTEGER_ZZ USER_NAME_ZZ STRING_ZZ PRIVILEGES_ZZ UNSIGNED_INTEGER_ZZ. [LDF 2013.06.05.]

Log

[LDF 2013.06.05.] Added this rule.

<Rules 11> +≡

```

[group_item:_USER_ID_ZZ_INTEGER_ZZ_USER_NAME_ZZ_STRING_ZZ_PRIVILEGES_ZZ_UNSIGNED_INTEGER_ZZ]
{
    Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread_" << param->thread_ctr << "] In 'zzparse', rule"
            << "group_item:_USER_ID_ZZ_INTEGER_ZZ_USER_NAME_ZZ_STRING_ZZ"
            << "PRIVILEGES_ZZ_UNSIGNED_INTEGER_ZZ'." << endl <<
            "'INTEGER_ZZ'"(2)(user_id)_<< "$2" << endl <<
            "'STRING_ZZ'"(4)(username)_<< "$4" << endl <<
            "'UNSIGNED_INTEGER_ZZ'"(6)(privileges)_<< oct << "$6" << "(octal)" << dec <<
            endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->group_vector.back().member_id_map.insert(make_pair($2, make_pair($4, $6)));
}
;
```

510. ⟨statement⟩ → AUTHENTICATION_ZZ ERROR_ZZ INTEGER_ZZ. [LDF 2013.05.10.]

Log

[LDF 2013.05.10.] Added this rule.

```

⟨Rules 11⟩ +≡
statement : AUTHENTICATION_ZZ ERROR_ZZ INTEGER_ZZ
{
    Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'statement:AUTHENTICATION_ZZ(ERROR_ZZ)INTEGER_ZZ'." << endl <<
            "'INTEGER_ZZ'" << "$2" << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    lock_cout_mutex();
    lock_cerr_mutex();
    cout << "Authentication_error-->" << endl << "Error_code:" << "$3" << endl << "Exiting." <<
        endl;
    unlock_cerr_mutex();
    unlock_cout_mutex();
    if ($3 ≡ 1) {
        lock_cerr_mutex();
        cerr << "Unauthenticated_connection_and\\\"DISTINGUISHED_NAME\\\"comma\\
            nd\\_was\\_either\\_not\\_\" << "sent\\_to\\_server,\\_or\\_failed." << endl <<
            "Please\\_note\\_that\\_unauthenticated\\_connections\\_are\\_only\\_for\\_testing\\_purposes!" <<
            endl << "Exiting." << endl;
        unlock_cerr_mutex();
    } /* if ($3 ≡ 1) */
    else if ($3 ≡ 2) {
        lock_cerr_mutex();
        cerr << "Server\\_failed\\_to\\_verify\\_user\\_certificate." << endl << "Exiting." << endl;
        unlock_cerr_mutex();
    } /* else if ($3 ≡ 2) */
param->response_deque.clear();
param->client_finished = param->server_finished = true;
return 0;
param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

511. ⟨statement⟩ → DISTINGUISHED_NAME_ZZ RESPONSE_ZZ INTEGER_ZZ STRING_ZZ INTEGER_ZZ STRING_ZZ. ■
[LDF 2013.05.14.]

Log

[LDF 2013.05.14.] Added this rule.

```

⟨Rules 11⟩ +≡
statement : DISTINGUISHED_NAME_ZZ RESPONSE_ZZ INTEGER_ZZ STRING_ZZ INTEGER_ZZ STRING_ZZ
{
    Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
        *>(zzget_extra(parameter));
#endif DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " << "statement:DISTING\"
            VISHED_NAME_ZZ RESPONSE_ZZ " << "INTEGER_ZZ STRING_ZZ INTEGER_ZZ STRING_ZZ' ." <<
            endl << "'INTEGER_ZZ' (Response_code) == " << $3 <<
            endl << "'STRING_ZZ' (Distinguished_Name) == " << $4 <<
            endl << "'INTEGER_ZZ' (User_ID) == " << $5 << endl <<
            "'STRING_ZZ' (Username) == " << $6 << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    lock_cout_mutex();
    lock_cerr_mutex();
    cout << "Distinguished_Name_response-->" << endl << "Response_code: " << $3 <<
        endl << "Distinguished_Name: " << $4 << endl << "User_ID: " << $5 <<
        endl << "Username: " << $6 << endl << endl;
    unlock_cerr_mutex();
    unlock_cout_mutex();
}
;
```

512. ⟨statement⟩ → GET_USER_ZZ RESPONSE_ZZ INTEGER_ZZ USER_ID_ZZ INTEGER_ZZ USER_NAME_ZZ STRING_ZZ [LDF 2013.05.16.]

Log

[LDF 2013.05.16.] Added this rule.

⟨Rules 11⟩ +≡

```

statement : GET_USER_ZZ RESPONSE_ZZ INTEGER_ZZ USER_ID_ZZ INTEGER_ZZ USER_NAME_ZZ STRING_ZZ [PRIVILEGES_ZZ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] In 'zzparse', rule" << endl <<
            "statement: GET_USER_ZZ RESPONSE_ZZ INTEGER_ZZ" << "USER_ID_ZZ INTEGER_\\
            ZZ USER_NAME_ZZ STRING_ZZ PRIVILEGES_ZZ UNSIGNED_INTEGER_ZZ'." << endl <<
            "Response_code:" << [$3] << endl << "User_ID:" << [$5] << endl << "Username:" <<
            [$7] << endl << "Privileges:" << [$9] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    if ([\$3] != 0) { /* This conditional suppresses output in the “normal” case. It was getting
                      annoying, seeing it all the time. [LDF 2013.08.07.] */
        lock_cout_mutex();
        lock_cerr_mutex();
        cout << "Get_user_response--" << endl << "Response_code:" << [$3] << endl;
        if ([\$3] == 0) {
            param->user_id = [$5];
            param->username = [$7];
            param->privileges = [$9];
            cout << "User_ID: " << param->user_id << endl << "Username: " << param->username << endl;
        }
    #if 0
        Scan_Parser_Parameter_Type::show_privileges(param->privileges, &cout);
    #endif
        cout << endl;
    }
    unlock_cerr_mutex();
    unlock_cout_mutex();
} /* if ([\$3] != 0) */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] In 'zzparse', rule" << endl <<
            "statement: GET_USER_ZZ RESPONSE_ZZ INTEGER_ZZ" << "USER_ID_ZZ INTEGER_\\
            ZZ USER_NAME_ZZ STRING_ZZ PRIVILEGES_ZZ UNSIGNED_INTEGER_ZZ'" << endl <<
            "'param->user_id' == " << param->user_id << endl << "'param->username' == " <<

```

```

    param->username << endl << "param->privileges'===" << oct << param->privileges <<
    "_(octal)" << dec << endl;
    unlock_cerr_mutex();
} /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

513. ⟨statement⟩ → GET_USER_INFO_ZZ RESPONSE_ZZ INTEGER_ZZ ⟨user info item list⟩ and ⟨statement⟩ → WHOAMI_ZZ RESPONSE_ZZ INTEGER_ZZ ⟨user info item list⟩ [LDF 2013.05.17.] [LDF 2013.05.19.]
These rules share an action. [LDF 2013.05.19.]

Log

[LDF 2013.05.17.] Added this rule.
[LDF 2013.05.19.] Added the WHOAMI_ZZ rule.

⟨Rules 11⟩ +≡

statement: GET_USER_INFO_ZZ RESPONSE_ZZ INTEGER_ZZ user_info_item_list	⟨Get user info action 514⟩
statement: WHOAMI_ZZ RESPONSE_ZZ INTEGER_ZZ user_info_item_list	⟨Get user info action 514⟩

514.

⟨Get user info action 514⟩ ≡

```

{ Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
    *>(zzget_extra(parameter));
bool save_PARSER_DEBUG = param->PARSER_DEBUG;
param->PARSER_DEBUG = false; /* true */
string temp_str;
string temp_str_1;
if ($1 ≡ GET_USER_INFO_ZZ) {
    temp_str = "GET_USER_INFO_ZZ";
    temp_str_1 = "Get_user_info";
}
else {
    temp_str = "WHOAMI_ZZ";
    temp_str_1 = "Whoami";
}
#endif DEBUG_COMPILE
if (param->PARSER_DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread_" << param->thread_ctr << "] In '_zzparse', '_rule" << endl << "statement:_" <<
        temp_str << "_RESPONSE_ZZ INTEGER_ZZ" << "user_info_item_list'." << endl;
    unlock_cerr_mutex();
} /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
lock_cout_mutex();
lock_cerr_mutex();
cout << temp_str_1 << "response-->" << endl << "Response_code:_" << [$3] << endl;

```

See also sections 515, 516, and 517.

This code is used in section 513.

515.

```

⟨ Get user info action 514 ⟩ +≡
if ([\$3] ≡ 0 ∧ param→user_info_ptr ≠ 0) {
    bool brief = ([\$1] ≡ WHOAMI_ZZ) ? true : false;
    param→user_info_ptr→distinguished_name = param→user_info_ptr→certificate;
    temp_strm.str("");
    param→user_info_ptr→show("User_Info:", &temp_strm, brief);
    cout ≪ temp_strm.str();
    temp_strm.str("");
} /* if ([\$3] ≡ 0 ∧ param→user_info_ptr ≠ 0) */
cout ≪ endl;
unlock_cerr_mutex();
unlock_cout_mutex();

```

516.

```

⟨ Get user info action 514 ⟩ +≡
#ifndef 0
if ([\$3] ≡ 0 ∧ param→user_info_ptr ≠ 0) { /* Show *param→user_info_ptr. This can be a lot of
information, so it will mostly be best to comment-out this conditional. [LDF 2013.05.19.] */
#endif DEBUG_COMPILE
if (param→PARSER_DEBUG) {
    lock_cerr_mutex();
    cerr ≪ "[Thread_"
    ≪ param→thread_ctr ≪ "]_In_`zzparse',_rule" ≪ endl ≪ "'statement:_"
    temp_str ≪ "_RESPONSE_ZZ_INTEGER_ZZ_" ≪ "user_info_item_list':" ≪ endl;
#endif 0
param→user_info_ptr→show("*param→user_info_ptr:");
#endif
unlock_cerr_mutex();
} /* if (param→PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if ([\$3] ≡ 0 ∧ param→user_info_ptr ≠ 0) */
#endif

```

517.

```

⟨ Get user info action 514 ⟩ +≡
if (param→user_info_ptr) {
    delete param→user_info_ptr;
    param→user_info_ptr = 0;
}
param→PARSER_DEBUG = save_PARSER_DEBUG; } ;

```

518. ⟨user info item list⟩.

Log

[LDF 2013.05.17.] Added these type declarations.

519. ⟨user info item list⟩ → Empty. [LDF 2013.05.17.]

Log

[LDF 2013.05.17.] Added this rule.

```

⟨Rules 11⟩ +≡
[ user_info_item_list : ↳/* ↲Empty ↳*/ ↳]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false;      /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " << 'user_info_item_list: ↳/* ↲Empty ↳*/' . " <<
            endl;
        unlock_cerr_mutex();
    }      /* if (param->PARSER_DEBUG) */
#endif      /* DEBUG_COMPILE */
    if (param->user_info_ptr) param->user_info_ptr->clear();
    else param->user_info_ptr = new User_Info_Type;
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

520. ⟨user info item list⟩ → ⟨user info item list⟩ ⟨user info item⟩. [LDF 2013.05.17.]

Log

[LDF 2013.05.17.] Added this rule.

```

⟨Rules 11⟩ +≡
[ user_info_item_list : ↳user_info_item_list ↳user_info_item ↳]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false;      /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'user_info_item_list: ↳user_info_item_list ↳user_info_item'" . " << endl;
        unlock_cerr_mutex();
    }      /* if (param->PARSER_DEBUG) */
#endif      /* DEBUG_COMPILE */
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

521. ⟨user info item⟩. [LDF 2013.05.17.]

Log

[LDF 2013.05.17.] Added this section.

522. ⟨user info item list⟩ → USER_ID_ZZ UNSIGNED_INTEGER_ZZ. [LDF 2013.05.17.]

Log

[LDF 2013.05.17.] Added this rule.

⟨Rules 11⟩ +≡

```

[ user_info_item : USER_ID_ZZ UNSIGNED_INTEGER_ZZ ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
            << "user_info_item:USER_ID_ZZ_UNSIGNED_INTEGER_ZZ'." << endl
            << "'UNSIGNED_INTEGER_ZZ'" <= < $2 << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->user_info_ptr->user_id = param->user_info_ptr->certificate.user_id = $2;
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

523. <user info item list> → USER_NAME_ZZ STRING_ZZ. [LDF 2013.05.17.]

Log

[LDF 2013.05.17.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
[ user_info_item : USER_NAME_ZZ STRING_ZZ ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'user_info_item:'" << USER_NAME_ZZ << STRING_ZZ ". " << endl << "'STRING_ZZ'" <=
            [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->user_info_ptr->username = param->user_info_ptr->certificate.user_name = [$2];
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

524. <user info item list> → CERTIFICATE_ID_ZZ UNSIGNED_INTEGER_ZZ. [LDF 2013.05.17.]

Log

[LDF 2013.05.17.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
[ user_info_item : CERTIFICATE_ID_ZZ UNSIGNED_INTEGER_ZZ ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'user_info_item:'" << CERTIFICATE_ID_ZZ << UNSIGNED_INTEGER_ZZ ". " << endl <<
            "'UNSIGNED_INTEGER_ZZ'" <= [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->user_info_ptr->certificate.certificate_id = [$2];
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

525. <user info item list> → SERIAL_NUMBER_ZZ UNSIGNED_INTEGER_ZZ. [LDF 2013.05.17.]

Log

[LDF 2013.05.17.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
[ user_info_item : SERIAL_NUMBER_ZZ UNSIGNED_INTEGER_ZZ ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'user_info_item: SERIAL_NUMBER_ZZ UNSIGNED_INTEGER_ZZ'." << endl <<
            "'UNSIGNED_INTEGER_ZZ'" <= <= [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->user_info_ptr->certificate.serialNumber = [$2];
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

526. <user info item list> → ISSUER_CERT_ID_ZZ UNSIGNED_INTEGER_ZZ. [LDF 2013.05.17.]

Log

[LDF 2013.05.17.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
[ user_info_item : ISSUER_CERT_ID_ZZ UNSIGNED_INTEGER_ZZ ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'user_info_item: ISSUER_CERT_ID_ZZ UNSIGNED_INTEGER_ZZ'." << endl <<
            "'UNSIGNED_INTEGER_ZZ'" <= <= [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->user_info_ptr->certificate.issuer_cert_id = [$2];
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

527. <user info item list> → COMMON_NAME_ZZ STRING_ZZ. [LDF 2013.05.17.]

Log

[LDF 2013.05.17.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
[ user_info_item : COMMON_NAME_ZZ STRING_ZZ ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'user_info_item:COMMON_NAME_ZZ STRING_ZZ'." << endl << "'STRING_ZZ'" <=
            [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->user_info_ptr->certificate.commonName = [$2];
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

528. <user info item list> → LOCALITY_NAME_ZZ STRING_ZZ. [LDF 2013.05.19.]

Log

[LDF 2013.05.19.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
[ user_info_item : LOCALITY_NAME_ZZ STRING_ZZ ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'user_info_item:LOCALITY_NAME_ZZ STRING_ZZ'." << endl << "'STRING_ZZ'" <=
            [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->user_info_ptr->certificate.localityName = [$2];
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

529. <user info item list> → STATE_OR_PROVINCE_NAME_ZZ STRING_ZZ. [LDF 2013.05.19.]

Log

[LDF 2013.05.19.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
[ user_info_item:STATE_OR_PROVINCE_NAME_ZZ_STRING_ZZ ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'user_info_item:STATE_OR_PROVINCE_NAME_ZZ_STRING_ZZ'." << endl <<
            "'STRING_ZZ'" <= <= $2 << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->user_info_ptr->certificate.stateOrProvinceName = $2;
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

530. <user info item list> → COUNTRY_NAME_ZZ STRING_ZZ. [LDF 2013.05.19.]

Log

[LDF 2013.05.19.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
[ user_info_item:COUNTRY_NAME_ZZ_STRING_ZZ ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'user_info_item:COUNTRY_NAME_ZZ_STRING_ZZ'." << endl <<
            "'STRING_ZZ'" <= <= $2 << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->user_info_ptr->certificate.countryName = $2;
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

531. <user info item list> → ORGANIZATION_ZZ STRING_ZZ. [LDF 2013.05.19.]

Log

[LDF 2013.05.19.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
[ user_info_item : ORGANIZATION_ZZ STRING_ZZ ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'user_info_item:ORGANIZATION_ZZ STRING_ZZ'." << endl << "'STRING_ZZ'" <<
            [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->user_info_ptr->certificate.organization = [$2];
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

532. <user info item list> → ORGANIZATIONAL_UNIT_NAME_ZZ STRING_ZZ. [LDF 2013.05.19.]

Log

[LDF 2013.05.19.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
[ user_info_item : ORGANIZATIONAL_UNIT_NAME_ZZ STRING_ZZ ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'user_info_item:ORGANIZATIONAL_UNIT_NAME_ZZ STRING_ZZ'." << endl <<
            "'STRING_ZZ'" << [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->user_info_ptr->certificate.organizationalUnitName = [$2];
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

533. ⟨user info item⟩ → IS_CA_ZZ INTEGER_ZZ. [LDF 2013.05.22.]

Log

[LDF 2013.05.22.] Added this rule.

```

⟨Rules 11⟩ +≡
[ user_info_item : IS_CA_ZZ INTEGER_ZZ ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'user_info_item:IS_CA_ZZ INTEGER_ZZ'." << endl << "'INTEGER_ZZ'" <=
            [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->user_info_ptr->certificate.is_ca = static_cast<bool>([$2]);
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

534. ⟨user info item⟩ → IS_PROXY_ZZ INTEGER_ZZ. [LDF 2013.05.22.]

Log

[LDF 2013.05.22.] Added this rule.

```

⟨Rules 11⟩ +≡
[ user_info_item : IS_PROXY_ZZ INTEGER_ZZ ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'user_info_item:IS_PROXY_ZZ INTEGER_ZZ'." << endl << "'INTEGER_ZZ'" <=
            [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->user_info_ptr->certificate.is_proxy = static_cast<bool>([$2]);
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

535. <user info item> → PRIVILEGES_ZZ UNSIGNED_INTEGER_ZZ. [LDF 2013.05.22.]

Log

[LDF 2013.05.22.] Added this rule.

```
<Rules 11> +≡
[ user_info_item :_PRIVILEGES_ZZ_UNSIGNED_INTEGER_ZZ ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'user_info_item:_PRIVILEGES_ZZ_UNSIGNED_INTEGER_ZZ'." << endl <<
            "'UNSIGNED_INTEGER_ZZ'" <= <= [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->user_info_ptr->privileges = [$2];
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

536. <user info item> → IRODS_HOMEDIR_ZZ STRING_ZZ. [LDF 2013.05.22.]

Log

[LDF 2013.05.22.] Added this rule.

```
<Rules 11> +≡
[ user_info_item :_IRODS_HOMEDIR_ZZ_STRING_ZZ ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'user_info_item:_IRODS_HOMEDIR_ZZ_STRING_ZZ'." << endl <<
            "'STRING_ZZ'" <= <= [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->user_info_ptr->irods_homedir = [$2];
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

537. ⟨user info item⟩ → IRODS_CURRENT_DIR_ZZ STRING_ZZ. [LDF 2013.05.22.]

Log

[LDF 2013.05.22.] Added this rule.

```
⟨ Rules 11 ⟩ +≡
[ user_info_item : IRODS_CURRENT_DIR_ZZ STRING_ZZ ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'user_info_item:IRODS_CURRENT_DIR_ZZ STRING_ZZ'." << endl <<
            "'STRING_ZZ'" <= <= [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->user_info_ptr->irods_current_dir = [$2];
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

538. ⟨user info item⟩ → IRODS_ZONE_ZZ STRING_ZZ. [LDF 2013.05.22.]

Log

[LDF 2013.05.22.] Added this rule.

```
⟨ Rules 11 ⟩ +≡
[ user_info_item : IRODS_ZONE_ZZ STRING_ZZ ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'user_info_item:IRODS_ZONE_ZZ STRING_ZZ'." << endl <<
            "'STRING_ZZ'" <= <= [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->user_info_ptr->irods_zone = [$2];
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

539. <user info item> → IRODS_DEFAULT_RESOURCE_ZZ STRING_ZZ. [LDF 2013.05.22.]

Log

[LDF 2013.05.22.] Added this rule.

```
<Rules 11> +≡
[ user_info_item:IRODS_DEFAULT_RESOURCE_ZZ_STRING_ZZ ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'user_info_item:IRODS_DEFAULT_RESOURCE_ZZ_STRING_ZZ'." << endl <<
            "'STRING_ZZ'" <=> [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->user_info_ptr->irods_default_resource = [$2];
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

540. <user info item> → DEFAULT_HANDLE_PREFIX_ZZ STRING_ZZ. [LDF 2013.05.22.]

Log

[LDF 2013.05.22.] Added this rule.

```
<Rules 11> +≡
[ user_info_item:DEFAULT_HANDLE_PREFIX_ZZ_STRING_ZZ ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'user_info_item:DEFAULT_HANDLE_PREFIX_ZZ_STRING_ZZ'." << endl <<
            "'STRING_ZZ'" <=> [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->user_info_ptr->default_handle_prefix = [$2];
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

541. ⟨user info item⟩ → DEFAULT_HANDLE_PREFIX_ID_ZZ UNSIGNED_INTEGER_ZZ. [LDF 2013.05.22.]

Log

[LDF 2013.05.22.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
[ user_info_item : DEFAULT_HANDLE_PREFIX_ID_ZZ UNSIGNED_INTEGER_ZZ ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'user_info_item:DEFAULT_HANDLE_PREFIX_ID_ZZ_UNSIGNED_INTEGER_ZZ' ." << endl <<
            "'UNSIGNED_INTEGER_ZZ'" <= <= [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->user_info_ptr->default_handle_prefix_id = [$2];
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

542. ⟨user info item⟩ → DEFAULT_INSTITUTE_NAME_ZZ STRING_ZZ. [LDF 2013.05.22.]

Log

[LDF 2013.05.22.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
[ user_info_item : DEFAULT_INSTITUTE_NAME_ZZ STRING_ZZ ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'user_info_item:DEFAULT_INSTITUTE_NAME_ZZ_STRING_ZZ' ." << endl <<
            "'STRING_ZZ'" <= <= [$2] << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->user_info_ptr->default_institute_name = [$2];
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

543. <user info item> → DEFAULT_INSTITUTE_ID_ZZ UNSIGNED_INTEGER_ZZ. [LDF 2013.05.22.]

Log

[LDF 2013.05.22.] Added this rule.

```
<Rules 11> +≡
[ user_info_item: DEFAULT_INSTITUTE_ID_ZZ UNSIGNED_INTEGER_ZZ ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'user_info_item: DEFAULT_INSTITUTE_ID_ZZ UNSIGNED_INTEGER_ZZ'." <<
            endl << "'UNSIGNED_INTEGER_ZZ'" <= " " << $2 << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    param->user_info_ptr->default_institute_id = $2;
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

544. ⟨statement⟩ → PROCESS_ZZ PENDING_ZZ OPERATIONS_ZZ RESPONSE_ZZ INTEGER_ZZ. [LDF 2013.05.23.] ■

Log

[LDF 2013.05.23.] Added this rule.

```

⟨Rules 11⟩ +≡
statement : PROCESS_ZZ PENDING_ZZ OPERATIONS_ZZ RESPONSE_ZZ INTEGER_ZZ
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "In 'zzparse', rule 'statement:PROCESS_ZZ PENDING_ZZ OPERATIONS_ZZ' <<
            "RESPONSE_ZZ INTEGER_ZZ' ." << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    lock_cout_mutex();
    lock_cerr_mutex();
    cout << "process pending operations response-->" << endl << "Response code:" << $5 <<
        endl << endl;
    unlock_cerr_mutex();
    unlock_cout_mutex();
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

545. ⟨statement⟩ → CREATE_ZZ HANDLE_ZZ RESPONSE_ZZ INTEGER_ZZ STRING_ZZ. [LDF 2013.05.24.]

Log

[LDF 2013.05.24.] Added this rule.

```

⟨Rules 11⟩ +≡
statement : CREATE_ZZ HANDLE_ZZ RESPONSE_ZZ INTEGER_ZZ STRING_ZZ
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "]"
            << "statement:CREATE_ZZ HANDLE_ZZ RESPONSE_ZZ INTEGER_ZZ STRING_ZZ"
            << endl << "'INTEGER_ZZ'(4)(Response_code)" << "$4" << endl <<
            "'STRING_ZZ'(5)(Handle)=="
            << "$5" << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    lock_cerr_mutex();
    lock_cout_mutex();
    cout << "Create_handle_response-->" << endl << "Response_code:" << "$4" << endl <<
        "Handle:" << "$5" << endl;
    unlock_cerr_mutex();
    unlock_cout_mutex();
    if ($4 ≠ 0) {
        lock_cerr_mutex();
        cerr << "Failed_to_create_handle"
            << "$5" << "."
            << endl << "Will_try_to_continue."
            << endl;
        unlock_cerr_mutex();
        ++param->errors_occurred;
    }
}
;
```

546. $\langle \text{statement} \rangle \rightarrow \text{ADD_ZZ HANDLE_VALUE_ZZ RESPONSE_ZZ INTEGER_ZZ STRING_ZZ INTEGER_ZZ STRING_ZZ}$
 STRING_ZZ . [LDF 2013.06.15.]

Log

[LDF 2013.06.15.] Added this rule.

[LDF 2013.07.03.] Added INTEGER_ZZ for the index, STRING_ZZ for the type of the handle value and STRING_ZZ for the data string or "(binary)", if the latter is binary.

$\langle \text{Rules 11} \rangle + \equiv$

```

statement : ADD_ZZ HANDLE_VALUE_ZZ RESPONSE_ZZ INTEGER_ZZ STRING_ZZ INTEGER_ZZ STRING_ZZ STRING_ZZ
{
    Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type>(
        zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
    string temp_str;
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "statement: ADD_ZZ HANDLE_VALUE_ZZ RESPONSE_ZZ " <<
            "INTEGER_ZZ STRING_ZZ INTEGER_ZZ STRING_ZZ STRING_ZZ" <<
            endl << "' INTEGER_ZZ' (Response_code)" << $4 <<
            endl << "' STRING_ZZ' (Handle)" << $5 <<
            endl << "' INTEGER_ZZ' (idx)" << $6 <<
            endl << "' STRING_ZZ' (type)" << $7 << endl <<
            "' STRING_ZZ' (data)" << $8 << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    lock_cerr_mutex();
    lock_cout_mutex();
    cout << "Add_Handle_Value_Response-->" << endl << "Response_code:" << $4;
    if ($4 == 0) cout << "(Success)";
    cout << endl << "Handle:" << $5 << endl;
    if ($6 > 0) {
        cout << "Index:" << $6;
        temp_str = Handle_Value_Type::idx_type_map[$6];
        if (!temp_str.empty()) cout << "(" << temp_str << ")";
        cout << endl;
    }
    else cout << "Index==0" << endl;
    if (strlen($7) > 0) cout << "Type:" << $7 << endl;
    else cout << "Type: (Empty)" << endl << endl;
    if (strlen($8) > 0) cout << "Data:" << $8 << endl;
    else cout << "Data: (Empty)" << endl;
    if ($4 == 2) {
        cerr << "WARNING! Handle_Value_with_type==" << $6;
        if (!temp_str.empty()) cerr << "(" << temp_str << ")";
    }
}

```

```
    cerr << "already_exists_in_handle" << [$5] << "." << endl << "Not_added..Continuing." <<
        endl << endl;
    ++param->warnings_occurred;
}
else if ([\$4] != 0) {
    cerr << "ERROR! Failed_to_add_handle_value_to_handle" << [$5] << "." << endl <<
        "Will_try_to_continue." << endl << endl;
    ++param->errors_occurred;
}
else cout << endl;
unlock_cerr_mutex();
unlock_cout_mutex();
}
;
```

547. ⟨statement⟩ → DELETE_ZZ HANDLE_VALUE_ZZ RESPONSE_ZZ INTEGER_ZZ STRING_ZZ STRING_ZZ.
[LDF 2013.07.04.]

Log

[LDF 2013.07.04.] Added this rule.

⟨Rules 11⟩ +≡

```

statement : DELETE_ZZ HANDLE_ZZ RESPONSE_ZZ INTEGER_ZZ STRING_ZZ STRING_ZZ
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
    string temp_str;
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " << "statement:DELETE_\
            ZZ_HANDLE_ZZ_RESPONSE_ZZ" << "INTEGER_ZZSTRING_ZZSTRING_ZZ" . " <<
            endl << "'INTEGER_ZZ'"(4)(Response_code)==>" << $4 <<
            endl << "'STRING_ZZ'"(5)(Handle)uuuuuuuu==>" << $5 << endl <<
            "'STRING_ZZ'"(6)(Message)uuuuuuuu==>" << $6 << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    lock_cerr_mutex();
    lock_cout_mutex();
    cout << "Delete_Handle_response-->" << endl << "Response_code:" << $4;
    if ($4 == 0) cout << "(Success)";
    cout << endl << "Handle:uuuuuuuu" << $5 << endl;
    if (strlen($6) > 0) {
        cout << "Message:uuuuuuuu" << $6 << endl;
    }
    else cout << "(No_message)" << endl;
    cout << endl;
    unlock_cerr_mutex();
    unlock_cout_mutex();
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

548. ⟨statement⟩ → UNDELETE_ZZ HANDLE_ZZ RESPONSE_ZZ INTEGER_ZZ STRING_ZZ STRING_ZZ. [LDF 2013.08.21.] ■

Log

[LDF 2013.08.21.] Added this rule.

```

⟨Rules 11⟩ +≡
statement : UNDELETE_ZZ HANDLE_ZZ RESPONSE_ZZ INTEGER_ZZ STRING_ZZ STRING_ZZ
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] In 'zzparse', rule"
            << "statement:UNDELETE_ZZ,HANDLE_ZZ,RESPONSE_ZZ,INTEGER_ZZ,ST\
RING_ZZ,STRING_ZZ:" << endl << "'INTEGER_ZZ' (response_code) ($4) == "
            << $4 << endl << "'STRING_ZZ' (handle) ($5) == " << $5 << endl <<
            "'STRING_ZZ' (message) ($6) == " << $6 << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    lock_cerr_mutex();
    lock_cout_mutex();
    cout << "Undelete_Handle_Response-->" << endl << "Response_code: " << $4;
    if ($4 != 0) cout << "(" << gwstrerror($4) << ")";
    cout << endl << "Handle: " << $5 << endl << "Message: " << $6 << endl << endl;
    unlock_cout_mutex();
    unlock_cerr_mutex();
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

549. ⟨statement⟩ → DELETE_ZZ HANDLE_VALUE_ZZ RESPONSE_ZZ INTEGER_ZZ STRING_ZZ INTEGER_ZZ STRING_ZZ STRING_ZZ INTEGER_ZZ UNSIGNED_LONG_INTEGER_ZZ UNSIGNED_INTEGER_ZZ. [LDF 2013.07.17.]

Log

[LDF 2013.07.17.] Added this rule.

[LDF 2013.09.11.] Added INTEGER_ZZ, UNSIGNED_LONG_INTEGER_ZZ and UNSIGNED_INTEGER_ZZ.

⟨Rules 11⟩ +≡

```

[statement:DELETE_ZZ HANDLE_VALUE_ZZ RESPONSE_ZZ INTEGER_ZZ STRING_ZZ INTEGER_ZZ STRING_ZZ STRI]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "statement:DELETE_ZZ HANDLE_VALUE_ZZ RESPONSE_ZZ " << "INTEGER_ZZ HANDLE_V\
ALUE_SPECIFICATION_ZZ STRING_ZZ " << "INTEGER_ZZ STRING_ZZ ." << endl <<
            "'INTEGER_ZZ'" (Response_code) << endl << $4 << endl <<
            "'STRING_ZZ'" (Handle_value_specifier) << endl << $5 << endl <<
            "'INTEGER_ZZ'" (Index) << endl << $6 << endl <<
            "'STRING_ZZ'" (Type) << endl << $7 << endl <<
            "'STRING_ZZ'" (Data) << endl << $8 << endl <<
            "'STRING_ZZ'" (Message) << endl << $9 << endl <<
            "'INTEGER_ZZ'" ("Immediate") << endl << $10 << endl <<
            "'UNSIGNED_LONG_INTEGER_ZZ'" (Deletion_timestamp) << endl << $11 << endl <<
            "'UNSIGNED_INTEGER_ZZ'" ('purge_database_limit') << endl << $12 << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    string temp_str = gwstrerror($4, true);
    if (!temp_str.empty()) {
        temp_str.insert(0, "(");
        temp_str += ")";
    }
    lock_cerr_mutex();
    lock_cout_mutex();
    cout << "Delete_Handle_Value_Response-->" << endl << "Response_code:" <<
        $4 << temp_str << endl << "Handle_Value_Specifier:" << endl <<
        $5 << endl << "Index:" << endl << $6 << endl <<
        "Type:" << endl << $7 << endl << "Data:" << endl <<
        $8 << endl << "Immediate_deletion_(or_not):" << endl << $10 << endl <<
        "Deletion_timestamp:" << endl << $11;
    if ($11 > 0) cout << " " << convert_seconds($11);
    cout << endl << "'purge_database_limit': " << endl << $12 << endl <<
        "Message:" << endl << $9 << endl << endl;
    unlock_cout_mutex();
}

```

```

unlock_cerr_mutex();
param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

550. <statement> → UNDELETE_ZZ HANDLE_VALUE_ZZ RESPONSE_ZZ INTEGER_ZZ STRING_ZZ INTEGER_ZZ STRING_ZZ STRING_ZZ STRING_ZZ. [LDF 2013.09.11.]

Log

[LDF 2013.09.11.] Added this rule.

```

<Rules 11> +≡
[statement:_UNDELETE_ZZ_HANDLE_VALUE_ZZ_RESPONSE_ZZ_INTEGER_ZZ] [STRING_ZZ_INTEGER_ZZ_STRING_ZZ_STRING_ZZ]
{
    Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread]" << param->thread_ctr << "]"
        << "statement:_UNDELETE_ZZ_HANDLE_VALUE_ZZ_RESPONSE_ZZ"
        << "INTEGER_ZZ_STRING_ZZ_INTEGER_ZZ" << "STRING_ZZ_STRING_ZZ_STRING_ZZ"
        . << endl << "INTEGER_ZZ"(Response_code)==uuuuuuuu" << $4 <<
        endl << "STRING_ZZ"(Handle_value_specifier)== " << $5 <<
        endl << "INTEGER_ZZ"(Index)==uuuuuuuuuuuuuu" << $6 <<
        endl << "STRING_ZZ"(Type)==uuuuuuuuuuuuuu" << $7 <<
        endl << "STRING_ZZ"(Data)==uuuuuuuuuuuuuu" << $8 << endl <<
        "STRING_ZZ"(Message)==uuuuuuuuuuuu" << $9 << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    lock_cout_mutex();
    lock_cerr_mutex();
    cout << "Undelete_Handle_Value_Response-->" << endl << "Response_code:uuuuuuuu" <<
        $4 << endl << "Handle_Value_Specifier:uu" << $5 << endl <<
        "Index:uuuuuuuuuuuu" << $6 << endl << "Type:uuuuuuuuuuuu" << $7 <<
        endl << "Data:uuuuuuuuuuuuuu" << $8 << endl << "Message:uuuuuuuuuuuu" <<
        $9 << endl << endl;
    unlock_cerr_mutex();
    unlock_cout_mutex();
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

551. ⟨statement⟩ → SEND_ZZ TAN_ZZ LIST_ZZ RESPONSE_ZZ INTEGER_ZZ STRING_ZZ. [LDF 2013.06.01.]

Log

[LDF 2013.06.01.] Added this rule.

```

⟨Rules 11⟩ +≡
statement: SEND_ZZ TAN_ZZ LIST_ZZ RESPONSE_ZZ INTEGER_ZZ STRING_ZZ
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
    *>(zzget_extra(parameter));
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'statement:SEND_ZZ,TAN_ZZ,LIST_ZZ,RESPONSE_ZZ,INTEGER_ZZ,STRING_ZZ'." <<
            endl << "'INTEGER_ZZ'(5)(Response_code)==" << $5 << endl <<
            "'STRING_ZZ'(6)(Message)==uuuuuuu" << $6 << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    lock_cerr_mutex();
    lock_cout_mutex();
    cout << "Send,tan,list,response-->" << endl << "Response_code:" << $5 << endl <<
        "Message:uuuuuuuu" << $6 << endl;
    unlock_cerr_mutex();
    unlock_cout_mutex();
    if ($5 == 3) ++param->warnings_occurred;
}
;
```

552. <statement> → MARK_ZZ IRODS_OBJECT_ZZ FOR_ZZ DELETION_ZZ RESPONSE_ZZ INTEGER_ZZ UNSIGNED_INTEGER_STRING_ZZ UNSIGNED_LONG_INTEGER_ZZ STRING_ZZ. [LDF 2013.08.07.]

Log

[LDF 2013.08.07.] Added this rule.
 [LDF 2013.08.08.] Added UNSIGNED_LONG_INTEGER_ZZ. It refers to the timestamp for the “delay” value.
 [LDF 2013.08.21.] Added UNSIGNED_INTEGER_ZZ. It contains the deletion options used, i.e., 0 for archive only, 1 for archive and database or 2 for the database only.

<Rules 11> +≡

```

statement: MARK_ZZ IRODS_OBJECT_ZZ FOR_ZZ DELETION_ZZ RESPONSE_ZZ [INTEGER_ZZ] [UNSIGNED_INTEGER_ZZ]
{
  Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
    *>(zzget_extra(parameter));
  bool save_PARSER_DEBUG = param->PARSER_DEBUG;
  param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
  if (param->PARSER_DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << param->thread_ctr << "] " <<
      "statement: MARK_ZZ IRODS_OBJECT_ZZ FOR_ZZ DELETION_ZZ" <<
      endl << "RESPONSE_ZZ" [INTEGER_ZZ] [UNSIGNED_INTEGER_ZZ] STRING_ZZ" <<
      "UNSIGNED_LONG_INTEGER_ZZ" [STRING_ZZ]. " << endl <<
      "' INTEGER_ZZ' [$6] Response_code == $6 << endl <<
      "' UNSIGNED_INTEGER_ZZ' [$7] Options == $7 << endl <<
      "' STRING_ZZ' [$8] iRODS_object_path == $8 << endl <<
      "' UNSIGNED_LONG_INTEGER_ZZ' [$9] timestamp == $9;
    if ($9 > 0UL) cerr << " == " << convert_seconds($9, true);
    cerr << endl << "' STRING_ZZ' [$10] Message == $10 << endl;
    unlock_cerr_mutex();
  } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
  lock_cout_mutex();
  cout << "Mark_iRODS_object_for_deletion_response-->" << endl <<
    "iRODS_object_path(s): $8 << endl <<
    "Response_code: $6 << endl <<
    "Options: $7 << endl;
  if ($6 != 0) cout << "Error: $6" << gwstrerror($6) << endl;
  else {
    if ($7 & 2U)
      cout << "Marked for deletion from archive and 'gwirdsif' database." << endl;
    else if ($7 & 4U) cout << "Marked for deletion from 'gwirdsif' database." << endl;
    else cout << "Marked for deletion from archive." << endl;
  }
  if ($9 > 0UL)
    cout << "Timestamp(deletion_time): $9 == " << convert_seconds($9,
      true) << endl;
  cout << "Message: $10 << endl << endl;
  unlock_cerr_mutex();
}

```

```

unlock_cout_mutex();
param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

553. ⟨statement⟩ → DELAY_ZZ OPTION_ZZ RESPONSE_ZZ INTEGER_ZZ STRING_ZZ. [LDF 2013.08.08.]

Log

[LDF 2013.08.08.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
statement : □DELAY_ZZ□OPTION_ZZ□RESPONSE_ZZ□INTEGER_ZZ□STRING_ZZ□
{
    Scan_Parse_Parameter_Type *param = static_cast<Scan_Parse_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'statement:□DELAY_ZZ□OPTION_ZZ□RESPONSE_ZZ□INTEGER_ZZ□STRING_ZZ'."
            << endl << "'INTEGER_ZZ'" <= □$4□ << endl << "'STRING_ZZ'" <= □$5□ << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    lock_cout_mutex();
    lock_cerr_mutex();
    cout << "Delay□option□response□-->" << endl << "Code□number:□" << □$4□ << endl <<
        "Message:□□□□□" << □$5□ << endl << endl;
    unlock_cerr_mutex();
    unlock_cout_mutex();
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

554. ⟨statement⟩ → DATABASE_ZZ OPTION_ZZ RESPONSE_ZZ INTEGER_ZZ STRING_ZZ. [LDF 2013.08.12.]

Log

[LDF 2013.08.12.] Added this rule.

```

⟨Rules 11⟩ +≡
statement : DATABASE_ZZ OPTION_ZZ RESPONSE_ZZ INTEGER_ZZ STRING_ZZ
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'statement:DATABASE_ZZOPTION_ZZRESPONSE_ZZINTEGER_ZZSTRING_ZZ'." << endl <<
            "'INTEGER_ZZ'" <= $4 << endl << "'STRING_ZZ'" <= $5 << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    lock_cout_mutex();
    lock_cerr_mutex();
    cout << "Databaseoptionresponse-->" << endl << "Code:number:" <= $4 << endl <<
        "Message:uuuuuu" <= $5 << endl << endl;
    unlock_cerr_mutex();
    unlock_cout_mutex();
    if ($4 == 1) ++param->errors_occurred;
    else if ($4 == 2) ++param->warnings_occurred;
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

555. ⟨statement⟩ → UNDELETE_ZZ RESPONSE_ZZ INTEGER_ZZ UNSIGNED_INTEGER_ZZ STRING_ZZ STRING_ZZ.
[LDF 2013.08.15.]

Log

[LDF 2013.08.15.] Added this rule.
[LDF 2013.08.19.] Added UNSIGNED_INTEGER_ZZ (option counter).

```

⟨Rules 11⟩ +≡
[statement : UNDELETE_ZZ RESPONSE_ZZ INTEGER_ZZ UNSIGNED_INTEGER_ZZ STRING_ZZ STRING_ZZ]
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread_"
            << param->thread_ctr << "] "
            << "statement: UNDELETE\
E_ZZ_RESPONSE_ZZ_INTEGER_ZZ "
            << "UNSIGNED_INTEGER_ZZ_STRING_ZZ_STRING_ZZ . "
            << endl << "INTEGER_ZZ"
            << (response_code) == "
            << $3 << endl <<
            "UNSIGNED_INTEGER_ZZ"
            << (option_counter) == "
            << $4 << endl <<
            "STRING_ZZ"
            << (filename) == "
            << $5 << endl << "STRING_ZZ"
            << (message) == "
            << $6 << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    lock_cout_mutex();
    lock_cerr_mutex();
    cout << "Undelete_response-->" << endl << "Response_code: "
        << $3 << endl <<
        "iRODS_object_name(s) : "
        << $5 << endl;
    if ($3 == 0) {
        if ($4 & 1U) cout << "iRODS_object(s)_undeleted_from_archive" << endl;
        if ($4 & 2U) cout << "iRODS_object(s)_undeleted_from_gwirdsif_database" << endl;
    }
    else {
        if ($4 & 1U) cout << "Failed_to_undelete_iRODS_object(s)_from_archive" << endl;
        if ($4 & 2U)
            cout << "Failed_to_undelete_iRODS_object(s)_from_gwirdsif_database" << endl;
    }
    cout << "Message: "
        << $6 << endl << endl;
    unlock_cerr_mutex();
    unlock_cout_mutex();
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;
```

556. ⟨statement⟩ → DUMMY_STATEMENT_ZZ INTEGER_ZZ. [LDF 2013.04.05.]

Log

[LDF 2013.04.05.] Added this rule. It's used for testing.

```

⟨Rules 11⟩ +≡
statement:DUMMY_STATEMENT_ZZ INTEGER_ZZ
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'statement:DUMMY_STATEMENT_ZZ INTEGER_ZZ'." << endl << "'INTEGER_ZZ'" <<
            "$2" << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    lock_cout_mutex();
    lock_cerr_mutex();
    cout << "Dummy(statement--)" << endl << "Code:number:" << "$2" << endl;
    unlock_cerr_mutex();
    unlock_cout_mutex();
    Response_Type response;
    response.type = Response_Type::COMMAND_ONLY_TYPE;
    temp_strm.str("");
    temp_strm << "DUMMY_STATEMENT_RESPONSE" << "$2";
    response.command = temp_strm.str();
    param->response_deque.push_back(response);
    temp_strm.str("");
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

557. ⟨statement⟩ → DUMMY_STATEMENT_ZZ RESPONSE_ZZ INTEGER_ZZ. [LDF 2013.04.19.]

Log

[LDF 2013.04.19.] Added this rule.

```

⟨Rules 11⟩ +≡
statement : DUMMY_STATEMENT_ZZ RESPONSE_ZZ INTEGER_ZZ
{
    Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
        *>(zzget_extra(parameter));
    bool save_PARSER_DEBUG = param->PARSER_DEBUG;
    param->PARSER_DEBUG = false; /* true save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
    if (param->PARSER_DEBUG) {
        lock_cerr_mutex();
        cerr << "[Thread" << param->thread_ctr << "] " <<
            "'statement:DUMMY_STATEMENT_ZZ|RESPONSE_ZZ|INTEGER_ZZ'." << endl <<
            "'INTEGER_ZZ'" <= < $3 << endl;
        unlock_cerr_mutex();
    } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
    lock_cout_mutex();
    lock_cerr_mutex();
    cout << "Dummy(statement|response--)" << endl << "Response|code:" <= < $3 << endl;
    unlock_cerr_mutex();
    unlock_cout_mutex();
    Response_Type response;
    response.type = Response_Type::COMMAND_ONLY_TYPE;
    temp_strm.str("");
    temp_strm << "DUMMY_STATEMENT|RESPONSE" <= < $3;
    response.command = temp_strm.str();
    param->response_deque.push_back(response);
    temp_strm.str("");
    param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

558. ⟨statement⟩ → DUMMY_STATEMENT_ZZ RESPONSE_ZZ STRING_ZZ. [LDF 2013.08.19.]

Log

[LDF 2013.08.19.] Added this rule.

```

⟨ Rules 11 ⟩ +≡
  statement : DUMMY_STATEMENT_ZZ | RESPONSE_ZZ | STRING_ZZ |
{
  Scan_Parser_Parameter_Type *param = static_cast<Scan_Parser_Parameter_Type
  *>(zzget_extra(parameter));
  bool save_PARSER_DEBUG = param->PARSER_DEBUG;
  param->PARSER_DEBUG = true; /* false save_PARSER_DEBUG */
#ifndef DEBUG_COMPILE
  if (param->PARSER_DEBUG) {
    lock_cerr_mutex();
    cerr << "[Thread" << param->thread_ctr << "] " <<
      "'statement:DUMMY_STATEMENT_ZZ|RESPONSE_ZZ|STRING_ZZ'." << endl <<
      "'STRING_ZZ'" <= < $3 << endl;
    unlock_cerr_mutex();
  } /* if (param->PARSER_DEBUG) */
#endif /* DEBUG_COMPILE */
  lock_cout_mutex();
  lock_cerr_mutex();
  cout << "Dummy" << statement << response-- << endl << "Response" << string : " << endl << $3 << endl;
  unlock_cerr_mutex();
  unlock_cout_mutex();
  param->PARSER_DEBUG = save_PARSER_DEBUG;
}
;

```

559.

⟨ Garbage 256 ⟩ +≡ /* Empty */

560. Putting parser together.

```

%{
  ⟨ Include files 3 ⟩
  using namespace std;
  using namespace gwrdifpk;
  static stringstream temp_strm;
  static int status;
  static char buffer[BUFFER_SIZE];
  ⟨ Declarations of additional functions 125 ⟩
#ifndef 0
  ⟨ Garbage 256 ⟩
#endif
%} ⟨ Options 5 ⟩
⟨ union declaration 127 ⟩
⟨ Token and type declarations 128 ⟩
%% ⟨ Rules 11 ⟩

```

561. Index.

???: 214.

ADD_HANDLE_VALUE_TYPE: 224.

ADD_YY: 18, 128.
 ADD_ZZ: 280, 403, 404.
admin_read: 435, 453.
 ADMIN_READ_ZZ: 300, 403, 406.
admin_write: 435, 454.
 ADMIN_WRITE_ZZ: 301, 403, 406.
 ALL_YY: 102, 128.
 ALL_ZZ: 339, 403.
 ASCII control characters: 117, 370, 392.
 ASTERISK_YY: 11, 129.
 ATTRIBUTE_ZZ: 318, 403, 406.
 AUTHENTICATION_ZZ: 341, 403, 406.
avu: 428.
avu_list: 428.
avu_vector: 426, 431.
back: 426, 504, 505, 506, 507, 508, 509.
begin: 155, 156, 192, 218, 244, 500.
brief: 515.
buffer: 370, 560.
 BUFFER_SIZE: 560.
 BUG FIX: 164, 420.
c_str: 116, 117, 370, 392, 446, 452, 470.
 CD_TYPE: 197, 198.
 CD_YY: 47, 128.
 CD_ZZ: 328, 404.
cerr: 11, 12, 13, 14, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 131, 133, 134, 136, 137, 138, 139, 140, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 153, 154, 155, 156, 159, 160, 161, 162, 163, 164, 165, 166, 168, 169, 171, 172, 174, 175, 177, 178, 179, 180, 181, 182, 183, 184, 185, 187, 188, 189, 190, 191, 192, 194, 195, 196, 197, 198, 199, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 215, 217, 218, 222, 223, 224, 226, 227, 228, 229, 230, 231, 232, 233, 235, 236, 237, 238, 239, 241, 243, 244, 246, 247, 248, 250, 251, 252, 253, 254, 255, 266, 267, 268, 269, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 392, 393, 394, 408, 410, 411, 412, 413, 414, 415, 416, 420, 421, 423, 424, 425, 426, 427, 429, 430, 431, 432, 433, 435, 436, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 472, 473, 474, 475, 476, 477, 478, 480, 481, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 514, 516, 519, 520, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558.
cert_ptr: 478, 480, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497.
certificate: 498, 499, 515, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534.
certificate_id: 483, 524.
 CERTIFICATE_ID_ZZ: 343, 403.
 CERTIFICATE_YY: 99, 128.
 CERTIFICATE_ZZ: 336, 403.
 CERTIFICATES_YY: 100, 128.
 CERTIFICATES_ZZ: 337, 403.
char_value: 403.
 CHMOD_YY: 42, 128.
 CHMOD_ZZ: 404.
 CHOWN_YY: 128.
 CHOWN_ZZ: 404.
clear: 151, 153, 155, 157, 165, 192, 194, 199, 224, 226, 244, 429, 436, 438, 480, 502, 510, 519.
client_finished: 207, 510.
client_sending_file_rule_func: 196.
client_side_filename: 164.
 CLIENT_SIDE_FILENAME_ZZ: 283, 403, 404.
 CLIENT_YY: 95, 128.
command: 136, 138, 145, 146, 160, 164, 165, 185, 191, 199, 208, 209, 212, 213, 217, 218, 247, 252, 253, 254, 465, 556, 557.
 COMMAND_ONLY_TYPE: 136, 138, 145, 146, 160, 164, 165, 185, 191, 199, 208, 209, 212, 213, 217, 218, 247, 248, 252, 253, 254, 556, 557.
 COMMON_NAME_ZZ: 349, 403.
commonName: 138, 491, 527.
 CONNECT_YY: 59, 128.
 CONNECT_ZZ: 405.
 CONNECTION_YY: 61, 128.
 CONNECTION_ZZ: 405.
const_iterator: 155, 156, 192, 244.
const_reverse_iterator: 165, 199.
 control characters, ASCII : 117, 370, 392.
convert_seconds: 435, 549, 552.
convert_time_spec: 66.
 COPY_YY: 39, 128.

COPY_ZZ: 404.
 COUNTRY_NAME_ZZ: 350, 403.
countryName: 492, 530.
cout: 210, 211, 252, 253, 254, 255, 412, 413, 414, 415, 416, 418, 425, 426, 427, 434, 435, 466, 467, 468, 469, 470, 472, 473, 474, 475, 476, 477, 478, 500, 510, 511, 512, 514, 515, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558.
 CREATE_HANDLE_TYPE: 223.
 CREATE YY: 32, 128.
 CREATE_ZZ: 307, 404.
created: 218, 435, 460, 508.
created_by_user_id: 435, 463.
created_by_user_name: 435, 463.
 CREATED_BY_USER_ZZ: 313, 403, 406.
 CREATED_ZZ: 308, 403, 406.
creator_id: 218, 506.
 CREATOR_ID_ZZ: 379, 403, 406.
creator_username: 218, 507.
 CREATOR_USERNAME_ZZ: 380, 403, 406.
data: 435, 444, 446.
 DATA_HEXL_ENCODED_ZZ: 293, 403, 406.
data_length: 435, 445, 446.
 DATA_LENGTH_ZZ: 292, 403, 406.
data_str: 232.
 DATA YY: 89, 128.
 DATA_ZZ: 291, 403, 406.
 DATABASE_ONLY YY: 111, 128.
 DATABASE YY: 110, 128.
 DATABASE_ZZ: 386, 403, 406.
days: 253.
 DEBUG_COMPILE: 11, 12, 13, 14, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 131, 133, 134, 136, 137, 138, 139, 140, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 153, 154, 155, 156, 159, 160, 161, 162, 163, 165, 166, 168, 169, 171, 172, 174, 175, 177, 178, 179, 180, 181, 182, 183, 184, 187, 188, 189, 190, 191, 192, 194, 195, 196, 197, 198, 199, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 215, 217, 218, 222, 223, 224, 226, 227, 228, 229, 230, 231, 232, 233, 235, 236, 237, 238, 239, 241, 243, 244, 246, 247, 248, 250, 251, 252, 253, 254, 255, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 408, 410, 411, 412, 413, 414, 415, 416, 420, 421, 423, 424, 425, 426, 427, 429, 430, 431, 432, 433, 435, 436, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 472, 473, 474, 475, 476, 477, 478, 480, 481, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 514, 516, 519, 520, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558.
dec: 156, 163, 233, 236, 237, 241, 244, 427, 509, 512.
default_handle_prefix: 223, 540.
default_handle_prefix_id: 541.
 DEFAULT_HANDLE_PREFIX_ID_ZZ: 365, 403, 406.
 DEFAULT_HANDLE_PREFIX_ZZ: 364, 403, 406.
default_institute_id: 543.
 DEFAULT_INSTITUTE_ID_ZZ: 367, 403, 406.
default_institute_name: 223, 542.
 DEFAULT_INSTITUTE_NAME_ZZ: 366, 403, 406.
delay: 66.
delay_value: 142, 144, 145, 146, 147, 148, 149, 156, 157, 159, 233, 237, 238, 241.
 DELAY YY: 108, 128.
 DELAY_ZZ: 387, 403, 406.
delayed_response_deque: 139, 140, 157, 199.
delete_from_database_timestamp: 435, 462.
 DELETE_FROM_DATABASE_TIMESTAMP_ZZ: 312, 403, 406.
 DELETE_HANDLE_TYPE: 233.
 DELETE_HANDLE_VALUE_TYPE: 241.
 DELETE YY: 33, 128.
 DELETE_ZZ: 381, 404.
 DELETED_ZZ: 309, 403, 406.
 DELETION YY: 35, 128.
 DELETION_ZZ: 383, 403, 404.
dest: 446, 452.
dest_length: 446, 452.
 DIRECTORY YY: 28, 128.
 DIRECTORY_ZZ: 404.
dirname: 197, 198.
 DISCONNECT YY: 60, 128.
 DISCONNECT_ZZ: 405.

distinguished_name: 515.
distinguished_name_rule_func: 137.
DISTINGUISHED_NAME YY: 17, 128.
DISTINGUISHED_NAME ZZ: 340, 405.
DOLLAR YY: 13, 129.
DUMMY_STATEMENT YY: 114, 128.
DUMMY_STATEMENT ZZ: 390, 403, 404.
empty: 223, 416, 466, 478, 546, 549.
end: 155, 156, 192, 210, 211, 212, 213, 218, 244, 500.
END_PUT_RULE: 164, 165.
end_server_enabled: 136.
END_SERVER_TYPE: 136.
END_SERVER YY: 74, 128.
END_SERVER ZZ: 332, 403.
END YY: 73, 115, 128.
END ZZ: 270, 391, 403.
endl: 11, 12, 13, 14, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 131, 133, 134, 136, 137, 138, 139, 140, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 153, 154, 155, 156, 159, 160, 161, 162, 163, 164, 165, 166, 168, 169, 171, 172, 174, 175, 177, 178, 179, 180, 181, 182, 183, 184, 185, 187, 188, 189, 190, 191, 192, 194, 195, 196, 197, 198, 199, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 215, 217, 218, 222, 223, 224, 226, 227, 228, 229, 230, 231, 232, 233, 235, 236, 237, 238, 239, 241, 243, 244, 246, 247, 248, 250, 251, 252, 253, 254, 255, 266, 267, 268, 269, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 392, 393, 394, 408, 410, 411, 412, 413, 414, 415, 416, 418, 420, 421, 423, 424, 425, 426, 427, 429, 430, 431, 432, 433, 434, 435, 436, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 472, 473, 474, 475, 476, 477, 478, 480, 481, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 514, 515, 516, 519, 520, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558.
EQUALS YY: 14, 128.
erase: 117, 370, 392.
errno: 470.
ERROR ZZ: 342, 403, 406.
errors_occurred: 137, 139, 140, 164, 210, 211, 212, 213, 218, 416, 434, 436, 446, 452, 466, 468, 469, 476, 478, 500, 545, 546, 554.
ETX (ASCII control character): 117, 392.
FAILED ZZ: 315, 403, 406.
false: 7, 66, 67, 137, 138, 139, 140, 142, 143, 144, 145, 146, 147, 148, 149, 151, 153, 154, 156, 157, 159, 160, 161, 162, 163, 191, 199, 208, 209, 210, 211, 212, 213, 215, 217, 218, 222, 223, 224, 226, 227, 228, 229, 230, 231, 232, 233, 235, 236, 237, 238, 239, 241, 243, 244, 246, 247, 248, 250, 251, 252, 253, 254, 255, 261, 389, 414, 416, 425, 426, 427, 433, 444, 445, 446, 469, 475, 477, 478, 480, 481, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 494, 495, 496, 497, 498, 499, 500, 502, 503, 504, 505, 506, 507, 508, 509, 510, 512, 514, 515, 519, 520, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 552, 553, 554, 555, 556, 557, 558.
field_width: 435.
FILE YY: 25, 128.
FILE ZZ: 282, 404, 416.
filename: 435, 440.
filename_vector: 155, 156, 157, 192, 194, 195, 244.
find: 117, 210, 211, 212, 213, 392.
FINISHED YY: 96, 128.
FINISHED ZZ: 331, 403, 406.
first: 165, 199, 210, 212, 213, 218.
FLAG YY: 116, 128.
FLAG ZZ: 403, 404.
flags: 151, 155, 156, 165, 166.
float_value: 127, 402.
flush: 475.
FOR YY: 101, 128.
FOR ZZ: 338, 403.
FORCE_ADD YY: 20, 128.
FORCE_ALL YY: 23, 128.
FORCE_PUT YY: 21, 128.
FORCE_STORE YY: 22, 128.
FORCE YY: 19, 128.
fwrite: 435, 446, 452.
gcry_control: 3, 124, 259, 399.

GENERATE_YY: 90, 91, 128.
get_all_groups: 218.
get_handle: 434.
GET_HANDLE_TYPE: 189, 190.
get_metadata: 185.
GET_METADATA_TYPE: 184.
GET_TYPE: 166.
get_user_info_func: 139, 140.
GET_USER_INFO_TYPE: 139, 140.
GET_USER_INFO_YY: 103, 128.
GET_USER_INFO_ZZ: 369, 403, 514.
GET_USER_ZZ: 368, 403.
GET_YY: 54, 128.
GET_ZZ: 273, 405.
group_id: 218, 504.
GROUP_ID_ZZ: 377, 403, 406.
group_item: 501.
group_item_list: 501.
group_name: 218, 505.
GROUP_NAME_ZZ: 378, 403, 406.
Group_Type: 218, 500, 502, 504.
group_vector: 218, 500, 502, 504, 505, 506, 507, 508, 509.
GROUP_YY: 112, 128.
GROUP_ZZ: 375, 403, 406.
GROUPS_YY: 113, 128.
GROUPS_ZZ: 376, 403, 406.
gwrdifpk: 122, 257, 397, 560.
gwstrerror: 548, 549, 552.
handle: 434, 435, 441.
handle_id: 435, 457.
HANDLE_ID_ZZ: 304, 403, 406.
handle_item: 437.
handle_item_list: 437.
handle_value: 433, 434, 435, 436, 438, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464.
handle_value_id: 435, 458.
HANDLE_VALUE_ID_ZZ: 305, 403, 406.
HANDLE_VALUE_SPECIFICATION_YY: 67, 128.
HANDLE_VALUE_SPECIFICATION_ZZ: 389, 403.
Handle_Value_Type: 433, 546.
HANDLE_VALUE_YY: 85, 128.
HANDLE_VALUE_ZZ: 287, 403, 405.
HANDLE_VALUES_YY: 86, 128.
HANDLE_VALUES_ZZ: 288, 403, 405.
HANDLE_YY: 83, 128.
HANDLE_ZZ: 285, 403, 405.
HANDLES_YY: 84, 128.
HANDLES_ZZ: 286, 403, 405.
HAVE_CONFIG_H: 3, 124, 259, 399.
hex: 233, 236, 237, 427.
hexl_decode: 446, 452.
HEXL_ENCODED_STRING_ZZ: 370, 403, 405.
hours: 253.
hvt: 224, 226, 230, 231, 232.
icommands: 466.
icommands_flag_str: 169.
idx: 230, 435, 442.
idx_type_map: 546.
IDX_YY: 87, 128.
IDX_ZZ: 289, 403, 406.
IMMEDIATE_YY: 36, 128.
INFO_YY: 45, 128.
INFO_ZZ: 404.
insert: 509, 549.
INSTITUTE_YY: 92, 128.
int_val: 184, 189, 190, 208, 215, 217.
int_value: 68, 127, 128, 129, 141, 267, 369, 371, 402, 403, 404, 405, 406, 428, 437.
INTEGER_YY: 68, 128.
INTEGER_ZZ: 267, 403.
irods_avu: 431.
Irods_AVU_Type: 431.
irods_current_dir: 466, 537.
IRODS_CURRENT_DIR_ZZ: 359, 403, 406.
irods_default_resource: 539.
IRODS_DEFAULT_RESOURCE_ZZ: 362, 403, 406.
irods_homedir: 536.
IRODS_HOME_DIR_ZZ: 360, 403, 406.
irods_object: 426, 429, 431.
irods_object_id: 435, 459.
IRODS_OBJECT_ID_ZZ: 306, 403, 406.
Irods_Object_Type: 429.
irods_object_vector: 426.
IRODS_OBJECT_ZZ: 363, 403, 406.
irods_zone: 538.
IRODS_ZONE_ZZ: 361, 403, 406.
is_ca: 487, 533.
IS_CA_ZZ: 356, 403.
is_proxy: 488, 534.
IS_PROXY_ZZ: 357, 403.
issuer_cert_id: 486, 526.
ISSUER_CERT_ID_ZZ: 345, 403.
iter: 155, 156, 165, 192, 199, 210, 211, 212, 213, 218, 244, 500.
iter_1: 218.
iterator: 210, 211, 212, 213, 218, 500.
jargon_core: 150.
jargon_trunk: 150.
last_modified: 435, 461.
LAST_MODIFIED_ZZ: 311, 403, 406.
left: 435.

length: 370.
LIST_YY: 80, 128.
LIST_ZZ: 278, 405.
local_filename: 165, 166, 184, 190, 199, 416, 465.
LOCAL_FILENAME_YY: 26, 128.
LOCAL_FILENAME_ZZ: 323, 403, 404.
LOCAL_HOST: 3, 124, 259, 399.
LOCAL_YY: 29, 128.
LOCAL_ZZ: 404.
LOCALITY_NAME_ZZ: 351, 403.
localityName: 493, 528.
lock_cerr_mutex: 11, 12, 13, 14, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 131, 133, 134, 136, 137, 138, 139, 140, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 153, 154, 155, 156, 159, 160, 161, 162, 163, 164, 165, 166, 168, 169, 171, 172, 174, 175, 177, 178, 179, 180, 181, 182, 183, 184, 185, 187, 188, 189, 190, 191, 192, 194, 195, 196, 197, 198, 199, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 215, 217, 218, 222, 223, 224, 226, 227, 228, 229, 230, 231, 232, 233, 235, 236, 237, 238, 239, 241, 243, 244, 246, 247, 248, 250, 251, 252, 253, 254, 255, 266, 267, 268, 269, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 392, 393, 394, 408, 410, 411, 412, 413, 414, 415, 416, 418, 420, 421, 423, 424, 425, 426, 427, 429, 430, 431, 432, 433, 434, 435, 436, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 472, 473, 474, 475, 476, 477, 478, 480, 481, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 514, 516, 519, 520, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558.

lock_cout_mutex: 210, 211, 252, 253, 254, 255, 412, 413, 414, 415, 416, 418, 425, 426, 427, 434, 435, 466, 467, 468, 469, 470, 472, 473, 474, 475, 476, 477, 478, 500, 510, 511, 512, 514, 516, 519, 520, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558.

LS_TYPE: 151.
LS_YY: 57, 128.
LS_ZZ: 272, 405.
lsUtil: 151.
lvalp: 125, 400.
make_pair: 509.
map: 165, 199, 210, 212, 218.
MARK_IRODS_OBJECTS_FOR_DELETION_TYPE: 157.
MARK_YY: 37, 128.
MARK_ZZ: 384, 403, 404.
marked_for_deletion: 435, 464.
MARKED_FOR_DELETION_ZZ: 310, 403, 406.
member_id_map: 218, 509.
memcpy: 370, 444, 446, 452.
memset: 370, 444, 446, 450, 452.
metadata_options: 199.
METADATA_YY: 48, 128.
METADATA_ZZ: 317, 405.
minutes: 253.
MKDIR_TYPE: 155.
MKDIR_YY: 75, 128.
MKDIR_ZZ: 329, 404.
MOVE_YY: 40, 128.
MOVE_ZZ: 404.
multimap: 211, 213.
mysql_ptr: 218, 436.
NAME_LEN: 3, 124, 259, 399.
no_delay: 151, 157, 241.
NO_DELAY_YY: 109, 128.
NOTE: 172, 175, 180, 466.
npos: 117, 392.
oct: 156, 163, 241, 244, 509, 512.
OPERATIONS_YY: 107, 128.
OPERATIONS_ZZ: 374, 403.
OPTION_ZZ: 388, 403, 406.
options: 157, 241, 244.
organization: 489, 531.
ORGANIZATION_ZZ: 347, 403.
ORGANIZATIONAL_UNIT_NAME_ZZ: 348, 403.
organizationalUnitName: 490, 532.
OUTPUT_YY: 49, 128.
OUTPUT_ZZ: 403, 405.
OVERWRITE_ZZ: 284, 403, 404.
pair: 218.
param: 66, 131, 133, 134, 136, 137, 138, 139, 140, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 153, 154, 155, 156, 157, 159, 160, 161, 162, 163.

164, 165, 166, 168, 169, 171, 172, 174, 175, 177, 178, 179, 180, 181, 182, 183, 184, 185, 187, 188, 189, 190, 191, 192, 194, 195, 196, 197, 198, 199, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 215, 217, 218, 222, 223, 224, 226, 227, 228, 229, 230, 231, 232, 233, 235, 236, 237, 238, 239, 241, 243, 244, 246, 247, 248, 250, 251, 252, 253, 254, 255, 408, 410, 411, 412, 413, 414, 415, 416, 418, 420, 421, 423, 424, 425, 426, 427, 429, 430, 431, 432, 433, 434, 435, 436, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 480, 481, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 514, 515, 516, 517, 519, 520, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558.

parameter: 120, 125, 131, 133, 134, 136, 137, 138, 139, 140, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 153, 154, 155, 156, 159, 160, 161, 162, 163, 166, 168, 169, 171, 172, 174, 175, 177, 178, 179, 180, 181, 182, 183, 184, 185, 187, 188, 189, 190, 191, 192, 194, 195, 196, 197, 198, 199, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 215, 217, 218, 222, 223, 224, 226, 227, 228, 229, 230, 231, 232, 233, 235, 236, 237, 238, 239, 241, 243, 244, 246, 247, 248, 250, 251, 252, 253, 254, 255, 395, 400, 408, 410, 411, 412, 413, 414, 415, 416, 420, 421, 423, 424, 425, 426, 427, 429, 430, 431, 432, 433, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 472, 473, 474, 475, 476, 477, 478, 480, 481, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 514, 519, 520, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558.

PARSER_DEBUG: 131, 133, 134, 136, 137, 138, 139, 140, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 153, 154, 155, 156, 157, 159, 160, 161, 162, 163, 165, 166, 168, 169, 171, 172, 174, 175, 177, 178, 179, 180, 181, 182, 183, 184, 187, 188, 189, 190, 191, 192, 194, 195, 196, 197, 198, 199, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 215, 217, 218, 222, 223, 224, 226, 227, 228, 229, 230, 231, 232, 233, 235, 236, 237, 238, 239, 241, 243, 244, 246, 247, 248, 250, 251, 252, 253, 254, 255, 408, 410, 411, 412, 413, 414, 415, 416, 420, 421, 423, 424, 425, 426, 427, 429, 430, 431, 432, 433, 434, 435, 436, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 480, 481, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 514, 519, 520, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558.

PASSWORD_ZZ: 405.

path: 426.

PENDING_ZZ: 373, 403.

pid_institute_str: 165, 177, 181, 223, 226.

pid_options: 165, 233.

pid_prefix_str: 165, 177, 183, 223, 226.

pid_str: 165, 177, 179, 189, 223, 224, 226, 228, 229, 233, 239.

pid_suffix_str: 165, 177, 182, 223, 226.

PID_ZZ: 325, 406.

PIDS_ZZ: 326, 403, 406.

pointer_value: 127, 370, 402, 403, 405, 446.

pos: 117, 392.

PREFIX_ZZ: 94, 128.

privileges: 217, 218, 512, 535.

PRIVILEGES_ZZ: 358, 403, 406.

process_pending_response: 156, 157.

PROCESS_PENDING_TYPE: 156, 165, 166, 184, 185, 189, 190, 199, 222, 223, 239, 241, 243, 244.

PROCESS_ZZ: 105, 128.

PROCESS_ZZ: 372, 403.

profiling: 136.

pthread_kill: 210, 211, 477.

pthread_mutex_lock: 165, 199.

pthread_mutex_unlock: 165, 199.

pthread_self: 477.

pub_read: 435, 455.

PUB_READ_ZZ: 302, 403, 406.

pub_write: 435, 456.

PUB_WRITE_ZZ: 303, 403, 406.

push_back: 136, 138, 139, 140, 145, 146, 150, 151, 154, 155, 157, 160, 164, 165, 166, 184, 185, 189, 190, 191, 194, 195, 197, 198, 199, 208, 209, 212, 213, 215, 217, 218, 222, 223, 224, 233, 239, 241, 243, 244, 247, 252, 253, 254, 426, 431, 465, 502, 504, 556, 557.
put: 163.
PUT_YY: 55, 128.
PUT_ZZ: 274, 403, 405.
PWD_TYPE: 150.
PWD YY: 46, 128.
PWD ZZ: 271, 404.
rbegin: 165, 199.
READ YY: 64, 128.
READ ZZ: 405.
receive_file: 192, 416, 469.
RECEIVE_METADATA_FILE_TYPE: 199.
RECEIVE_PUT_FILE_TYPE: 165.
RECEIVE_ZZ: 275, 403, 406.
ref_ctr: 165, 199.
REFERENCE YY: 78, 128.
REFERENCE ZZ: 327, 403, 404.
refs: 435, 450, 452.
REFS_HEXL_ENCODED_ZZ: 299, 403, 406.
refs_length: 435, 451, 452.
REFS_LENGTH_ZZ: 298, 403, 406.
REFS_ZZ: 297, 403, 406.
remote_filename: 165, 166, 199.
REMOTE_FILENAME YY: 27, 128.
REMOTE_FILENAME ZZ: 324, 403, 404.
REMOTE YY: 30, 128.
REMOTE ZZ: 404.
RENAME YY: 41, 128.
RENAME ZZ: 404.
replace: 117, 392.
REPOSITORY YY: 58, 128.
REPOSITORY ZZ: 405.
RESET YY: 62, 128.
RESET ZZ: 405.
response: 136, 138, 139, 140, 145, 146, 150, 151, 155, 156, 157, 160, 163, 164, 165, 166, 184, 185, 189, 190, 191, 196, 197, 198, 199, 208, 209, 212, 213, 215, 217, 218, 222, 223, 224, 233, 239, 241, 243, 244, 247, 248, 252, 253, 254, 465, 556, 557.
response_deque: 136, 138, 145, 146, 150, 151, 155, 157, 160, 164, 165, 166, 184, 185, 189, 190, 191, 197, 198, 208, 209, 212, 213, 215, 217, 218, 222, 223, 224, 233, 239, 241, 243, 244, 247, 252, 253, 254, 465, 510, 556, 557.
response_map: 163, 165, 196, 199.
response_map_mutex: 165, 199.

Response_Type: 136, 138, 139, 140, 145, 146, 150, 151, 155, 156, 157, 160, 163, 164, 165, 166, 184, 185, 189, 190, 191, 197, 198, 199, 208, 209, 212, 213, 215, 217, 218, 222, 223, 224, 233, 239, 241, 243, 244, 247, 248, 252, 253, 254, 255, 465, 466, 467, 468, 469, 471, 472, 473, 474, 475, 476, 477, 478, 480, 481, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 502, 503, 504, 505, 506, 507, 508, 509, 510, 512, 514, 517, 519, 520, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558.
save_PARSER_DEBUG: 137, 138, 139, 140, 142, 143, 144, 145, 146, 147, 148, 149, 151, 153, 154, 155, 156, 157, 159, 160, 161, 162, 163, 165, 191, 199, 208, 209, 210, 211, 212, 213, 215, 217, 218, 222, 223, 224, 226, 227, 228, 229, 230, 231, 232, 233, 235, 236, 237, 238, 239, 241, 243, 244, 246, 247, 248, 250, 251, 252, 253, 254, 255, 414, 416, 418, 425, 426, 427, 433, 436, 444, 445, 446, 469, 471, 475, 477, 478, 480, 481, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 502, 503, 504, 505, 506, 507, 508, 509, 510, 512, 514, 517, 519, 520, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558.
save_scanner_debug: 66, 67, 389.
Scan_Parse_Parameter_Type: 131, 133, 134, 136, 137, 138, 139, 140, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 153, 154, 155, 156, 159, 160, 161, 162, 163, 166, 168, 169, 171, 172, 174, 175, 177, 178, 179, 180, 181, 182, 183, 184, 185, 187, 188, 189, 190, 191, 192, 194, 195, 196, 197, 198, 199, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 215, 217, 218, 222, 223, 224, 226, 227, 228, 229, 230, 231, 232, 233, 235, 236, 237, 238, 239, 241, 243, 244, 246, 247, 248, 250, 251, 252, 253, 254, 255, 408, 410, 411, 412, 413, 414, 415, 416, 420, 421, 423, 424, 425, 426, 427, 429, 430, 431, 432, 433, 434, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 472, 473, 474, 475, 476, 477, 478, 480, 481, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 514, 519, 520, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558.

SCANNER_DEBUG: 7, 11, 12, 13, 14, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 261, 266, 267, 268, 269, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 392, 393.
second: 165, 199, 210, 211, 212, 213, 218.
seconds: 253.
SEND_FILE_TYPE: 465.
SEND_TAN_LIST_TYPE: 191.
SEND YY: 76, 128.
SEND ZZ: 276, 405.
SENDING YY: 77, 128.
SENDING ZZ: 279, 403, 405.
SERIAL_NUMBER_ZZ: 353, 403.
serialNumber: 138, 495, 525.
server_finished: 432, 510.
SERVER YY: 71, 128.
SERVER ZZ: 322, 403, 406.
SESSION YY: 56, 128.
SESSION ZZ: 405.
SET YY: 53, 128.
SET ZZ: 405.
setw: 435.
show: 138, 165, 199, 218, 224, 426, 435, 465, 478, 500, 515, 516.
SHOW_CERTIFICATE_TYPE: 215, 217.
SHOW_CERTIFICATES_PRIVILEGE: 217.
SHOW_GROUPS_PRIVILEGE: 218.
show_privileges: 512.
SHOW YY: 43, 128.
SHOW ZZ: 335, 404.
signal_client_enabled: 212, 213, 477.
signal_name_map: 210, 211, 212, 477.
signal_number_map: 211, 213.
signal_server_enabled: 210, 211.
SIGNAL YY: 98, 128.
SIGNAL ZZ: 334, 403.
size: 117, 157, 164, 165, 199, 218, 392, 426, 466, 500.
size_type: 117, 392.
sleep: 475.

sleep_client_enabled: 209, 475.
sleep_server_enabled: 208.
SLEEP_TYPE: 208.
SLEEP YY: 97, 128.
SLEEP ZZ: 333, 403, 404.
sscanf: 68, 69, 70, 267, 268, 269.
STATE_OR_PROVINCE_NAME_ZZ: 352, 403.
stateOrProvinceName: 494, 529.
status: 137, 139, 140, 192, 196, 218, 257, 416, 436, 446, 452, 469, 470, 475, 560.
std: 122, 257, 397, 560.
stderr: 446, 452.
stdout: 435.
STORE YY: 24, 128.
STORE ZZ: 281, 403, 404.
str: 138, 145, 146, 160, 164, 165, 185, 191, 199, 208, 209, 212, 213, 218, 247, 252, 253, 254, 426, 465, 470, 478, 500, 515, 556, 557.
strcat: 169.
strcpy: 67, 116, 117, 118, 168, 169, 171, 172, 174, 175, 389, 392, 393, 420, 421, 450.
strerror: 470.
string: 66, 116, 117, 154, 155, 156, 164, 192, 210, 211, 212, 213, 218, 244, 370, 392, 412, 416, 446, 452, 466, 469, 514, 546, 547, 549.
string_val: 140, 241, 243.
string_value: 67, 116, 117, 118, 127, 128, 389, 392, 393, 402, 403, 404, 405, 446.
string_vector: 151, 153, 154, 155, 156, 223, 244.
STRING YY: 117, 118, 128.
STRING ZZ: 392, 393, 405.
stringstream: 257, 426, 560.
strlen: 165, 169, 370, 416, 435, 444, 450, 467, 468, 472, 473, 546, 547.
STX (ASCII control character): 117, 392.
SU YY: 52, 128.
SU ZZ: 405.
SUCCEEDED ZZ: 316, 403, 406.
SUFFIX YY: 93, 128.
system: 470.
TAN YY: 79, 128.
TAN ZZ: 277, 405.
temp_filename: 416, 469, 470.
temp_str: 66, 116, 117, 392, 412, 446, 452, 466, 514, 516, 546, 547, 549.
temp_str_1: 514.
temp_strm: 138, 145, 146, 160, 164, 165, 185, 191, 199, 208, 209, 212, 213, 218, 247, 252, 253, 254, 257, 426, 465, 470, 478, 500, 515, 556, 557, 560.
thread_ctr: 131, 133, 134, 136, 137, 138, 139, 140, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 153, 154, 155, 156, 159, 160, 161, 162, 163, 164,

165, 166, 168, 169, 171, 172, 174, 175, 177, 178,
 179, 180, 181, 182, 183, 184, 185, 187, 188, 189,
 190, 191, 192, 194, 195, 196, 197, 198, 199, 201,
 202, 203, 204, 205, 206, 207, 208, 209, 210, 211,
 212, 213, 215, 217, 218, 222, 223, 224, 226, 227,
 228, 229, 230, 231, 232, 233, 235, 236, 237, 238,
 239, 241, 243, 244, 246, 247, 248, 250, 251, 252,
 253, 254, 255, 408, 410, 411, 412, 413, 414, 415,
 416, 420, 421, 423, 424, 425, 426, 427, 429, 430,
 431, 432, 433, 436, 438, 439, 440, 441, 442, 443,
 444, 445, 446, 447, 448, 449, 450, 451, 452, 453,
 454, 455, 456, 457, 458, 459, 460, 461, 462, 463,
 464, 465, 466, 467, 468, 469, 470, 472, 473, 474,
 475, 476, 477, 478, 480, 481, 483, 484, 485, 486,
 487, 488, 489, 490, 491, 492, 493, 494, 495, 496,
 497, 498, 499, 500, 502, 503, 504, 505, 506, 507,
 508, 509, 510, 511, 512, 514, 516, 519, 520, 522,
 523, 524, 525, 526, 527, 528, 529, 530, 531, 532,
 533, 534, 535, 536, 537, 538, 539, 540, 541, 542,
 543, 544, 545, 546, 547, 548, 549, 550, 551,
 552, 553, 554, 555, 556, 557, 558.
thread_ctr_id_map: 210, 211.
TILDE YY: 12, 129.
TIME_SET_ZZ: 321, 403, 406.
TIME_SPECIFICATION YY: 66, 128.
timestamp: 435, 449.
TIMESTAMP_ZZ: 296, 403, 406.
TO YY: 65, 128.
TO ZZ: 405.
TODO: 164, 191.
token: 128, 129, 403, 404, 405, 406.
true: 7, 66, 67, 137, 138, 139, 140, 142, 143, 144,
 145, 146, 147, 148, 149, 150, 151, 153, 154, 156,
 159, 160, 161, 162, 163, 191, 199, 207, 208, 209,
 210, 211, 212, 213, 215, 217, 218, 222, 223, 224,
 226, 227, 228, 229, 230, 231, 232, 233, 235, 236,
 237, 238, 239, 241, 243, 244, 246, 247, 248, 250,
 251, 252, 253, 254, 255, 261, 389, 414, 416, 425,
 426, 427, 432, 433, 435, 436, 444, 445, 446, 466,
 469, 475, 477, 478, 480, 481, 483, 484, 485, 486,
 487, 488, 489, 490, 491, 492, 493, 494, 495, 496,
 497, 498, 499, 500, 502, 503, 504, 505, 506, 507,
 508, 509, 510, 512, 514, 515, 519, 520, 522, 523,
 524, 525, 526, 527, 528, 529, 530, 531, 532,
 533, 534, 535, 536, 537, 538, 539, 540, 541,
 542, 543, 544, 545, 546, 547, 548, 549, 550,
 552, 553, 554, 555, 556, 557, 558.
ttl: 435, 448.
ttl_type: 435, 447.
TTL_TYPE_ZZ: 294, 403, 406.
TTL_ZZ: 295, 403, 406.
type: 136, 138, 139, 140, 145, 146, 150, 151, 155,
 156, 157, 160, 164, 165, 166, 184, 185, 189,
 190, 191, 197, 198, 199, 208, 209, 212, 213,
 215, 217, 218, 222, 223, 224, 231, 233, 239,
 241, 243, 244, 247, 248, 252, 253, 254, 428,
 435, 437, 443, 465, 556, 557.
TYPE YY: 88, 128.
TYPE ZZ: 290, 403, 406.
uint_value: 69, 127, 128, 268, 402, 403.
ulint_value: 66, 70, 127, 128, 141, 269, 402.
ULONG_MAX: 66, 147, 148.
UNDELETE_FILE_TYPE: 244.
UNDELETE_HANDLE_TYPE: 239.
UNDELETE_HANDLE_VALUE_TYPE: 243.
UNDELETE YY: 34, 128.
UNDELETE ZZ: 382, 403, 404.
UNITS ZZ: 320, 403, 406.
unlock_cerr_mutex: 11, 12, 13, 14, 16, 17, 18, 19,
 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32,
 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45,
 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58,
 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71,
 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97,
 98, 99, 100, 101, 102, 103, 104, 105, 106, 107,
 108, 109, 110, 111, 112, 113, 114, 115, 116, 117,
 118, 119, 131, 133, 134, 136, 137, 138, 139, 140,
 142, 143, 144, 145, 146, 147, 148, 149, 150, 151,
 153, 154, 155, 156, 159, 160, 161, 162, 163, 164,
 165, 166, 168, 169, 171, 172, 174, 175, 177, 178,
 179, 180, 181, 182, 183, 184, 185, 187, 188, 189,
 190, 191, 192, 194, 195, 196, 197, 198, 199, 201,
 202, 203, 204, 205, 206, 207, 208, 209, 210, 211,
 212, 213, 215, 217, 218, 222, 223, 224, 226, 227,
 228, 229, 230, 231, 232, 233, 235, 236, 237, 238,
 239, 241, 243, 244, 246, 247, 248, 250, 251, 252,
 253, 254, 255, 266, 267, 268, 269, 332, 333, 334,
 335, 336, 337, 338, 339, 340, 341, 342, 343, 344,
 345, 346, 347, 348, 349, 350, 351, 352, 353, 354,
 355, 356, 357, 358, 359, 360, 361, 362, 363, 364,
 365, 366, 367, 368, 369, 370, 371, 372, 373, 374,
 375, 376, 377, 378, 379, 380, 381, 382, 383, 384,
 385, 386, 387, 388, 389, 390, 392, 393, 394, 408,
 410, 411, 412, 413, 414, 415, 416, 418, 420, 421,
 423, 424, 425, 426, 427, 429, 430, 431, 432, 433,
 434, 435, 436, 438, 439, 440, 441, 442, 443, 444,
 445, 446, 447, 448, 449, 450, 451, 452, 453, 454,
 455, 456, 457, 458, 459, 460, 461, 462, 463, 464,
 465, 466, 467, 468, 469, 470, 472, 473, 474, 475,
 476, 477, 478, 480, 481, 483, 484, 485, 486, 487,
 488, 489, 490, 491, 492, 493, 494, 495, 496, 497,
 498, 499, 500, 502, 503, 504, 505, 506, 507, 508,
 509, 510, 511, 512, 514, 515, 516, 519, 520, 522,

523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558.

unlock_cout_mutex: 210, 211, 252, 253, 254, 255, 412, 413, 414, 415, 416, 418, 425, 426, 427, 434, 435, 466, 467, 468, 469, 470, 472, 473, 474, 475, 476, 477, 478, 500, 510, 511, 512, 515, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558.

UNMARK_YY: 38, 128.

UNMARK_ZZ: 385, 403, 404.

UNSIGNED_INTEGER_YY: 69, 128.

UNSIGNED_INTEGER_ZZ: 268, 403.

UNSIGNED_LONG_INTEGER_YY: 70, 128.

UNSIGNED_LONG_INTEGER_ZZ: 269, 403.

URI_YY: 50, 128.

URI_ZZ: 405.

user_cert: 138.

user_id: 138, 217, 218, 484, 512, 522.

USER_ID_ZZ: 344, 403.

user_info_ptr: 498, 499, 515, 516, 517, 519, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543.

User_Info_Type: 519.

user_name: 485, 523.

USER_NAME_ZZ: 346, 403.

USER_YY: 44, 128.

USER_ZZ: 404.

username: 138, 217, 218, 512, 523.

v: 121, 125, 396, 400.

VALIDITY_NOT_AFTER_ZZ: 355, 403.

VALIDITY_NOT_BEFORE_ZZ: 354, 403.

Validity_notAfter: 497, 499.

Validity_notBefore: 496, 498.

VALUE_ZZ: 319, 403, 406.

vector: 155, 156, 192, 218, 244, 500.

warnings_occurred: 66, 147, 148, 160, 185, 191, 208, 209, 210, 211, 212, 213, 217, 218, 247, 434, 446, 452, 475, 476, 477, 478, 500, 546, 551, 554.

WEXITSTATUS: 470.

WHOAMI_YY: 104, 128.

WHOAMI_ZZ: 371, 403, 515.

WIFEXITED: 470.

write_to_database: 433, 436.

WRITE_YY: 63, 128.

WRITE_ZZ: 405.

X509_Cert_Type: 480.

yyerror: 121, 125.

yyget_extra: 131, 133, 134, 136, 137, 138, 139, 140, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 153, 154, 155, 156, 159, 160, 161, 162, 163, 166, 168, 169, 171, 172, 174, 175, 177, 178, 179, 180, 181, 182, 183, 184, 185, 187, 188, 189, 190, 191, 192, 194, 195, 196, 197, 198, 199, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 215, 217, 218, 222, 223, 224, 226, 227, 228, 229, 230, 231, 232, 233, 235, 236, 237, 238, 239, 241, 243, 244, 246, 247, 248, 250, 251, 252, 253, 254, 255.

yylex: 125.

yylval: 66, 67, 68, 69, 70, 116, 117, 118, 267, 268, 269, 369, 370, 371, 389, 392, 393.

yyscan_t: 120, 125, 395, 400.

YYSTYPE: 125, 400.

yytext: 11, 12, 13, 14, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 116, 117, 118, 119, 267, 268, 269, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 392, 393, 394.

yywrap: 120, 125.

zzerror: 396, 400.

zzget_extra: 408, 410, 411, 412, 413, 414, 415, 416, 420, 421, 423, 424, 425, 426, 427, 429, 430, 431, 432, 433, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 472, 473, 474, 475, 476, 477, 478, 480, 481, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 514, 519, 520, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558.

zzlex: 400.

zzwrap: 395, 400.

⟨ Declarations of additional functions 125, 400 ⟩ Used in sections 257 and 560.
⟨ Delete handle value or values 241 ⟩ Used in section 240.
⟨ Execute on entry to *yylex* 8 ⟩ Used in section 122.
⟨ Execute on entry to *zzlex* 262 ⟩ Used in section 397.
⟨ Garbage 256, 559 ⟩ Used in sections 257 and 560.
⟨ Get user info action 514, 515, 516, 517 ⟩ Used in section 513.
⟨ Include files 3, 124, 259, 399 ⟩ Used in sections 122, 257, 397, and 560.
⟨ Local variables for *yylex* 7 ⟩ Used in section 122.
⟨ Local variables for *zzlex* 261 ⟩ Used in section 397.
⟨ Name definitions 6 ⟩ Used in section 122.
⟨ Options 5, 126, 260, 401 ⟩ Used in sections 122, 257, 397, and 560.
⟨ Rules 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 130, 131, 133, 134, 136, 137, 138, 139, 140, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 153, 154, 155, 156, 157, 159, 160, 161, 162, 163, 164, 165, 166, 168, 169, 171, 172, 174, 175, 177, 178, 179, 180, 181, 182, 183, 184, 185, 187, 188, 189, 190, 191, 192, 194, 195, 196, 197, 198, 199, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 216, 218, 220, 221, 223, 224, 226, 227, 228, 229, 230, 231, 232, 233, 235, 236, 237, 238, 239, 240, 242, 244, 246, 247, 248, 250, 251, 252, 253, 254, 255, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 407, 408, 410, 411, 412, 413, 414, 415, 416, 417, 418, 420, 421, 423, 424, 425, 426, 427, 429, 430, 431, 432, 433, 434, 435, 436, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 480, 481, 483, 484, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 519, 520, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558 ⟩ Used in sections 122, 257, 397, and 560.
⟨ Start conditions 4 ⟩ Used in section 122.
⟨ Token and type declarations 128, 129, 141, 152, 158, 167, 170, 173, 176, 186, 193, 200, 225, 234, 245, 249, 403, 404, 405, 406, 419, 422, 428, 437, 501 ⟩ Used in sections 257 and 560.
⟨ Undelete handle value or values 243 ⟩ Used in section 242.
⟨ process pending operations action 222 ⟩ Used in sections 220 and 221.
⟨ show certificate user action 215 ⟩ Used in section 214.
⟨ show certificates all action 217 ⟩ Used in section 216.
⟨ **union** declaration 127, 402 ⟩ Used in sections 257 and 560.
⟨ *yyerror* definition 121 ⟩ Used in section 122.
⟨ *yywrap* definition 120 ⟩ Used in section 122.
⟨ *zzerror* definition 396 ⟩ Used in section 397.
⟨ *zzwrap* definition 395 ⟩ Used in section 397.

GWDG iRODS/Handle Interface Client/Server Application III: Version 1.0

by Laurence D. Finston

September 2013

	Section	Page
GWDG iRODS/Handle Interface Client-Server Application	1
Scanner for server program gwirdsif	2	1
Parser for server program gwirdsif	123	41
Client Scanner	258	144
Client Parser	398	180
Index	561	289

1. GWDG iRODS/Handle Interface Client-Server Application.

2. FastCGI Web Application gwrdbap. For use with iRODS and the Handle System. [LDF 2012.06.28.] ■

3. Include files. [LDF 2012.06.28.]

```
<Include files 3>≡
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>
#include <time.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>
#include <sys/mman.h>
#include <iomanip>
#include <iostream>
#include <map>
#include <fstream>
#include <sstream>
#include <string>
#include <vector>
#include <deque>
#include <stack>
#include <set>
#include <sys/types.h>
#include <sys/wait.h> /* Library is /usr/lib64/libfcgi.a */
#include <fcgi_stdio.h>
#include <pthread.h>
#include <gcrypt.h> /* for gcry_control */
#include <gnutls/gnutls.h>
#include <gnutls/x509.h>
#include <mysql.h>
#include <expat.h>
#include <curl/curl.h>
#undef NAME_LEN
#undef LOCAL_HOST
#ifndef HAVE_CONFIG_H
    # include <config.h >
#endif
#ifndef _XOPEN_SOURCE
#define _XOPEN_SOURCE
#endif
#include "glblcnst.h++"
#include "glblvrbl.h++"
#include "excptntp.h++"
#include "utilfncts.h++"
#include "groupntp.h++"
#include "hndlvltp.h++"
#include "irdsavtp.h++"
#include "cmdlnopt.h++"
```

```
#include "rspnstp.h++"
#include "irdsobtp.h++"
#include "hdltype.h++"
#include "dcmtddtp.h++"
#include "x509cert.h++"
#include "dstngnmt.h++"
    typedef int YYSTYPE;
#include "scprpmtp.h++"
#include "pidfncts.h++"
#include "tanfncts.h++"
#include "clntfncts.h++"
#include "ckietype.h++"
#include "utlwbfcs.h++"
```

See also sections 31 and 68.

This code is used in sections 29, 64, and 75.

4. Global variables. [LDF 2012.06.28.]

```
⟨ Global variables 4 ⟩ ≡
extern char **environ;
CURL *curl;
CURLcode curl_res;
const string sock_path = "/tmp/gwirdsif.sock";
```

This code is used in section 29.

5. Finish (exit handler). [LDF 2012.07.24.]

Log

[LDF 2012.07.24.] Added this function.

[LDF 2013.09.19.] Added code for unlocking the memory for *gpg-key-id* and *gpg-passphrase* and deleting them.

```
<finish definition 5> ≡
void finish(void)
{
    bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    if (gpg-key-id ≠ 0) {
        munlock(gpg-key-id, 10);
        delete[] gpg-key-id;
        gpg-key-id = 0;
    }
    if (gpg-passphrase ≠ 0) {
        munlock(gpg-passphrase, gpg-passphrase_length);
        delete[] gpg-passphrase;
        gpg-passphrase = 0;
    }
    pthread_mutex_destroy(&cerr_mutex);
    pthread_mutex_destroy(&session_data_mutex);
    pthread_mutex_destroy(&log_strm_mutex);
    pthread_mutex_destroy(&cookie_vector_mutex);
    return;
} /* End of finish definition */
```

This code is used in section 29.

6. Main function definition. [LDF 2012.06.28.]

Log

[LDF 2012.07.24.] Added code for opening log file.

```

⟨ main definition 6 ⟩ ≡
int main(int argc, char *argv[]){ is_gwrdbap = true;
    int loop_ctr = 0;
    bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int j;
    curl = curl_easy_init();
    int temp_val = 0;
    int args = 0;
    char **arg_str_vector = 0;
    pthread_mutex_init(&cerr_mutex, 0); /* Required in client_func. */
    pthread_mutex_init(&session_data_mutex, 0);
    pthread_mutex_init(&log_strm_mutex, 0);
    pthread_mutex_init(&cookie_vector_mutex, 0);
    map<string, Scan_Parse_Parameter_Type *> session_data_map;
    char log_filename[] = "/tmp/gwrdbap.log";
    log_strm.open(log_filename, ios_base::app);
    stringstream outer_temp_strm;
    if (!(log_strm & log_strm.is_open())) {
        outer_temp_strm.str("");
        outer_temp_strm << "[gwrdbap] ERROR! In 'main': Failed to open 'log_strm'." <<
            endl << "Exiting 'gwrdbap.fcgi' unsuccessfully with exit status 1." << endl;
        fprintf(stderr, outer_temp_strm.str().c_str());
        exit(1);
    } /* if */
#endif DEBUG_COMPILE
    if (DEBUG) {
        chmod(log_filename, 0744);
        log_strm << "Here I am." << endl;
        log_strm << flush;
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    int outer_status = atexit(finish);
    if (outer_status != 0) {
        log_strm << "ERROR! In 'main': 'atexit' failed, returning" << outer_status <<
            "." << endl << "Failed to set exit function 'finish'." << endl <<
            "Exiting program 'gwrdbap.fcgi' with exit status 1." << endl;
        exit(1);
    } /* if */
#endif DEBUG_COMPILE
    else
        if (DEBUG) {
            log_strm << "Set exit function 'finish' successfully." << endl;
        } /* else if (DEBUG) */

```

```
#endif /* DEBUG_COMPILE */ /* !! TODO: LDF 2012.07.24. Fake some expired cookies to
   see if Cookie_Type::delete_expired_cookies works properly. */ /* The global variables
   admin_data and admin_data_length aren't used by gwrdbap.fcgi. [LDF 2013.01.11.] */
   memset(admin_data, 0, 64);
   admin_data_length = 0;
```

See also sections 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, and 28.

This code is used in section 29.

7. Start thread for deleting expired cookies. [LDF 2012.07.24.]

Log

[LDF 2012.07.24.] Added this section.

[LDF 2012.11.16.] Added **unsigned int sleep_val**. Now passing it to **Cookie_Type::delete_expired_cookies**.

```
{ main definition 6 } +≡
pthread_attr_t attr;
pthread_attr_init(&attr);
pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);
pthread_t delete_expired_cookies_thread_id;
unsigned int sleep_val = 10; /* 900 */ /* 900seconds = 15minutes */
outer_status = pthread_create(&delete_expired_cookies_thread_id, &attr, /* Attribute */
Cookie_Type::delete_expired_cookies, static_cast<void*>(&sleep_val));
if (outer_status != 0) {
    log_strm << "ERROR! In 'main': 'pthread_create' failed for 'delete_expired_cookies'." <<
        endl << "Exiting 'gwrdbap.fcgi' unsuccessfully with exit status 1." << endl;
    exit(1);
}
#endif DEBUG_COMPILE
else
    if (DEBUG) {
        log_strm << "In 'main': 'pthread_create' succeeded for 'delete_expired_cookies'." <<
            endl;
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

8. *FCGLAccept* loop. [LDF 2012.06.28.]

This is the main “accept” loop. It will iterate each time this web application is accessed, by means of a browser, `curl`, `wget` or some other way. [LDF 2012.06.28.]

```
< main definition 6 > +≡
while ((j = FCGLAccept()) ≥ 0) { int status;
  string temp_str;
  string content_type;
  string client_dn;
  stringstream temp_strm;
  stringstream atemp_strm;
  char c;
  char *temp_buff = 0;
  Scan_Parse_Parameter_Type *param = 0;
  ++loop_ctr;
  bool post_data_present = false;
  bool get_data_present = false;
  char *unesapeded_str = 0;
  int char_ctr = 0;
  string session_id;
  map<string, Scan_Parse_Parameter_Type *>::iterator session_data_iter;
  log_strm ≪ "*****" ≪ time(0) ≪ " " ≪ get_datestamp() ≪ endl;
```

9. Check HTTP_COOKIE. [LDF 2012.07.23.]

Log

[LDF 2012.07.23.] Added this section.

```

⟨ main definition 6 ⟩ +≡
    temp_buff = getenv("HTTP_COOKIE");
    if (temp_buff & strlen(temp_buff) > 0) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        atemp_strm.str("");
        atemp_strm ≪ "HTTP_COOKIE" ≪ temp_buff ≪ "." ≪ endl;
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    status = Cookie_Type::parse_cookies(temp_buff, session_id, atemp_strm);
    if (status ≠ 0) {
        atemp_strm ≪ "WARNING! 'parser_cookies' failed, returning" ≪ status ≪
            ". " ≪ "Failed to parse cookies. Any session data present in a cookie" ≪
            "could not be processed." ≪ endl ≪ "Will try to continue." ≪ endl;
    } /* if (status ≠ 0) */
#endif DEBUG_COMPILE
    else
        if (DEBUG) {
            atemp_strm ≪ "'parse_cookies' succeeded, returning 0." ≪ endl ≪ "'session_id' =="
                session_id ≪ endl;
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    } /* if (temp_buff & strlen(temp_buff) > 0) */
    else {
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            atemp_strm ≪ "HTTP_COOKIE is empty." ≪ endl;
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        if (session_id.empty()) {
#ifndef DEBUG_COMPILE
            if (DEBUG) {
                atemp_strm ≪ "'session_id' is empty. Calling 'Cookie_Type::generate_session_id'."
                    endl;
            } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
            status = Cookie_Type::generate_session_id(session_id, atemp_strm);
            if (status ≠ 0) {
                atemp_strm ≪ "ERROR! In 'main': 'Cookie_Type::generate_session_id' failed, "
                    "returning" ≪ status ≪ ". " ≪ endl ≪
                    "Failed to generate session ID. Continuing." ≪ endl;
            } /* if (status ≠ 0) */
#endif DEBUG_COMPILE
            else
                if (DEBUG) {

```

```

atemp_strm << "In 'main': 'Cookie_Type::generate_session_id' succeeded," <<
    "returning 0." << endl << "'session_id'" <= " " << session_id << endl;
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
} /* if (session_id.empty()) */
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        atemp_strm << "'session_id' is not empty." <<
            "Not calling 'Cookie_Type::generate_session_id'." << endl;
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

10. Output HTML file headers, the counter for the *FCGI_Accept* loop and the return value of *FCGI_Accept*. [LDF 2012.06.28.]

```

< main definition 6 > +≡
temp_strm.str("");
temp_strm << "Content-Type: text/plain; charset=iso-8859-1\n";
atemp_strm << "About_to_output_header.cookie_vector.size()" <= " " << cookie_vector.size() <<
    endl;
for (vector<Cookie_Type>::const_iterator iter = cookie_vector.begin(); iter != cookie_vector.end();
    ++iter) {
    iter->show("Cookie:", &atemp_strm);
    temp_strm << "Set-Cookie:" << iter->name << "=";
    if (!iter->value.empty()) temp_strm << iter->value;
    temp_strm << ";" << "Expires=" << iter->expires_str << ";" << "Domain=.pcfinston.gwdg.de" << endl;
} /* for */
temp_strm << "\n" << "<!DOCTYPE html\n" <<
    "PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"\n" <<
    "\"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd\">\n";
temp_strm << atemp_strm.str();
atemp_strm.str("");
temp_strm << "loop_ctr" <= " " << loop_ctr << endl;
if (temp_buff) {
    temp_strm << "HTTP_USER_AGENT" <= " " << temp_buff << endl;
}
else temp_strm << "HTTP_USER_AGENT" <= "NULL" << endl;
printf("%s", temp_strm.str().c_str());

```

11. Output environment variables. [LDF 2010.03.31.]

```

< main definition 6 > +≡
char *curr_var;
int k = 0;
#ifndef 0 /* 1 */
temp_strm << "Environment Variables\n\n";
while ((curr_var = environ[k++])) temp_strm << curr_var << endl;
printf("%s", temp_strm.str().c_str());
temp_strm.str("");
k = 0;
#endif

```

12. Get the value of `HTTP_USER_AGENT` and output it. [LDF 2012.06.28.]

```
{ main definition 6 } +≡
temp_buff = getenv("HTTP_USER_AGENT");
```

13. Parse the value of the `QUERY_STRING` environment variable. It contains any keyword-value pairs passed in the URL. [LDF 2010.03.31.] [LDF 2010.09.30.]

```
{ main definition 6 } +≡
vector<pair<string, string>> parameters;
string query_string = getenv("QUERY_STRING");
if (DEBUG) {
    temp_strm.str("");
    temp_strm << "Query\u00a0String\n";
} /* if (DEBUG) */
if (query_string.empty()) {
    if (DEBUG) temp_strm << "query_string\u00a0=\u00a0NULL.\u00a0" << endl;
}
else {
    int m = 0;
    get_data_present = true;
    if (DEBUG) temp_strm << "query_string\u00a0=\u00a0" << query_string << endl;
} /* else */
if (DEBUG) printf("%s", temp_strm.str().c_str());
```

14. Get client Distinguished Name. It's stored in the `mod_ssl` environment variable `SSL_CLIENT_S_DN`. The client certificate is stored in the `mod_ssl` environment variable `SSL_CLIENT_CERT`. [LDF 2012.06.28.]

Log

[LDF 2012.07.27.] Added error handling. Without `SSL_CLIENT_S_DN`, `Scan_Parse_Parameter_Type::Distinguished_Name` can't be set, i.e., the user can't be identified.

```
{ main definition 6 } +≡
/* !! TODO: LDF 2013.05.10. Removed Scan_Parse_Parameter_Type::Distinguished_Name. I
   could put it back, if needed. Not currently working on gwrdbap. */
temp_buff = getenv("SSL_CLIENT_S_DN");
temp_strm.str("");
if (temp_buff) {
    temp_strm << "SSL_CLIENT_S_DN\u00a0=\u00a0" << temp_buff << endl;
    client_dn = temp_buff;
    if (DEBUG) printf("%s", temp_strm.str().c_str());
}
else /* Error handling */
{
    temp_strm.str("");
    temp_strm << "[gwrdbap]\u00a0ERROR!\u00a0In\u00a0'main':\u00a0SSL_CLIENT_S_DN\u00a0=\u00a0NULL" << endl <<
        "No\u00a0Distinguished\u00a0Name.\u00a0\u00a0This\u00a0isn't\u00a0allowed.\u00a0" << endl << "Breaking.\u00a0" << endl;
    printf("%s", temp_strm.str().c_str());
    goto END_OF_MAIN_ACCEPT_LOOP;
} /* else */
```

15.

```

⟨ main definition 6 ⟩ +≡
  if (DEBUG) {
    temp_strm.str("");
    temp_buff = getenv("SSL_PROTOCOL");
    if (temp_buff) temp_strm << "SSL_PROTOCOL" << temp_buff << endl;
    else temp_strm << "SSL_PROTOCOL=NULL" << endl;
  #if 0
    temp_buff = getenv("SSL_CLIENT_CERT");
    if (temp_buff) temp_strm << "SSL_CLIENT_CERT" << temp_buff << endl;
    else temp_strm << "SSL_CLIENT_CERT=NULL" << endl;
    temp_buff = getenv("SSL_SERVER_CERT");
    if (temp_buff) temp_strm << "SSL_SERVER_CERT" << temp_buff << endl;
    else temp_strm << "SSL_SERVER_CERT=NULL" << endl;
  #endif
    temp_buff = getenv("SSL_CLIENT_V_END");
    if (temp_buff) temp_strm << "SSL_CLIENT_V_END" << temp_buff << endl;
    else temp_strm << "SSL_CLIENT_V_END=NULL" << endl;
    printf("%s", temp_strm.str().c_str());
  } /* if (DEBUG) */
}

```

16. Check REQUEST_METHOD. [LDF 2012.07.09.]

```

⟨ main definition 6 ⟩ +≡
  temp_buff = getenv("REQUEST_METHOD");
  if (temp_buff & strlen(temp_buff) > 0 & strcmp(temp_buff, "POST") ≡ 0) {
    if (DEBUG) {
      temp_strm.str("");
      temp_strm << "REQUEST_METHOD=POST. Will process POST data." << endl;
      printf("%s", temp_strm.str().c_str());
    } /* if (DEBUG) */
  }
  else {
    if (DEBUG) {
      temp_strm.str("");
      temp_strm << "REQUEST_METHOD!=POST. No POST data to process." << endl;
      printf("%s", temp_strm.str().c_str());
    } /* if (DEBUG) */
    goto END_PROCESS_POST_DATA;
  } /* else */
}

```

17. Check for POST data. [LDF 2012.07.03.]

```

{ main definition 6 } +≡ /* !! TODO: LDF 2012.07.26. Check CONTENT_TYPE with other requests: I
    think curl may send POST data with other requests, too. */
temp_str = "application/x-www-form-urlencoded";
temp_buff = getenv("CONTENT_TYPE");
temp_strm.str("");
if (temp_buff) {
    temp_strm << "POST_data_is_present ." << endl;
    content_type = temp_buff;
    status = content_type.compare(0, temp_str.size(), temp_str);
}
else {
    status = -1;
    temp_strm << "No_parameters_submitted_to_stdin_using_POST ." << endl;
}
printf("%s", temp_strm.str().c_str());
if (status ≠ 0) {
    temp_strm.str("");
    temp_strm << "WARNING!_Content_type_=_" << temp_buff << endl <<
        "Should_be_\"application/x-www-form-urlencoded\"." << endl <<
        "Can't_process_POST_data._Will_try_to_continue." << endl;
    printf("%s", temp_strm.str().c_str());
    goto END_PROCESS_POST_DATA;
} /* if (status ≠ 0) */
else if (DEBUG) {
    temp_strm.str("");
    temp_strm << "Content_type_is_\"application/x-www-form-urlencoded\"." << endl <<
        "Will_process_POST_data." << endl;
    printf("%s", temp_strm.str().c_str());
} /* else if (DEBUG) */

```

18.

Log

[LDF 2012.07.27.] Moved this section above the one where POST data is processed. This makes it possible to pass *param* to *parse_post_data* and put *param-Distinguished_Name* onto the front of the POST data.

```

⟨ main definition 6 ⟩ +≡
    pthread_mutex_lock(&session_data_mutex);
    session_data_iter = session_data_map.find(session_id);
    if (session_data_iter == session_data_map.end()) {
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            temp_strm.str("");
            temp_strm << "'session_data_iter'" << " == " << 'session_data_map.end()' . " << endl <<
                "New_session_.Creating_a_new_Scan_Parse_Parameter_Type' object" <<
                "and_pushing_new_pair_onto_map' . " << endl;
            printf("%s", temp_strm.str().c_str());
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
        param = new Scan_Parse_Parameter_Type;
        session_data_map[session_id] = param;
    } /* if (session_data_iter == session_data_map.end()) */
    else {
        param = session_data_iter->second;
#ifndef DEBUG_COMPILE
        if (DEBUG) {
            temp_strm.str("");
            temp_strm << "'session_data_iter'" != "session_data_map.end()' . " << endl <<
                "Old_session_.Will_use_existing_Scan_Parse_Parameter_Type' object." << endl;
            printf("%s", temp_strm.str().c_str());
        } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    } /* else */
    param->set_expires(7200);
    /* Increase Scan_Parse_Parameter_Type::expires value by two hours (7200 seconds). set_expires
       is so unlikely to fail that I don't bother checking the return value. [LDF 2012.07.27.] */
    if ((client_dn.length() + strlen("DISTINGUISHED_NAME") + 4) > BUFFER_SIZE) {
        temp_strm.str("");
        temp_strm << "[gwrdbap]_ERROR!_main:'" <<
            "((client_dn.length() + strlen(\"DISTINGUISHED_NAME\")) + 4)" <<
            ">" << "BUFFER_SIZE'" << endl << "client_dn.length()" == " << client_dn.length() <<
            endl << "((client_dn.length() + strlen(\"DISTINGUISHED_NAME\")) + 4)" == " <<
            "(client_dn.length() + strlen(\"DISTINGUISHED_NAME\") + 4) << endl << "BUFFER_SIZE" == " <<
            BUFFER_SIZE << endl << "Breaking." << endl;
        printf("%s", temp_strm.str().c_str());
        if (param) {
            delete param;
            param = 0;
            session_data_map.erase(session_data_iter);
        }
        goto END_OF_MAIN_ACCEPT_LOOP;
    } /* if */

```

```

strcpy(param->data_buffer, "DISTINGUISHED_NAME\"");
strcat(param->data_buffer, client_dn.c_str());
strcat(param->data_buffer, "\n");
#ifndef DEBUG_COMPILE
if (DEBUG) {
    char atemp_str[128];
    temp_strm.str("");
    temp_strm << "'param->get_expires()' == " << param->get_expires(atemp_str,
        127) << " == " << atemp_str << endl;
    printf("%s", temp_strm.str().c_str());
}
/* if (DEBUG) */
#endif /* DEBUG_COMPILE */
pthread_mutex_unlock(&session_data_mutex);

```

19. Read POST data from standard input. [LDF 2012.06.28.]

```

⟨ main definition 6 ⟩ +≡
status = parse_post_data(*param);

```

20. *parse_post_data* failed, returning -1. [LDF 2012.07.27.]

```

⟨ main definition 6 ⟩ +≡
if (status ≡ -1) {
    temp_strm.str("");
    temp_strm << "ERROR! 'parse_post_data' failed, returning -1." << endl <<
        "Failed to parse standard input, which contains the POST data." << endl <<
        "Breaking." << endl;
    printf("%s", temp_strm.str().c_str());
    if (param) {
        delete param;
        param = 0;
        session_data_map.erase(session_data_iter);
    }
    goto END_OF_MAIN_ACCEPT_LOOP;
} /* if (status ≠ 0) */

```

21. *parse_post_data* succeeded, returning 0. POST data is in *param->data_buffer*. [LDF 2012.07.27.]

```

⟨ main definition 6 ⟩ +≡
#ifndef DEBUG_COMPILE
else
if (DEBUG) {
    temp_strm.str("");
    temp_strm << "'parse_post_data' succeeded, returning " << status << "." << endl;
    printf("%s", temp_strm.str().c_str());
}
/* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

22. Error handling: *param->data_buffer* is NULL or empty. [LDF 2012.07.27.]

```

⟨ main definition 6 ⟩ +≡
  if (status == 0 ∧ (param->data_buffer == 0 ∨ strlen(param->data_buffer) == 0)) {
    temp_strm.str("");
    temp_strm << "ERROR! " "status" == 0, but "param->data_buffer" == 0" <<
      "or " "strlen(param->data_buffer)" == 0." << endl << "This isn't supposed to happen." <<
      endl << "Failed to parse standard input, which contains the POST data." << endl <<
      "Breaking." << endl;
    printf("%s", temp_strm.str().c_str());
  if (param) {
    delete param;
    param = 0;
    session_data_map.erase(session_data_iter);
  }
  goto END_OF_MAIN_ACCEPT_LOOP;
} /* if */

```

23.

```

⟨ main definition 6 ⟩ +≡
  else
    if (status == 0) {
      post_data_present = true;
#ifndef DEBUG_COMPILE
      if (DEBUG) {
        temp_strm.str("");
        temp_strm << "'status' == 0, 'param->data_buffer' != 0" <<
          "and " "strlen(param->data_buffer)" > 0." << endl <<
          "POST data is contained in 'param->data_buffer'." << endl;
        printf("%s", temp_strm.str().c_str());
      } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    } /* else if (status == 0) */

```

24. *parse_post_data* returned 1: POST data has been written to *out_strm*. [LDF 2012.07.27.]

```

⟨ main definition 6 ⟩ +≡
  else
    if (status == 1) {
#ifndef DEBUG_COMPILE
      if (DEBUG) {
        temp_strm.str("");
        temp_strm << "'status' == 1. POST data has been written to 'out_strm'." << endl;
        printf("%s", temp_strm.str().c_str());
      } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
      post_data_present = true;
    } /* else if (status == 1) */
END_PROCESS_POST_DATA:

```

25. Check for GET data. [LDF 2012.07.03.]

Log

[LDF 2012.07.03.] Added this section.

(main definition 6) +≡

26. !! TODO: Handle GET data. [LDF 2012.07.09.]

```
( main definition 6 ) +≡
if ( $\neg$ (post_data_present  $\vee$  get_data_present)) {
    temp_strm.str("");
    temp_strm  $\ll$  "WARNING! No POST or GET data. Nothing to do. Quitting."  $\ll$  endl;
    printf("%s", temp_strm.str().c_str());
    goto END_OF_MAIN_ACCEPT_LOOP;
} /* if ( $\neg$ (post_data_present  $\vee$  get_data_present)) */
#if DEBUG_COMPILE
else
    if (DEBUG) {
        temp_strm.str("");
        if (post_data_present) temp_strm  $\ll$  "POST data is present."  $\ll$  endl;
        if (get_data_present) temp_strm  $\ll$  "GET data is present."  $\ll$  endl;
        printf("%s", temp_strm.str().c_str());
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

27.

```
( main definition 6 ) +≡
    status = client_func(*param);
    if (status  $\neq$  0) {
        temp_strm.str("");
        temp_strm  $\ll$  "ERROR! 'client_func' failed, returning"  $\ll$  status  $\ll$  "."  $\ll$  endl  $\ll$ 
            "Breaking."  $\ll$  endl;
        printf("%s", temp_strm.str().c_str());
        if (param) {
            delete param;
            param = 0;
            session_data_map.erase(session_data_iter);
        }
        goto END_OF_MAIN_ACCEPT_LOOP;
} /* if (status  $\neq$  0) */
else if (DEBUG) {
    printf(" 'client_func' succeeded, returning 0.\n");
} /* else if (DEBUG) */
```

28.

```
( main definition 6 ) +≡
END_OF_MAIN_ACCEPT_LOOP: ; } /* while */
curl_easy_cleanup(curl);
log_strm.close();
exit(0); } /* End of main definition */
```

29. This is what's compiled. [LDF 2012.06.28.]

!! PLEASE NOTE: The **typedefs** for *jobject*, *JNIEnv* and *yyscan_t* below are needed for types used in class **Scan_Parse_Parameter_Type** or its member functions and **int yydebug** is used in *process_command_line_options* in *cmdlnopt.web*. If I add a scanner/parser pair for this program, I will need to get rid of the **typedef** for *yyscan_t* and the declaration of **int yydebug**. [LDF 2012.07.20.]

```
typedef void *yyscan_t;
⟨ Include files 3 ⟩
using namespace std;
using namespace gwrdifpk;
GCRY_THREAD_OPTION_PTHREAD_IMPL;
gnutls_anon_server_credentials_t anoncred;
⟨ Global variables 4 ⟩
int yydebug;
⟨ finish definition 5 ⟩
⟨ main definition 6 ⟩ /* !! TODO: LDF 2012.07.20. Put these definitions somewhere else and include them here and in 'gwirdcli.web'. */
```

30. **Cookie_Type.** [LDF 2011.12.22.]

Log

[LDF 2011.12.22.] Created this file.

[LDF 2012.01.09.] Now including *mysql.h*.

[LDF 2012.07.23.] Copied this file from the directory [...] /optinum/Installer/optwbsrv/src/ of the OptiNum-Grid project.

format *Cookie_Type int*

31. Include files.

```
<Include files 3> +≡
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <mysql.h>
#include <iomanip>
#include <iostream>
#include <map>
#include <fstream>
#include <sstream>
#include <string>
#include <vector>
#include <algorithm>
#include <pthread.h>
#include <config.h>
#include "glblcnst.h++"
#include "glblvrb1.h++"
#include "excptntp.h++"
#include "utilfncts.h++"
#ifndef _XOPEN_SOURCE
#define _XOPEN_SOURCE
#endif
```

32. Global variables for **Cookie_Type**. [LDF 2012.07.24.]

Log

[LDF 2012.07.24.] Added this section.

```
<Cookie_Type Global variables 32> ≡
pthread_mutex_t cookie_vector_mutex;
vector<Cookie_Type> cookie_vector;
```

This code is used in section 64.

33.

```
<Cookie_Type extern declarations for global variables 33> ≡
extern pthread_mutex_t cookie_vector_mutex;
extern vector<Cookie_Type> cookie_vector;
```

This code is used in section 66.

34. class Cookie_Type. [LDF 2011.12.22.]**Log**

[LDF 2011.12.22.] Added this **struct** declaration.
 [LDF 2012.07.26.] Changed **Cookie_Type** from a **struct** to a **class**. Added data member **Scan_Parse_Parameter_Type** **param*.
 [LDF 2012.07.26.] Made **vector<Cookie_Type>** *cookie_vector* a **static** data member of **Cookie_Type**.

```
⟨ class Cookie_Type declaration 34⟩ ≡
class Cookie_Type {
    friend int main(int, char *[]);
    string name;
    string value;
    string domain;
    string path;
    string expires_str;
    time_t expires;
    Scan_Parse_Parameter_Type *param;
    vector<pair<string, string>> name_value_pairs;
public: ⟨ class Cookie_Type function declarations 35⟩
};
```

This code is used in sections 64 and 66.

35. Default constructor. [LDF 2010.09.30.]**Log**

[LDF 2010.09.30.] Added this function.

```
⟨ class Cookie_Type function declarations 35⟩ ≡
Cookie_Type(void);
```

See also sections 37, 39, 41, 43, 45, 47, 49, 53, and 58.

This code is used in section 34.

36.

```
⟨ class Cookie_Type constructor definitions 36⟩ ≡
Cookie_Type::Cookie_Type(void)
{
    name = "";
    value = "";
    domain = "";
    path = "";
    expires_str = "";
    expires = 0;
    return;
}
```

See also section 38.

This code is used in section 64.

37. Copy constructor. [LDF 2011.12.22.]

Log

[LDF 2011.12.22.] Added this function.

`< class Cookie_Type function declarations 35 > +≡
Cookie_Type(const Cookie_Type &c);`

38.

`< class Cookie_Type constructor definitions 36 > +≡
Cookie_Type::Cookie_Type(const Cookie_Type &c)
{
 name = c.name;
 value = c.value;
 domain = c.domain;
 path = c.path;
 expires_str = c.expires_str;
 expires = c.expires;
 for (vector<pair<string, string>>::const_iterator iter = c.name_value_pairs.begin();
 iter ≠ c.name_value_pairs.end(); ++iter) name_value_pairs.push_back(*iter);
 return;
}`

39. Assignment operator. [LDF 2011.12.22.]

Log

[LDF 2011.12.22.] Added this function.

`< class Cookie_Type function declarations 35 > +≡
void operator=(const Cookie_Type &c);`

40.

`< Cookie_Type::operator= definition 40 > ≡
void Cookie_Type::operator=(const Cookie_Type &c)
{
 name = c.name;
 value = c.value;
 domain = c.domain;
 path = c.path;
 expires_str = c.expires_str;
 expires = c.expires;
 name_value_pairs.clear();
 for (vector<pair<string, string>>::const_iterator iter = c.name_value_pairs.begin();
 iter ≠ c.name_value_pairs.end(); ++iter) name_value_pairs.push_back(*iter);
 return;
}`

This code is used in section 64.

41. Equality operator. [LDF 2012.07.23.]

Log

[LDF 2012.07.23.] Added this function.
 [LDF 2013.03.20.] Made this function **const**.

`< class Cookie_Type function declarations 35 > +≡
 int operator≡(const Cookie_Type &c) const;`

42.

`< Cookie_Type::operator≡ definition 42 > ≡
 int Cookie_Type::operator≡(const Cookie_Type &c) const
 {
 return (name ≡ c.name ∧ value ≡ c.value ∧ domain ≡ c.domain ∧ path ≡ c.path);
}`

See also section 44.

This code is used in section 64.

43. Less-than operator. [LDF 2012.07.24.]

Log

[LDF 2012.07.24.] Added this function.

`< class Cookie_Type function declarations 35 > +≡
 bool operator<(const Cookie_Type &c) const;`

44.

`< Cookie_Type::operator< definition 42 > +≡
 bool Cookie_Type::operator<(const Cookie_Type &c) const
{
return (expires < c.expires);
}`

45. Clear. [LDF 2011.12.22.]

Log

[LDF 2011.12.22.] Added this function.

`< class Cookie_Type function declarations 35 > +≡
 void clear();`

46.

```

<Cookie_Type::clear definition 46> ==
void Cookie_Type::clear()
{
    name = "";
    value = "";
    domain = "";
    path = "";
    expires_str = "";
    expires = 0;
    name_value_pairs.clear();
    return;
}

```

This code is used in section 64.

47. Show. [LDF 2011.12.22.]

Log

[LDF 2011.12.22.] Added this function.

```
<class Cookie_Type function declarations 35> +=  
    void show(string s = "Cookie_Type:", stringstream *t = 0) const;
```

48.

This code is used in section 64.

49. Parse Cookies. [LDF 2011.12.22.]

Log

[LDF 2011.12.22.] Added this function.

[LDF 2012.01.06.] Now handling cookie values correctly and filtering out the “*LtpaToken*”, if present. See the comment below for an explanation.

[LDF 2012.07.26.] Made this function a **static** member function of class **Cookie_Type**.

```
<class Cookie_Type function declarations 35> +≡
    static int parse_cookies(const char *http_cookie_str, string &session_id, stringstream &out_strm);
```

50.

```
<Cookie_Type::parse_cookies definition 50> ≡
    int Cookie_Type::parse_cookies(const char *http_cookie_str, string &session_id, stringstream
        &out_strm){ bool DEBUG = false; /* true */
        set_debug_level(DEBUG, 0, 0);
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        out_strm << "Entering 'parse_cookies'.\n";
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    Cookie_Type curr_cookie;
    string cookie_str = http_cookie_str;
    out_strm << "Before_parsing: 'cookie_str' == " << cookie_str << endl; while
        (!cookie_str.empty()) { curr_cookie.name = cookie_str;
        curr_cookie.value = cookie_str;
        string::size_type s = curr_cookie.name.find("=");
        if (s != string::npos) {
            curr_cookie.name.erase(s);
            curr_cookie.value.erase(0, s + 1);
        }
        s = curr_cookie.value.find(";");
        if (s != string::npos) {
            curr_cookie.value.erase(s);
        }
        if (cookie_str.size() > curr_cookie.name.size() + curr_cookie.value.size() + 3)
            cookie_str.erase(0, curr_cookie.name.size() + curr_cookie.value.size() + 3);
        else cookie_str = "";
#endif DEBUG_COMPILE
    if (DEBUG) {
        out_strm << "curr_cookie_name == " << curr_cookie.name.c_str() << endl <<
            "curr_cookie_value == " << curr_cookie.value.c_str() << endl << "cookie_str == " <<
            cookie_str.c_str() << endl;
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

See also sections 51 and 52.

This code is used in section 64.

51. On `pcfinston.gwdg.de`, the “`LtpaToken`” (`curr_cookie.name ≡ "LtpaToken"`) is always present, but apparently not stored by the browser. It seems that it always has the same value when the application is accessed multiple times. I don’t know when it changes, if ever. Nor do I know what it’s for. !! TODO: Look into this. This is, however, not urgent.

If I push it onto `cookie_vector`, it appears multiple times in the `HTTP_COOKIE` environment variable, so I don’t do this. [LDF 2012.01.06.] [LDF 2012.07.23.]

Log

[LDF 2012.07.23.] Now only pushing a cookie onto `cookie_vector` if `curr_cookie.name ≡ "gwiridsif_session_id"`. That is, no longer pushing arbitrary cookies onto `cookie_vector`. If I start using other cookies, they can be pushed onto it, too.

```

⟨Cookie_Type::parse_cookies definition 50⟩ +≡
  if (curr_cookie.name ≡ "gwiridsif_session_id") {
    pthread_mutex_lock(&session_data_mutex);
    session_id = curr_cookie.value;
    vector<Cookie_Type>::iterator iter = find(cookie_vector.begin(), cookie_vector.end(), curr_cookie);
    if (iter != cookie_vector.end()) {
#if DEBUG_COMPILE
      if (DEBUG) {
        out_strm << "In 'parse_cookies': " << "Found matching cookie in 'cookie_vector'." <<
          endl << "Will update 'expires_str' field." << endl;
      } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
      iter->expires_str = get_timestamp(2, 0, &iter->expires);
    } /* if */
    else {
#if DEBUG_COMPILE
      if (DEBUG) {
        out_strm << "In 'parse_cookies': " << "No matching cookie in 'cookie_vector'." <<
          endl << "Will push 'curr_cookie' onto it." << endl;
      } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
      curr_cookie.expires_str = get_timestamp(2, 0, &curr_cookie.expires);
      cookie_vector.push_back(curr_cookie);
    } /* else */
    pthread_mutex_unlock(&session_data_mutex);
  } /* if (curr_cookie.name ≡ "gwiridsif_session_id") */
} /* while (!cookie_str.empty()) */

```

52.

```

⟨Cookie_Type::parse_cookies definition 50⟩ +≡
#endif /* DEBUG_COMPILE
  if (DEBUG) {
    out_strm << "Exiting 'parse_cookies'.\n";
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  return 0; /* End of Cookie_Type::parse_cookies definition */

```

53. Generate session ID. [LDF 2012.01.09.]

Log

[LDF 2012.01.09.] Added this function.

[LDF 2012.07.26.] Made this function a **static** member function of **Cookie_Type**.

```
< class Cookie_Type function declarations 35 > +≡
    static int generate_session_id(string &session_id, stringstream &out_strm);
```

54.

```
< Cookie_Type::generate_session_id definition 54 > ≡
    int Cookie_Type::generate_session_id(string &session_id, stringstream &out_strm){ bool
        DEBUG = false; /* true */
        set_debug_level(DEBUG, 0, 0);
        FILE *fp = 0;
        int status;
```

See also sections 55, 56, and 57.

This code is used in section 64.

55.

```
< Cookie_Type::generate_session_id definition 54 > +≡
    fp = fopen("/dev/urandom", "r");
    if (fp == 0) {
        out_strm << "ERROR! In 'generate_session_id': 'fopen' failed:" <<
            endl << "fopen_error:" << strerror(errno) << endl <<
            "Failed to open '/dev/urandom'. Can't generate session ID." << endl <<
            "Exiting function unsuccessfully" << "with return value 1." << endl;
        return 1;
    } /* if (fp == 0) */
    else if (DEBUG) {
        out_strm << "In 'generate_session_id': 'fopen' succeeded." << endl;
    } /* else if (DEBUG) */
```

56.

```
< Cookie_Type::generate_session_id definition 54 > +≡
    if (DEBUG) {
        out_strm.str("");
        if (session_id_vector.size() == 0)
            out_strm << "In 'generate_session_id': session_id_vector.size() == 0" << endl;
        else {
            for (vector<string>::const_iterator iter = session_id_vector.begin(); iter != session_id_vector.end();
                ++iter) out_strm << "*iter" << *iter << endl;
        }
    } /* if (DEBUG) */
```

57.

```

⟨ Cookie_Type::generate_session_id definition 54 ⟩ +≡
    char buffer[16];
    char c;
    int i;

    vector<string>::const_iterator iter;
    for ( ; ; ) {
        memset(buffer, 0, 16);
        for (i = 0; i < 15; )      /* Increment i below, not here! [LDF 2012.01.09.] */
        {
            c = fgetc(fp);
            if (isalnum(c)) {
#ifndef DEBUG_COMPILE
                if (DEBUG) {
                    out_strm << "i== " << i << ",c== " << c << endl;
                } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
                buffer[i++] = c;
            }
        } /* Inner for */
        if (session_id_vector.size() == 0) {
            session_id = buffer;
            if (DEBUG) {
                out_strm << "In 'generate_session_id': " << "session_id_vector" << endl
                    << "is empty." << endl << "'buffer'== " << buffer << endl << "'session_id'== "
                    << session_id << endl << "Breaking." << endl;
            } /* if (DEBUG) */
            break;
        } /* if */
        if (DEBUG) {
            out_strm << "'buffer'== " << buffer << endl;
        } /* if */
        pthread_mutex_lock(&session_data_mutex);
        iter = find(session_id_vector.begin(), session_id_vector.end(), string(buffer));
        if (iter != session_id_vector.end()) {
            out_strm << "WARNING! In 'generate_session_id': "
                << "Random_string found in 'session_id_vector'!" << endl <<
                "This is very unlikely to happen, so something" << "has probably gone wrong." <<
                endl << "'buffer'== " << buffer << endl << "*iter== " << *iter << endl <<
                "Iterating." << endl;
            pthread_mutex_unlock(&session_data_mutex);
            continue;
        }
    } else {
        session_id = buffer;
        if (DEBUG) {
            out_strm << "In 'generate_session_id': "
                << "Found_random_string not in 'session_id_vector'." << endl <<
                "This is what's supposed to happen." << endl << "'buffer'== " << buffer << endl <<
                "'session_id'== " << session_id << endl << "Breaking." << endl;
        } /* if (DEBUG) */
        pthread_mutex_unlock(&session_data_mutex);
    }
}

```

```

        break;
    } /* else */
} /* Outer for */
fclose(fp);
fp = 0;
Cookie_Type curr_cookie;
curr_cookie.name = "gwirdsif_session_id";
curr_cookie.value = session_id;
curr_cookie.expires_str = get_datestamp(2, 0, &curr_cookie.expires);
cookie_vector.push_back(curr_cookie);
return 0; } /* End of Cookie_Type::generate_session_id definition. */

```

58. Deleted expired cookies (*delete_expired_cookies*). [LDF 2012.07.24.]

Log

[LDF 2012.07.24.] Added this thread function.

[LDF 2012.11.16.] Now passing **unsigned int** *sleep_val* to this function using the **void *v** argument.

⟨ class **Cookie_Type** function declarations 35 ⟩ +≡
static void **delete_expired_cookies*(**void** **v*);

59.

⟨ **Cookie_Type**::*delete_expired_cookies* definition 59 ⟩ ≡
void ***Cookie_Type**::*delete_expired_cookies*(**void** **v*){ **bool** DEBUG = *false*; /* true */
set_debug_level(DEBUG, 0, 0);
#if DEBUG_COMPILE
if (DEBUG) {
pthread_mutex_lock(&log_strm_mutex);
log_strm ≪ "Entering 'delete_expired_cookies' ." ≪ *endl*;
pthread_mutex_unlock(&log_strm_mutex);
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
unsigned int sleep_val = ***static_cast**<**unsigned int** *>(*v*);
time_t now;
vector<**Cookie_Type**>::**iterator** iter;

See also sections 60, 61, 62, and 63.

This code is used in section 64.

60.

```
<Cookie_Type::delete_expired_cookies definition 59> +≡
  for ( ; ; ) /* Loop forever */
  { pthread_mutex_lock(&cookie_vector_mutex);
    if (cookie_vector.size() == 0) {
#ifndef DEBUG_COMPILE
      if (DEBUG) {
        pthread_mutex_lock(&log_strm_mutex);
        log_strm << "In 'delete_expired_cookies': " <<
          "'cookie_vector.size()' == 0. Going to sleep." << endl;
        pthread_mutex_unlock(&log_strm_mutex);
        pthread_mutex_unlock(&cookie_vector_mutex);
        goto DELETE_EXPIRED_COOKIES_SLEEP;
      } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    } /* if (cookie_vector.size() == 0) */
```

61. *cookie_vector* isn't empty. [LDF 2012.07.24.]

```
<Cookie_Type::delete_expired_cookies definition 59> +≡
  now = time(0);
  sort(cookie_vector.begin(), cookie_vector.end());
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    pthread_mutex_lock(&log_strm_mutex);
    log_strm << "In 'delete_expired_cookies': " <<
      "'cookie_vector.size()' == " <<
      cookie_vector.size() << endl << "Starting inner loop." << endl;
    pthread_mutex_unlock(&log_strm_mutex);
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  for (iter = cookie_vector.begin(); iter != cookie_vector.end(); ++iter) {
    if (iter->expires < now) {
#ifndef DEBUG_COMPILE
      if (DEBUG) {
        pthread_mutex_lock(&log_strm_mutex);
        log_strm << "In 'delete_expired_cookies': Expired cookie found. " <<
          "Will delete expired cookies." << endl;
        pthread_mutex_unlock(&log_strm_mutex);
      } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
      break;
    } /* if (iter->expires < now) */
  } /* inner for */
```

62.

```
<Cookie_Type::delete_expired_cookies definition 59> +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        pthread_mutex_lock(&log_strm_mutex);
        log_strm << "In 'delete_expired_cookies': After_inner_loop." << endl;
        pthread_mutex_unlock(&log_strm_mutex);
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

63.

```
<Cookie_Type::delete_expired_cookies definition 59> +≡
    if (iter != cookie_vector.end()) {
        cookie_vector.erase(iter, cookie_vector.end());
    #ifndef DEBUG_COMPILE
        if (DEBUG) {
            pthread_mutex_lock(&log_strm_mutex);
            log_strm << "In 'delete_expired_cookies': "
                  << "Deleted_expired_cookies." << endl <<
                  "'cookie_vector.size()' == " << cookie_vector.size() << endl;
            pthread_mutex_unlock(&log_strm_mutex);
        } /* if (DEBUG) */
    #endif /* DEBUG_COMPILE */
    } /* if (iter != cookie_vector.end()) */
    #ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            pthread_mutex_lock(&log_strm_mutex);
            log_strm << "In 'delete_expired_cookies': 'iter' == 'cookie_vector.end()' : "
                  << "No_expired_cookies. Not_deleting." << endl;
            pthread_mutex_unlock(&log_strm_mutex);
        } /* else if (DEBUG) */
    #endif /* DEBUG_COMPILE */
    pthread_mutex_unlock(&cookie_vector_mutex);
DELETE_EXPIRED_COOKIES_SLEEP: pthread_mutex_lock(&log_strm_mutex);
    log_strm << "In 'delete_expired_cookies': Going_to_sleep_for"
          << sleep_val << " seconds"
          << "(" << (sleep_val * 60) << " minutes")..." << endl;
    pthread_mutex_unlock(&log_strm_mutex);
    sleep(sleep_val);
    pthread_mutex_lock(&log_strm_mutex);
    log_strm << "In 'delete_expired_cookies': Woke_up." << endl;
    pthread_mutex_unlock(&log_strm_mutex);
    continue; } /* outer for */
return 0; /* Should never be reached. [LDF 2012.07.24.] */
} /* End of delete_expired_cookies definition */
```

64. Putting ‘ckietype.web’ together. [LDF 2010.10.06.]

```
⟨ Include files 3 ⟩  
using namespace std;  
using namespace gwrdifpk;  
⟨ class Cookie_Type declaration 34 ⟩  
⟨ Cookie_Type Global variables 32 ⟩  
⟨ class Cookie_Type constructor definitions 36 ⟩  
⟨ Cookie_Type::operator= definition 40 ⟩  
⟨ Cookie_Type::operator≡ definition 42 ⟩  
⟨ Cookie_Type::clear definition 46 ⟩  
⟨ Cookie_Type::show definition 48 ⟩  
⟨ Cookie_Type::parse_cookies definition 50 ⟩  
⟨ Cookie_Type::generate_session_id definition 54 ⟩  
⟨ Cookie_Type::delete_expired_cookies definition 59 ⟩
```

65. This is what’s written to the header file `ckietype.h`.

66.

```
⟨ ckietype.h 66 ⟩ ≡  
#ifndef CKIETYPE_H  
#define CKIETYPE_H 1  
using namespace std;  
using namespace gwrdifpk;  
⟨ class Cookie_Type declaration 34 ⟩  
⟨ Cookie_Type extern declarations for global variables 33 ⟩  
#endif
```

67. Utility functions for the FastCGI web application `gwrdbap.fcgi`. [LDF 2012.07.25.]

Log

[LDF 2012.07.25.] Added this file.

68. Include files. [LDF 2012.07.25.]

```
<Include files 3> +≡
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>
#include <time.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>
#include <iomanip>
#include <iostream>
#include <map>
#include <fstream>
#include <sstream>
#include <string>
#include <vector>
#include <deque>
#include <stack>
#include <set>
#include <sys/types.h>
#include <sys/wait.h> /* Library is /usr/lib64/libfcgi.a (on pcfinston LDF 2012.09.03.) */
#include <fcgi_stdio.h>
#include <pthread.h>
#include <gcrypt.h> /* for gcry_control */
#include <gnutls/gnutls.h>
#include <gnutls/x509.h>
#include <mysql.h>
#include <expat.h>
#include <curl/curl.h>
#undef NAME_LEN
#undef LOCAL_HOST
#ifndef HAVE_CONFIG_H
    # include <config.h >
#endif
#ifndef _XOPEN_SOURCE
#define _XOPEN_SOURCE
#endif
#include "glblcnst.h++"
#include "glblvrbl.h++"
#include "excptntp.h++"
#include "utilfncs.h++"
#include "grouptp.h++"
#include "hndlvltp.h++"
#include "irdsavtp.h++"
#include "rspnstp.h++"
#include "irdsobtp.h++"
```

```
#include "hdltype.h++"
#include "dcmtddtp.h++"
#include "x509cert.h++"
#include "dstngnmt.h++"
typedef int YYSTYPE;
#include "scprpmtp.h++"
#include "pidfncts.h++"
#include "tanfncts.h++"
#include "clntfncts.h++"
#include "ckietype.h++"
```

69. *parse_post_data*. [LDF 2012.07.25.]

Log

[LDF 2012.07.25.] Added this function.

[LDF 2012.07.27.] Added argument **string &out_filename** for storing the filename, if a temporary file is created.

[LDF 2012.07.27.]

[LDF 2012.07.27.] Added argument **Scan_Parser_Parameter_Type ¶m**. Removed the arguments **char *buffer**, **size_t buffer_size**, **ofstream &out_strm** and **string &out_filename**. Now using **Scan_Parser_Parameter_Type** data members instead.

⟨ parse_post_data declaration 69 ⟩ ≡

```
int parse_post_data(Scan_Parser_Parameter_Type &param);
```

This code is used in sections 75 and 76.

70.

⟨ parse_post_data definition 70 ⟩ ≡

```
int parse_post_data(Scan_Parser_Parameter_Type &param){ int status;
    bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    stringstream temp_strm;
    ofstream out_strm;
```

See also sections 71, 72, and 73.

This code is cited in section 74.

This code is used in section 75.

71. We shouldn't have to worry about *stdin* blocking, because this function will only be called if POST data is present. [LDF 2012.07.26.]

```

⟨ parse_post_data definition 70 ⟩ +≡
  bool file_open = false;
  char *unesaped_str = 0;
  char *res = 0;
  int char_ctr = 0;
  const size_t MAX_LINE = 1024;
  char line_buffer[MAX_LINE]; for ( ; ; ) { memset(line_buffer, 0, MAX_LINE);
  res = fgets(line_buffer, MAX_LINE, stdin);
  /* fread reads MAX_LINE - 1 characters. [LDF 2012.07.27.] */
  if (res ≡ 0 ∧ strlen(param.data_buffer) ≡ 0 ∧ file_open ≡ false) {
    temp_strm.str("");
    temp_strm ≪ "ERROR! In 'parse_post_data': "
    " 'fgets' returned 0 and 'strlen(param.data_buffer)' == 0 "
    " and 'file_open' == 'false'." ≪ endl ≪ "Failed to read any characters." ≪ endl ≪
    "Exiting function unsuccessfully with return value -1." ≪ endl;
    printf("%s", temp_strm.str().c_str());
    memset(param.data_buffer, 0, BUFFER_SIZE);
    return -1;
  }
  else if (res ≡ 0) {
    break;
  }
  res = 0;
  unescaped_str = curl_easy_unescape(curl, line_buffer, strlen(line_buffer), 0);
  if (unescaped_str ≡ 0) {
    temp_strm.str("");
    temp_strm ≪ "ERROR! In 'parse_post_data': "
    " 'curl_easy_unescape' failed, returning 0."
    "Exiting function unsuccessfully with return value -1." ≪ endl;
    printf("%s", temp_strm.str().c_str());
    memset(param.data_buffer, 0, BUFFER_SIZE);
    return -1;
  }
  else if (strlen(unescaped_str) ≡ 0) {
    temp_strm.str("");
    temp_strm ≪ "ERROR! 'curl_easy_unescape' returned an empty string." ≪ endl ≪
    "Exiting function unsuccessfully with return value -1." ≪ endl;
    printf("%s", temp_strm.str().c_str());
    memset(param.data_buffer, 0, BUFFER_SIZE);
    curl_free(unescaped_str);
    unescaped_str = 0;
    return -1;
  }
#endif DEBUG_COMPILE
  else
    if (DEBUG) {
      temp_strm.str("");
      temp_strm ≪ "'unescaped_str' == "
      " 'strlen(unescaped_str)' == "
      " strlen(unescaped_str) < ". ≪ endl;
    }
}

```

```
#endif /* DEBUG_COMPILE */
```

72.

```
<parse_post_data definition 70> += /* !! TODO: LDF 2012.12.13. In other places, I'm using
    BUFFER_SIZE instead of BUFFER_SIZE - 1. I may want to do this here, too. */
    if (file_open == false & strlen(unescaped_str) + strlen(param.data_buffer) > (BUFFER_SIZE - 1)) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        temp_strm.str("");
        temp_strm << "In 'parse_post_data': "
        << "'file_open' == 'false' and 'strlen(unes\
            caped_str)' <= "
        << "'strlen(param.data_buffer)' > ('BUFFER_SIZE' - 1): "
        << endl << "'strlen(unescaped_str)' == "
        << strlen(unescaped_str) << endl <<
        "'strlen(param.data_buffer)' == "
        << strlen(param.data_buffer) <<
        endl << "'BUFFER_SIZE' == "
        << BUFFER_SIZE << endl <<
        "Will open 'out_strm' and write 'unescaped_str' to it." << endl;
        printf("%s", temp_strm.str().c_str());
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    file_open = true;
    < Open output stream 74>
    out_strm << unescaped_str;
    memset(param.data_buffer, 0, BUFFER_SIZE);
} /* if */
else if (file_open == true) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        temp_strm.str("");
        temp_strm << "In 'parse_post_data': "
        << "'file_open' == 'true' ."
        << endl;
        printf("%s", temp_strm.str().c_str());
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    out_strm << unescaped_str;
} /* else if */
else if (file_open == false) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        temp_strm.str("");
        temp_strm << "In 'parse_post_data': "
        << "'file_open' == 'false' ."
        << endl;
        printf("%s", temp_strm.str().c_str());
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    strcat(param.data_buffer, unescaped_str);
} /* else if */
curl_free(unescaped_str);
unescaped_str = 0; } /* for */
```

73.

```
<parse_post_data definition 70> +≡
  if (file_open == false ∧ (param.data_buffer == 0 ∨ strlen(param.data_buffer) == 0)) {
    temp_strm.str("");
    temp_strm << "ERROR! In 'parse_post_data': 'file_open' == 'false' and "
    temp_strm << "'param.data_buffer' == 0 or 'strlen(param.data_buffer)' == 0." <<
    endl << "Failed to read POST data. This shouldn't happen." << endl <<
    "Exiting function 'parse_post_data' unsuccessfully with return value -1." << endl;
    printf("%s", temp_strm.str().c_str());
    memset(param.data_buffer, 0, BUFFER_SIZE);
    return -1;
  } /* if */
  else if (file_open) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    temp_strm.str("");
    temp_strm << "In 'parse_post_data': 'file_open' == 'true'. Closing 'out_strm'." <<
    endl << "Exiting 'parse_post_data' successfully with return value 1.";
    printf("%s", temp_strm.str().c_str());
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    out_strm.close();
    return 1;
  } /* if (out_strm.is_open()) */
  else {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    temp_strm.str("");
    temp_strm << "In 'parse_post_data': 'file_open' == 'false'." <<
    "POST data is in 'param.data_buffer': " << endl <<
    "'strlen(param.data_buffer)' == " << strlen(param.data_buffer) << endl <<
    "'param.data_buffer' == " << endl << param.data_buffer << endl <<
    "Exiting 'parse_post_data' successfully with return value 0." << endl;
    printf("%s", temp_strm.str().c_str());
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    out_strm.close();
    return 0;
} /* else */
} /* End of parse_post_data definition. */
```

74. Open output stream **ofstream** &*out strm*. [LDF 2012.07.26.]**Log**

[LDF 2012.07.26.] Added this section. It's used in *(parse_post_data* definition 70).

```

⟨ Open output stream 74 ⟩ ≡
{
    int fd;
    char temp_filename[] = "/tmp/gwrdbap.XXXXXX";
    errno = 0;
    fd = mkostemp(temp_filename, S_IRWXU | S_IRGRP | S_IROTH);
    if (fd == -1) {
        temp_strm.str("");
        temp_strm << "ERROR! In '⟨Open_output_stream⟩': 'mkostemp' failed, re\
            turning -1." << endl << "Failed to open temporary file." <<
            endl << "mkostemp error: " << strerror(errno) << endl <<
            "Exiting function 'parse_post_data' unsuccessfully with return value -1." << endl;
        printf("%s", temp_strm.str().c_str());
        return -1;
    } /* if (fd == -1) */
    else if (DEBUG) {
        temp_strm.str("");
        temp_strm << "In '⟨Open_output_stream⟩': 'mkostemp' succeeded. " <<
            "'temp_filename' == " << temp_filename << endl;
        printf("%s", temp_strm.str().c_str());
    } /* else if (DEBUG) */
    errno = 0;
    status = fchmod(fd, S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH);
    if (status == -1) {
        temp_strm.str("");
        temp_strm << "ERROR! In '⟨Open_output_stream⟩': 'fchmod' failed, returning -1." <<
            endl << "Failed to change mode of temporary file." <<
            endl << "fchmod error: " << strerror(errno) << endl <<
            "Exiting function 'parse_post_data' unsuccessfully with return value -1." << endl;
        printf("%s", temp_strm.str().c_str());
        close(fd);
        return -1;
    } /* if (status == -1) */
    close(fd); /* We just need the name. [LDF 2012.07.03.] */
    fd = 0;
    out_strm.open(temp_filename);
    if (!out_strm & !out_strm.is_open()) {
        temp_strm.str("");
        temp_strm << "ERROR! In '⟨Open_output_stream⟩': Failed to open 'out_strm'." <<
            endl << "Exiting function 'parse_post_data' unsuccessfully with retu\
            rn value -1." << endl;
        printf("%s", temp_strm.str().c_str());
        return -1;
    } /* !out_strm */
#endif DEBUG_COMPILE
else
    if (DEBUG) {

```

```

    temp_strm.str("");
    temp_strm << "In 'Open<output>': Opened 'out_strm' successfully." <<
        endl;
    printf("%s", temp_strm.str().c_str());
}
/* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
param.data_filename = temp_filename;
} /* End of section {Open output stream 74} */

```

This code is cited in section 74.

This code is used in section 72.

75. This is what's compiled. [LDF 2012.06.28.]

!! PLEASE NOTE: The **typedefs** for *jobject*, *JNIEnv* and **yyscan_t** below are needed for types used in class **Scan_Parse_Parameter_Type** or its member functions and **int yydebug** is used in *process_command_line_options* in *cmdlnopt.web*. If I add a scanner/parser pair for this program, I will need to get rid of the **typedef** for **yyscan_t** and the declaration of **int yydebug**. [LDF 2012.07.20.]

```

typedef void *yyscan_t;
<Include files 3>
using namespace std;
using namespace gwrdifpk;
extern CURL*curl;
extern CURLcode curl_res;
<parse_post_data declaration 69>
<parse_post_data definition 70>

```

76. This is what's written to the header file *utlwbfcs.h*. [LDF 2008.12.05.]

```

<utlwbfcs.h 76>≡
#ifndef UTLWBFCS_H
#define UTLWBFCS_H 1
using namespace std;
using namespace gwrdifpk;
<parse_post_data declaration 69>
#endif

```

77. Index.

<i>_XOPEN_SOURCE</i> :	<u>3</u> , <u>31</u> , <u>68</u> .	<i>BUFFER_SIZE</i> :	<u>18</u> , <u>71</u> , <u>72</u> , <u>73</u> .
<i>admin_data</i> :	<u>6</u> .	<i>buffer_size</i> :	<u>69</u> .
<i>admin_data_length</i> :	<u>6</u> .	<i>c</i> :	<u>8</u> , <u>37</u> , <u>38</u> , <u>39</u> , <u>40</u> , <u>41</u> , <u>42</u> , <u>43</u> , <u>44</u> , <u>57</u> .
<i>anoncred</i> :	<u>29</u> .	<i>c_str</i> :	<u>6</u> , <u>10</u> , <u>11</u> , <u>13</u> , <u>14</u> , <u>15</u> , <u>16</u> , <u>17</u> , <u>18</u> , <u>20</u> , <u>21</u> , <u>22</u> ,
<i>app</i> :	<u>6</u> .		<u>23</u> , <u>24</u> , <u>26</u> , <u>27</u> , <u>48</u> , <u>50</u> , <u>71</u> , <u>72</u> , <u>73</u> , <u>74</u> .
<i>arg_str_vector</i> :	<u>6</u> .	<i>cerr_mutex</i> :	<u>5</u> , <u>6</u> .
<i>argc</i> :	<u>6</u> .	<i>char_ctr</i> :	<u>8</u> , <u>71</u> .
<i>args</i> :	<u>6</u> .	<i>chmod</i> :	<u>6</u> .
<i>argv</i> :	<u>6</u> .	<i>CKIETYPE_H</i> :	<u>66</u> .
<i>atemp_str</i> :	<u>18</u> .	<i>clear</i> :	<u>40</u> , <u>45</u> , <u>46</u> .
<i>atemp_strm</i> :	<u>8</u> , <u>9</u> , <u>10</u> .	<i>client_dn</i> :	<u>8</u> , <u>14</u> , <u>18</u> .
<i>atexit</i> :	<u>6</u> .	<i>client_func</i> :	<u>6</u> , <u>27</u> .
<i>attr</i> :	<u>7</u> .	<i>close</i> :	<u>28</u> , <u>73</u> , <u>74</u> .
<i>begin</i> :	<u>10</u> , <u>38</u> , <u>40</u> , <u>48</u> , <u>51</u> , <u>56</u> , <u>57</u> , <u>61</u> .	<i>compare</i> :	<u>17</u> .
<i>buffer</i> :	<u>57</u> , <u>69</u> .	<i>config</i> :	<u>3</u> , <u>68</u> .

const_iterator: 10, 38, 40, 48, 56, 57.
CONTENT_TYPE: 17.
content_type: 8, 17.
cookie_str: 50, 51.
Cookie_Type: 6, 7, 9, 10, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 46, 48, 49, 50, 51, 52, 53, 54, 57, 59.
cookie_vector: 10, 32, 33, 34, 51, 57, 60, 61, 63.
cookie_vector_mutex: 5, 6, 32, 33, 60, 63.
curl: 4, 6, 28, 71, 75.
CURL: 4, 75.
curl_easy_cleanup: 28.
curl_easy_init: 6.
curl_easy_unescape: 71.
curl_free: 71, 72.
curl_res: 4, 75.
CURLcode: 4, 75.
curr_cookie: 50, 51, 57.
curr_var: 11.
data_buffer: 18, 21, 22, 71, 72, 73.
data_filename: 74.
DEBUG: 5, 6, 7, 9, 13, 14, 15, 16, 17, 18, 21, 23, 24, 26, 27, 50, 51, 52, 54, 55, 56, 57, 59, 60, 61, 62, 63, 70, 71, 72, 73, 74.
DEBUG_COMPILE: 6, 7, 9, 18, 21, 23, 24, 26, 50, 51, 52, 57, 59, 60, 61, 62, 63, 71, 72, 73, 74.
delete_expired_cookies: 6, 7, 58, 59, 63.
DELETE_EXPIRED_COOKIES_SLEEP: 60, 63.
delete_expired_cookies_thread_id: 7.
Distinguished_Name: 14, 18.
domain: 34, 36, 38, 40, 42, 46, 48.
empty: 9, 10, 13, 50, 51.
end: 10, 18, 38, 40, 48, 51, 56, 57, 61, 63.
END_OF_MAIN_ACCEPT_LOOP: 14, 18, 20, 22, 26, 27, 28.
END_PROCESS_POST_DATA: 16, 17, 24.
endl: 6, 7, 8, 9, 10, 11, 13, 14, 15, 16, 17, 18, 20, 21, 22, 23, 24, 26, 27, 48, 50, 51, 55, 56, 57, 59, 60, 61, 62, 63, 71, 72, 73, 74.
environ: 4, 11.
erase: 18, 20, 22, 27, 50, 63.
errno: 55, 74.
exit: 6, 7, 28.
expires: 18, 34, 36, 38, 40, 44, 46, 48, 51, 57, 61.
expires_str: 10, 34, 36, 38, 40, 46, 48, 51, 57.
false: 5, 6, 8, 50, 54, 59, 70, 71, 72, 73.
FCGL_Accept: 8, 10.
fchmod: 74.
fclose: 57.
fd: 74.
fgetc: 57.
fgets: 71.
file_open: 71, 72, 73.
find: 18, 50, 51, 57.
finish: 5, 6.
first: 48.
flush: 6.
fopen: 55.
fp: 54, 55, 57.
fprintf: 6.
fread: 71.
gcry_control: 3, 68.
GCRY_THREAD_OPTION_PTHREAD_IMPL: 29.
generate_session_id: 9, 53, 54, 57.
get_data_present: 8, 13, 26.
get_datestamp: 8, 51, 57.
get_expires: 18.
getenv: 9, 12, 13, 14, 15, 16, 17.
gnutls_anon_server_credentials_t: 29.
gpg_key_id: 5.
gpg_passphrase: 5.
gpg_passphrase_length: 5.
gwrdifpk: 29, 64, 66, 75, 76.
HAVE_CONFIG_H: 3, 68.
HTTP_COOKIE: 9, 51.
http_cookie_str: 49, 50.
HTTP_USER_AGENT: 12.
i: 57.
ios_base: 6.
is_gwrdwbap: 6.
is_open: 6, 73, 74.
isalnum: 57.
iter: 10, 38, 40, 48, 51, 56, 57, 59, 61, 63.
iterator: 8, 51, 59.
j: 6.
JNIEnv: 29, 75.
jobject: 29, 75.
k: 11.
length: 18.
line_buffer: 71.
LOCAL_HOST: 3, 68.
log_filename: 6.
log_strm: 6, 7, 8, 28, 59, 60, 61, 62, 63.
log_strm_mutex: 5, 6, 59, 60, 61, 62, 63.
loop_ctrl: 6, 8, 10.
LtpaToken: 49.
m: 13.
main: 6, 28, 34.
map: 6, 8.
MAX_LINE: 71.
memset: 6, 57, 71, 72, 73.
mkostemp: 74.
munlock: 5.
name: 10, 34, 36, 38, 40, 42, 46, 48, 50, 51, 57.

NAME_LEN: 3, 68.
name_value_pairs: 34, 38, 40, 46, 48.
NOTE: 29, 75.
now: 59, 61.
npos: 50.
ofstream: 69, 70, 74.
open: 6, 74.
out_filename: 69.
out_strm: 24, 49, 50, 51, 52, 53, 54, 55, 56, 57, 69, 70, 72, 73, 74.
outer_status: 6, 7.
outer_temp_strm: 6.
pair: 13, 34, 38, 40, 48.
param: 8, 18, 19, 20, 21, 22, 27, 34, 69, 70, 71, 72, 73, 74.
parameters: 13.
parse_cookies: 9, 49, 50, 52.
parse_post_data: 18, 19, 20, 21, 24, 69, 70, 73.
path: 34, 36, 38, 40, 42, 46, 48.
post_data_present: 8, 23, 24, 26.
printf: 10, 11, 13, 14, 15, 16, 17, 18, 20, 21, 22, 23, 24, 26, 27, 48, 71, 72, 73, 74.
process_command_line_options: 29, 75.
pthread_attr_init: 7.
pthread_attr_setdetachstate: 7.
pthread_attr_t: 7.
pthread_create: 7.
PTHREAD_CREATE_DETACHED: 7.
pthread_mutex_destroy: 5.
pthread_mutex_init: 6.
pthread_mutex_lock: 18, 51, 57, 59, 60, 61, 62, 63.
pthread_mutex_t: 32, 33.
pthread_mutex_unlock: 18, 51, 57, 59, 60, 61, 62, 63.
pthread_t: 7.
push_back: 38, 40, 51, 57.
query_string: 13.
QUERY_STRING: 13.
REQUEST_METHOD: 16.
res: 71.
s: 47, 48.
S_IROTH: 74.
S_IRUSR: 74.
S_IWUSR: 74.
Scan_Parse_Parameter_Type: 6, 8, 14, 18, 29, 34, 69, 70, 75.
second: 18, 48.
session_data_iter: 8, 18, 20, 22, 27.
session_data_map: 6, 18, 20, 22, 27.
session_data_mutex: 5, 6, 18, 51, 57.
session_id: 8, 9, 18, 49, 50, 51, 53, 54, 57.
session_id_vector: 56, 57.
set_debug_level: 5, 6, 50, 54, 59, 70.
set_expires: 18.
show: 10, 47, 48.
size: 10, 17, 48, 50, 56, 57, 60, 61, 63.
size_type: 50.
sleep: 63.
sleep_val: 7, 58, 59, 63.
sock_path: 4.
sort: 61.
SSL_CLIENT_CERT: 14.
SSL_CLIENT_S_DN: 14.
status: 5, 8, 9, 17, 19, 20, 21, 22, 23, 24, 27, 54, 70, 74.
std: 29, 64, 66, 75, 76.
stderr: 6.
stdin: 71.
str: 6, 9, 10, 11, 13, 14, 15, 16, 17, 18, 20, 21, 22, 23, 24, 26, 27, 48, 56, 71, 72, 73, 74.
strcat: 18, 72.
strcmp: 16.
strcpy: 18.
strerror: 55, 74.
string: 4, 6, 8, 13, 34, 38, 40, 47, 48, 49, 50, 53, 54, 56, 57, 69.
stringstream: 6, 8, 47, 48, 49, 50, 53, 54, 70.
strlen: 9, 16, 18, 22, 71, 72, 73.
t: 47, 48.
temp_buff: 8, 9, 10, 12, 14, 15, 16, 17.
temp_filename: 74.
temp_str: 8, 17.
temp_strm: 8, 10, 11, 13, 14, 15, 16, 17, 18, 20, 21, 22, 23, 24, 26, 27, 48, 70, 71, 72, 73, 74.
temp_val: 6.
time: 8, 61.
TODO: 26, 51.
true: 5, 6, 13, 23, 24, 50, 54, 59, 70, 72.
typedefs: 29, 75.
unescape_str: 8, 71, 72.
UTLWBFCS_H: 76.
v: 58, 59.
value: 10, 34, 36, 38, 40, 42, 46, 48, 50, 51, 57.
vector: 10, 13, 32, 33, 34, 38, 40, 48, 51, 56, 57, 59.
yydebug: 29, 75.
yyscan_t: 29, 75.
YYSTYPE: 3, 68.

⟨ Global variables 4 ⟩ Used in section 29.
⟨ Include files 3, 31, 68 ⟩ Used in sections 29, 64, and 75.
⟨ Open output stream 74 ⟩ Cited in section 74. Used in section 72.
⟨ ckietype.h 66 ⟩
⟨ utlwbfc.h 76 ⟩
⟨ **Cookie_Type**::*clear* definition 46 ⟩ Used in section 64.
⟨ **Cookie_Type**::*delete_expired_cookies* definition 59, 60, 61, 62, 63 ⟩ Used in section 64.
⟨ **Cookie_Type**::*generate_session_id* definition 54, 55, 56, 57 ⟩ Used in section 64.
⟨ **Cookie_Type**::**operator** \equiv definition 42, 44 ⟩ Used in section 64.
⟨ **Cookie_Type**::**operator** $=$ definition 40 ⟩ Used in section 64.
⟨ **Cookie_Type**::*parse_cookies* definition 50, 51, 52 ⟩ Used in section 64.
⟨ **Cookie_Type**::*show* definition 48 ⟩ Used in section 64.
⟨ **Cookie_Type** Global variables 32 ⟩ Used in section 64.
⟨ **Cookie_Type** extern declarations for global variables 33 ⟩ Used in section 66.
⟨ class **Cookie_Type** constructor definitions 36, 38 ⟩ Used in section 64.
⟨ class **Cookie_Type** declaration 34 ⟩ Used in sections 64 and 66.
⟨ class **Cookie_Type** function declarations 35, 37, 39, 41, 43, 45, 47, 49, 53, 58 ⟩ Used in section 34.
⟨ *finish* definition 5 ⟩ Used in section 29.
⟨ *main* definition 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28 ⟩ Used in section 29.
⟨ *parse_post_data* declaration 69 ⟩ Used in sections 75 and 76.
⟨ *parse_post_data* definition 70, 71, 72, 73 ⟩ Cited in section 74. Used in section 75.

GWDG iRODS/Handle Interface Client/Server Application IV: Version 1.0

by Laurence D. Finston

September 2013

	Section	Page
GWDG iRODS/Handle Interface Client-Server Application	1	1
FastCGI Web Application gwrdbap	2	1
Cookie_Type	30	17
Utility functions for the FastCGI web application <code>gwrdbap.fcgi</code>	67	30
Index	77	37

1. GWDG iRODS/Handle Interface Client-Server Application.**2. Set up databases.** [LDF 2013.09.02.]

This query can be used to check what tables are created in database YYY:

```
"select\u00a0table_name\u00a0from\u00a0tables\u00a0where\u00a0table_schema\u00a0=\u00a0'YYY'\u00a0order\u00a0by\u00a0table_name"
```

[LDF 2013.09.05.]

3. Include files. [LDF 2013.09.02.]

```
<Include files 3>≡
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>    /* POSIX threads */
#include <gcrypt.h>      /* for gcry_control */
#include <gnutls/gnutls.h>
#include <gnutls/x509.h>
#include <mysql.h>
#include <expat.h>
#include <iomanip>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <getopt.h>
#include <map>
#include <vector>
#include <deque>
#include <set>
#include <stack>
#include <utility>
#if HAVE_CONFIG_H
#include <config.h>
#endif
#include "rspercds.h++"
#include "glblcnst.h++"
#include "glblvrbl.h++"
#include "excptntp.h++"
#include "utilfncts.h++"
#include "grouptp.h++"
#include "hndlvltp.h++"
#include "irdsavtp.h++"
#include "helper.h++"
#include "tanfncts.h++"
#include "pidfncts.h++"
#include "parser.h++"
#include "scanner.h++"
#include "rspnstp.h++"
#include "irdsobtp.h++"
#include "hdltype.h++"
#include "dcmtddtp.h++"
#include "x509cert.h++"
#include "dstngnmt.h++"
#include "usrinftp.h++"
#include "scprpmtp.h++"
#include "stpclopt.h++"
#include "stpcdsif.h++"
```

```
#include "stparobj.h++"
#include "stpdblcr.h++"
#include "stpcdcli.h++"
#include "stpcdhdl.h++"
#include "stpcrdbs.h++"
```

See also sections 18, 26, 34, 67, and 95.

This code is used in sections 17, 24, 32, 65, 93, and 110.

4. Finish (*finish*). [LDF 2013.09.02.]

```
<finish definition 4> ≡
void finish(void)
{
    if (out_strm.is_open()) out_strm.close();
    return;
}
```

This code is used in section 17.

5. Main. [LDF 2013.09.02.]

```
<Main 5> ≡
int main(int argc, char *argv[]){ bool DEBUG = false; /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    char *temp_ptr = 0;
    char buffer[1024];
    memset(buffer, 0, 1024);
    cerr << "Entering 'setup dbs' ('main')." << endl;
    status = atexit(finish);
    if (status ≠ 0) {
        cerr << "[setup dbs] ERROR! In 'main': Cannot set exit function 'finish'." <<
            endl << "Exiting 'setup dbs' with exit status 1." << endl;
        exit(1);
    }
#if DEBUG_COMPILE
    else
        if (DEBUG) {
            cerr << "[setup dbs] In 'main': Set exit function 'finish' successfully." << endl;
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

See also sections 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, and 16.

This code is used in section 17.

6.

```
<Main 5> +≡
status = process_command_line_options(argc, argv);
if (status ≠ 0) {
    cerr ≪ "[setupdbs] ↴ERROR! ↴In ↴‘main’: ↴"
    "‘process_command_line_options’ ↴failed, ↴returning" ≪ status ≪ endl ≪
    "Exiting ↴‘setupdbs’ ↴unsuccessfully ↴with ↴exit ↴status ↴1." ≪ endl;
    exit(1);
} /* if (status ≠ 0) */
#if DEBUG_COMPILE
else
if (DEBUG) {
    cerr ≪ "[setupdbs] ↴In ↴‘main’: ↴"
    "‘process_command_line_options’ ↴succeeded\"
    "d, ↴returning ↴0." ≪ endl;
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

7.

```

⟨ Main 5 ⟩ +≡
    license_strm.open("./cpyrtc.txt");
    if (¬license_strm.is_open()) {
        cerr << "[setupdbs] ERROR! In 'main': " << "Failed to open file './cpyrtc.txt' for \
            input." << endl << "Exiting 'setupdbs' unsuccessfully with exit status 1." << endl;
        exit(1);
    } /* if (¬license_strm.is_open()) */
#endif /* DEBUG_COMPILE
else
if (DEBUG) {
    cerr << "[setupdbs] In 'main': " << "Opened file './cpyrtc.txt' for output successfully." << endl;
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
out_strm.open("./setupdbs_output.sql", ios_base::trunc);
if (¬out_strm.is_open()) {
    cerr << "[setupdbs] ERROR! In 'main': " <<
        "Failed to open file './setupdbs_output.sql' for output." << endl <<
        "Exiting 'setupdbs' unsuccessfully with exit status 1." << endl;
    exit(1);
} /* if (¬out_strm.is_open()) */
#endif /* DEBUG_COMPILE
else
if (DEBUG) {
    cerr << "[setupdbs] In 'main': " << "Opened file './setupdbs_output.sql' for \
        output successfully." << endl;
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
memset(buffer, 0, 1024);
temp_ptr = getenv("PWD");
if (strlen(temp_ptr) < 1024) {
    strcpy(buffer, temp_ptr);
    strcat(buffer, "/");
}
temp_ptr = 0;
out_strm << /* " */ << buffer << "setupdbs_output.sql" << endl << "Generated" <<
    get_datestamp() << " from 'setupdbs' */" << endl << endl;
while (license_strm.is_open() ∧ license_strm.good() ∧ ¬license_strm.eof()) {
    memset(buffer, 0, 1024);
    license_strm.read(buffer, 1023);
    out_strm << buffer;
}
license_strm.close();
out_strm << static_cast<char>(12) << endl << endl << /* (1) */ << endl << endl;

```

8.

```

⟨ Main 5 ⟩ +≡
  if (prefix_vector.size() == 0) {
    cerr << "[setupdbs] NOTICE: No prefixes specified with the '--prefix' option." <<
        endl << "Will set up default prefix '12345'." << endl;
    prefix_vector.push_back(string("12345"));
  }
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    cerr << "prefix_vector.size() == " << prefix_vector.size() << endl;
    for (vector<string>::iterator iter = prefix_vector.begin(); iter != prefix_vector.end(); ++iter) {
      cerr << *iter << endl;
    }
    cerr << endl;
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

9.

```

⟨ Main 5 ⟩ +≡
Scan_Parse_Parameter_Type param;
/* This takes care of setting up the connection to the MySQL client. [LDF 2013.09.04.] */
status = create_databases(param);
if (status != 0) {
  cerr << "[setupdbs] ERROR! In 'main': " << "'create_databases' failed, returning" <<
      status << endl << "Exiting 'setupdbs' unsuccessfully with exit status" << status <<
      "." << endl;
  exit(status);
} /* if (status != 0) */
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      cerr << "[setupdbs] In 'main': " << "'create_databases' succeeded, returning 0." <<
          endl;
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

10. Handles database (default: handlesystem or handlesystem_standalone). [LDF 2013.09.04.]

```
(Main 5) +≡
if (handles_database_created ≡ true) {
    cerr ≪ "[setupdbs] In 'main': 'handles_database_created' == 'true'." ≪
        "Will_create_tables_in" ≪ " " ≪ handles_database_name ≪ "'_database.'" ≪ endl;
    status = create_tables_handles(param);
    if (status ≠ 0) {
        cerr ≪ "[setupdbs] ERROR! In 'main': 'create_tables_handles' failed, "
            "returning" ≪ status ≪ "." ≪ endl ≪ "Failed_to_create_tables_in" ≪
                " " ≪ handles_database_name ≪ "'_database.'" ≪ endl ≪
                    "Exiting 'setupdbs' unsuccessfully with exit status 1." ≪ endl;
        exit(1);
    } /* if (status ≠ 0) */
    else {
        cerr ≪ "[setupdbs] In 'main': 'create_tables_handles' succeeded, "
            "returning 0." ≪ endl ≪ "Created_tables_in" ≪ " " ≪ handles_database_name ≪
                "'_database_successfully.'" ≪ endl;
    }
} /* if (handles_database_created ≡ true) */
else {
    cerr ≪ "[setupdbs] In 'main': 'handles_database_created' == 'false'." ≪
        "Not_creating_tables_in" ≪ " " ≪ handles_database_name ≪ "'_database.'" ≪ endl;
} /* else */
```

11. Server-side database (gwirdsif). [LDF 2013.09.04.]

```
(Main 5) +≡
if (gwirdsif_database_created ≡ true) { cerr ≪ "[setupdbs] In 'main': 'gwirdsif_database_created' == 'true'." ≪
    "Will_create_tables_in" ≪ " " ≪ gwirdsif_database_name ≪ "'_database.'" ≪ endl;
    status = create_tables_gwirdsif(param);
    if (status ≠ 0) {
        cerr ≪ "[setupdbs] ERROR! In 'main': 'create_tables_gwirdsif' failed, "
            "returning" ≪ status ≪ "." ≪ endl ≪ "Failed_to_create_tables_in" ≪
                " " ≪ gwirdsif_database_name ≪ "'_database.'" ≪ endl ≪
                    "Exiting 'setupdbs' unsuccessfully with exit status 1." ≪ endl;
        exit(1);
    } /* if (status ≠ 0) */
    else {
        cerr ≪ "[setupdbs] In 'main': 'create_tables_gwirdsif' succeeded, "
            "returning 0." ≪ endl ≪ "Created_tables_in" ≪ " " ≪ gwirdsif_database_name ≪
                "'_database_successfully.'" ≪ endl;
    }
}
```

12.

```
(Main 5) +≡
status = create_tables_archive(param);
if (status ≠ 0) {
    cerr ≪ "[setupdbs] In 'main': 'create_tables_archive' failed, " ≪
        "returning " ≪ status ≪ ". " ≪ endl ≪ "Failed to create tables for archive obj\"
        ects in " ≪ " " ≪ gwirdsif_database_name ≪ "' database." ≪ endl ≪
        "Exiting 'setupdbs' unsuccessfully with exit status 1." ≪ endl;
    exit(1);
} /* if (status ≠ 0) */
else {
    cerr ≪ "[setupdbs] In 'main': 'create_tables_archive' succeeded, " ≪ "returning 0. " ≪
        endl ≪ "Created tables for archive objects in " ≪ " " ≪ gwirdsif_database_name ≪
        "' database successfully." ≪ endl;
}
```

13.

Log

[LDF 2013.09.06.] Added this section.

```
(Main 5) +≡
status = create_tables_dublin_core(param);
if (status ≠ 0) {
    cerr ≪ "[setupdbs] In 'main': 'create_tables_dublin_core' failed, " ≪
        "returning " ≪ status ≪ ". " ≪ endl ≪ "Failed to create tables for archive obj\"
        ects in " ≪ " " ≪ gwirdsif_database_name ≪ "' database." ≪ endl ≪
        "Exiting 'setupdbs' unsuccessfully with exit status 1." ≪ endl;
    exit(1);
} /* if (status ≠ 0) */
else {
    cerr ≪ "[setupdbs] In 'main': 'create_tables_dublin_core' succeeded, " ≪
        "returning 0. " ≪ endl ≪ "Created tables for archive objects in " ≪ " " ≪ gwirdsif_database_name ≪
        "' database successfully." ≪ endl;
}
/* if (gwirdsif_database_created ≡ true) */
else {
    cerr ≪ "[setupdbs] In 'main': 'gwirdsif_database_created' == 'false' . " ≪
        "Not creating tables in " ≪ " " ≪ gwirdsif_database_name ≪ "' database." ≪ endl;
}
/* else */
```

14. Client-side database (gwirdcli). [LDF 2013.09.04.]

```
(Main 5) +≡
if (gwirdcli_database_created == true) {
    cerr << "[setupdbs] In 'main': gwirdcli_database_created' == 'true'." <<
        "Will create tables in " << "\"" << gwirdcli_database_name << "' database." << endl;
    status = create_tables_gwirdcli(param);
    if (status != 0) {
        cerr << "[setupdbs] ERROR! In 'main': 'create_tables_gwirdcli' failed, " <<
            "returning " << status << "." << endl << "Failed to create tables in " <<
            "\"" << gwirdcli_database_name << "' database." << endl <<
            "Exiting 'setupdbs' unsuccessfully with exit status 1." << endl;
        exit(1);
    } /* if (status != 0) */
    else {
        cerr << "[setupdbs] In 'main': 'create_tables_gwirdcli' succeeded, " <<
            "returning 0." << endl << "Created tables in " << "\"" << gwirdcli_database_name <<
            "' database successfully." << endl;
    }
} /* if (gwirdcli_database_created == true) */
else {
    cerr << "[setupdbs] In 'main': 'gwirdcli_database_created' == 'false'." <<
        "Not creating tables in " << "\"" << gwirdcli_database_name << "' database." << endl;
} /* else */
```

15.

```
(Main 5) +≡
```

16. Exit successfully. [LDF 2013.09.02.]

```
(Main 5) +≡
cerr << "Exiting 'setupdbs' successfully with exit status 0." << endl;
exit(0); /* End of main definition */
```

17. This is what's compiled. [LDF 2013.09.03.]

```
using namespace std;
< Include files 3>
using namespace gwrdifpk;
< finish definition 4>
(Main 5)
```

18. Include files. [LDF 2013.09.04.]

```
<Include files 3> +≡
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h> /* POSIX threads */
#include <gcrypt.h> /* for gcry_control */
#include <gnutls/gnutls.h>
#include <gnutls/x509.h>
#include <mysql.h>
#include <expat.h>
#include <iomanip>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <getopt.h>
#include <map>
#include <vector>
#include <deque>
#include <set>
#include <stack>
#include <utility>
#if HAVE_CONFIG_H
#include <config.h>
#endif
#include "rspercds.h++"
#include "glblcnst.h++"
#include "glblvrbl.h++"
#include "excptntp.h++"
#include "utilfncts.h++"
#include "grouptp.h++"
#include "hndlvltp.h++"
#include "irdsavtp.h++"
#include "helper.h++"
#include "tanfncts.h++"
#include "pidfncts.h++"
#include "parser.h++"
#include "scanner.h++"
#include "rspnstp.h++"
#include "irdsobtp.h++"
#include "hdltype.h++"
#include "dcmtddtp.h++"
#include "x509cert.h++"
#include "dstngnmt.h++"
#include "usrinftp.h++"
#include "scprpmtp.h++"
#include "stpclopt.h++"
```

19. Create client-side database tables (gwirdcli). [LDF 2013.09.04.]

```
< create_tables_gwirdcli declaration 19 > ≡
int create_tables_gwirdcli(Scan_Parse_Parameter_Type &param);
```

This code is used in section 25.

20.

```
< create_tables_gwirdcli definition 20 > ≡
int create_tables_gwirdcli(Scan_Parse_Parameter_Type &param){ bool DEBUG = false;
    /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    stringstream sql_strm;
    stringstream temp_strm;
    MYSQL_RES * result = 0;
    MYSQL_ROW curr_row;
    size_t result_array_size = 16;
    MYSQL_RES * result_array[16] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
    unsigned int row_ctr = 0;
    unsigned int field_ctr = 0;
    long int affected_rows = 0;
    vector<string> query_vector;
    vector<unsigned int *> row_ctr_vector;
    vector<unsigned int *> field_ctr_vector;
    vector<long int *> affected_rows_vector;
    vector<string> comment_vector;
    string comma_str;
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        cerr << "Entering 'create_tables_gwirdcli'." << endl;
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    sql_strm << "use" << gwirdcli_database_name;
    query_vector.push_back(sql_strm.str());
    row_ctr_vector.push_back(0);
    field_ctr_vector.push_back(0);
    affected_rows_vector.push_back(0);
```

See also sections 21, 22, and 23.

This code is used in section 24.

21.

```

⟨ create_tables_gwirdcli definition 20 ⟩ +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        cerr << "nas_table_create_str" << nas_table_create_str << endl <<
        "nas_table_insert_str" << nas_table_insert_str << endl <<
        "handles_table_create_str" << handles_table_create_str << endl <<
        "handles_table_alter_str[0]" << handles_table_alter_str[0] <<
        endl << "handles_table_alter_str[1]" << handles_table_alter_str[1] <<
        endl << "Users_table_create_str" << Users_table_create_str << endl <<
        "Users_table_insert_str" << Users_table_insert_str << endl;
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
query_vector.push_back(nas_table_create_str);
row_ctr_vector.push_back(0);
field_ctr_vector.push_back(0);
affected_rows_vector.push_back(0);
query_vector.push_back(nas_table_insert_str);
row_ctr_vector.push_back(0);
field_ctr_vector.push_back(0);
affected_rows_vector.push_back(new long int(0L));
comment_vector.push_back("insert_nas");
sql_strm.str("");
query_vector.push_back(handles_table_create_str);
row_ctr_vector.push_back(0);
field_ctr_vector.push_back(0);
affected_rows_vector.push_back(0);
query_vector.push_back(handles_table_alter_str[0]);
row_ctr_vector.push_back(0);
field_ctr_vector.push_back(0);
affected_rows_vector.push_back(0);
query_vector.push_back(handles_table_alter_str[1]);
row_ctr_vector.push_back(0);
field_ctr_vector.push_back(0);
affected_rows_vector.push_back(0);
query_vector.push_back(Users_table_create_str);
row_ctr_vector.push_back(0);
field_ctr_vector.push_back(0);
affected_rows_vector.push_back(0);
query_vector.push_back(Users_table_insert_str);
row_ctr_vector.push_back(0);
field_ctr_vector.push_back(0);
affected_rows_vector.push_back(0);
affected_rows_vector.push_back(new long int(0L));
comment_vector.push_back("insert_Users");
sql_strm.str("");
#endif DEBUG_COMPILE
    if (DEBUG) {
        cerr << "query_vector.size()" << query_vector.size() << endl;
        for (vector<string>::iterator iter = query_vector.begin(); iter != query_vector.end(); ++iter) {
            cerr << *iter << endl;
        } /* for */
        cerr << endl;
    }

```

```

    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
status = param.submit_mysql_queries(query_vector, result_array, row_ctr_vector, field_ctr_vector,
                                    affected_rows_vector, false);
if (status != 0) {
    cerr << "ERROR! In 'create_tables_gwirdcli': 'submit_mysql_queries' failed, "
    << "returning " << status << endl << "Exiting 'create_tables_gwirdcli' unsuccessfully with return value 1." << endl;
delete_and_clear(result_array, result_array_size, row_ctr_vector, field_ctr_vector, affected_rows_vector,
                 query_vector);
return 1;
} /* if (status != 0) */

```

22.

```

⟨ create_tables_gwirdcli definition 20 ⟩ +≡
#ifndef DEBUG_COMPILE
else
if (DEBUG) {
    cerr << "In 'create_tables_gwirdcli': 'submit_mysql_queries' succeeded, "
    << "returning 0." << endl << "Created tables in " << gwirdcli_database_name <<
    " database successfully." << endl;
int i = 0;
int j = 0;
for (vector<long int *>::iterator iter = affected_rows_vector.begin();
     iter != affected_rows_vector.end(); ++iter) {
    if (*iter != 0) {
        cerr << "*affected_rows_vector" << setw(4) << left << temp_strm.str() << " == "
        << *affected_rows_vector[i];
        if (comment_vector.size() > j) cerr << " " << comment_vector[j++];
        cerr << endl;
    }
    ++i;
} /* for */
cerr << endl;
} /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

23.

```

⟨ create_tables_gwirdcli definition 20 ⟩ +≡
delete_and_clear(result_array, result_array_size, row_ctr_vector, field_ctr_vector, affected_rows_vector,
                  query_vector);
sql_strm.str("");
#ifndef DEBUG_COMPILE
if (DEBUG) {
    cerr << "Exiting 'create_tables_gwirdcli' successfully with return value 0." << endl;
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
return 0; /* End of create_tables_gwirdcli definition */

```

24. This is what's compiled. [LDF 2013.09.03.]

```
using namespace std;
⟨ Include files 3 ⟩
using namespace gwrdifpk;
⟨ create_tables_gwirdcli definition 20 ⟩
```

25. This is what's written to the header file `stpcdcli.h`. [LDF 2013.09.03.]

```
⟨ stpcdcli.h 25 ⟩ ≡
#ifndef STPCDCLI_H
#define STPCDCLI_H 1
⟨ create_tables_gwirdcli declaration 19 ⟩
#endif
```

26. Include files. [LDF 2013.09.04.]

```
<Include files 3> +≡
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>      /* POSIX threads */
#include <gcrypt.h>        /* for gcry_control */
#include <gnutls/gnutls.h>
#include <gnutls/x509.h>
#include <mysql.h>
#include <expat.h>
#include <iomanip>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <getopt.h>
#include <map>
#include <vector>
#include <deque>
#include <set>
#include <stack>
#include <utility>
#if HAVE_CONFIG_H
#include <config.h>
#endif
#include "rspercds.h++"
#include "glblcnst.h++"
#include "glblvrbl.h++"
#include "excptntp.h++"
#include "utilfncts.h++"
#include "grouptp.h++"
#include "hndlvltp.h++"
#include "irdsavtp.h++"
#include "helper.h++"
#include "tanfncts.h++"
#include "pidfncts.h++"
#include "parser.h++"
#include "scanner.h++"
#include "rspnstp.h++"
#include "irdsobtp.h++"
#include "hdltype.h++"
#include "dcmtddtp.h++"
#include "x509cert.h++"
#include "dstngnmt.h++"
#include "usrinftp.h++"
#include "scprpmtp.h++"
#include "stpclopt.h++"
```

27. Create handle database tables (default: `handlesystem` or `handlesystem_standalone`). [LDF 2013.09.04.] ■
⟨ *create_tables_handles* declaration 27 ⟩ ≡
int *create_tables_handles*(Scan_Parse_Parameter_Type &*param*);

This code is used in section 33.

28.

```

⟨ create_tables_handles definition 28 ⟩ ≡
int create_tables_handles(Scan_Parse_Parameter_Type &param){ bool DEBUG = false;
    /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    stringstream sql_strm;
    stringstream temp_strm;
    MYSQL_RES * result = 0;
    MYSQL_ROW curr_row;
    size_t result_array_size = 16;
    MYSQL_RES * result_array[16] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
    unsigned int row_ctr = 0;
    unsigned int field_ctr = 0;
    long int affected_rows = 0;
    vector<string> query_vector;
    vector<unsigned int *> row_ctr_vector;
    vector<unsigned int *> field_ctr_vector;
    vector<long int *> affected_rows_vector;
    string comma_str;
#define DEBUG_COMPILE
    if (DEBUG) {
        cerr << "Entering 'create_tables_handles'." << endl;
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    sql_strm << "use " << handles_database_name;
    query_vector.push_back(sql_strm.str());
    row_ctr_vector.push_back(0);
    field_ctr_vector.push_back(0);
    affected_rows_vector.push_back(0);
    sql_strm.str("");
    nas_table_create_str = "create_table_nas_(na varchar(255) not null, PRIMARY KEY(na))";
    query_vector.push_back(nas_table_create_str);
    row_ctr_vector.push_back(0);
    field_ctr_vector.push_back(0);
    affected_rows_vector.push_back(0);
    nas_table_insert_str = "";
    comma_str = "";
    if (standalone_handle ∨ prefix_vector.size() > 0) {
        nas_table_insert_str = "insert into nas_(na) values ";
    }
    if (standalone_handle) {
        nas_table_insert_str += "('0.NA/0.NA')";
        comma_str = ",\n";
    }
    if (prefix_vector.size() > 0) { /* This should always be true. [LDF 2013.09.04.] */
        for (vector<string>::iterator iter = prefix_vector.begin(); iter ≠ prefix_vector.end(); ++iter) {
            sql_strm << comma_str << "('0.NA/" << *iter << ")";
            comma_str = ",";
        }
    }
}

```

```

comma_str = "";
nas_table_insert_str += sql_strm.str();
} /* if */
if (nas_table_insert_str.size() > 0) {
    query_vector.push_back(nas_table_insert_str);
    row_ctr_vector.push_back(0);
    field_ctr_vector.push_back(0);
    affected_rows_vector.push_back(new long int(0L));
}
else {
    cerr << "'standalone_handle'==false and no prefixes specified." << endl <<
        "Not inserting row(s) into " << handles_database_name << ".nas" << endl;
}
sql_strm.str("");
sql_strm << "create_table_handles";
/* The following fields are specified by the Handle system. LDF 2013.08.22. */
sql_strm << "handle_varchar(255)not null," << "idx_int4not null,\n"
    "type_blob," << "data_blob," << "ttl_type_int2," << "ttl_int4," << "timestamp_int4," << "refs_blob," << "ad
    /* The following fields are additional, i.e., not specified or used by the Handle system! LDF
       2013.08.22. */
    sql_strm << "created_timestamp_default_0," << "last_modified_timestamp_default_0," <<
        "created_by_user_id_int," << "irods_object_id bigint unsigned not null\"
        1_default_0," << "handle_id bigint unsigned not null default_0," <<
        "handle_value_id bigint unsigned not null default_0," <<
        "marked_for_deletion_boolean not null default false," <<
        "delete_from_database_timestamp_timestamp_default_0," <<
        "PRIMARY KEY(handle, idx))";
handles_table_create_str = sql_strm.str();
query_vector.push_back(sql_strm.str());
row_ctr_vector.push_back(0);
field_ctr_vector.push_back(0);
affected_rows_vector.push_back(0);
sql_strm.str("");
sql_strm << "alter_table_handles_add_foreign_key(created_by_user_id)" <<
    "references_gwidsif.Users(user_id)";
handles_table_alter_str[0] = sql_strm.str();
query_vector.push_back(sql_strm.str());
row_ctr_vector.push_back(0);
field_ctr_vector.push_back(0);
affected_rows_vector.push_back(0);
sql_strm.str("");
sql_strm << "alter_table_handles_add_constraint_unique_key_(handle_id, h\
    andle_value_id)";
handles_table_alter_str[1] = sql_strm.str();
query_vector.push_back(sql_strm.str());
row_ctr_vector.push_back(0);
field_ctr_vector.push_back(0);
affected_rows_vector.push_back(0);
sql_strm.str("");
sql_strm << "create_table_admin_data(" << "handle_varchar(255)not null," <<
    "data_blob)";
query_vector.push_back(sql_strm.str());

```

```

row_ctr_vector.push_back(0);
field_ctr_vector.push_back(0);
affected_rows_vector.push_back(0);
sql_strm.str("");
sql_strm << "create_table_pid_counters(" << "prefix_varchar(16)_primary_key_not_null\
    , " << "pid_counter bigint unsigned not null)";
query_vector.push_back(sql_strm.str());
row_ctr_vector.push_back(0);
field_ctr_vector.push_back(0);
affected_rows_vector.push_back(0);
sql_strm.str("");
sql_strm << "insert_into_pid_counters(prefix, pid_counter) " <<
    "values('00001', 0), ('12345', 0)";
query_vector.push_back(sql_strm.str());
row_ctr_vector.push_back(0);
field_ctr_vector.push_back(0);
affected_rows_vector.push_back(new long int(0L));
sql_strm.str("");
#endif DEBUG_COMPILE
if (DEBUG) {
    cerr << `query_vector.size() == " << query_vector.size() << endl;
    cerr << `query_vector: " << endl;
    for (vector<string>::iterator iter = query_vector.begin(); iter != query_vector.end(); ++iter) {
        cerr << *iter << endl;
    }
    cerr << endl;
}
/* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

See also sections 29, 30, and 31.

This code is used in section 32.

29.

```

⟨ create_tables_handles definition 28 ⟩ +≡
  if (query_vector.size() > result_array_size) {
    cerr << "[setupdbs] ERROR! In 'main': " << endl << "query_vector.size() == " <<
      query_vector.size() << "> result_array_size == " << result_array_size << endl <<
      "Can't call 'Scan_Parse_Parameter_Type::submit_mysql_queries'." << endl <<
      "Exiting function unsuccessfully with exit status 1." << endl;
    exit(1);
  }
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      cerr << "query_vector.size() == " << query_vector.size() << endl <<
        "result_array_size == " << result_array_size << endl;
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
  status = param.submit_mysql_queries(query_vector, result_array, row_ctr_vector, field_ctr_vector,
    affected_rows_vector, false);
  if (status != 0) {
    cerr << "ERROR! In 'main': 'submit_mysql_queries' failed, returning " << status << "."
    endl << "Exiting 'setupdbs' unsuccessfully with exit status 1." << endl;
    delete_and_clear(result_array, result_array_size, row_ctr_vector, field_ctr_vector, affected_rows_vector,
      query_vector);
    exit(1);
  } /* if (status != 0) */

```

30.

```

⟨ create_tables_handles definition 28 ⟩ +≡
#ifndef DEBUG_COMPILE
  else
    if (DEBUG) {
      cerr << "In 'main': 'submit_mysql_queries' succeeded, returning 0." << endl <<
        "Created tables in " << handles_database_name << ", database successfully." << endl;
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

31.

```

⟨ create_tables_handles definition 28 ⟩ +≡
  delete_and_clear(result_array, result_array_size, row_ctr_vector, field_ctr_vector, affected_rows_vector,
    query_vector);
  sql_strm.str("");
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    cerr << "Exiting 'create_tables_handles' successfully with return value 0." << endl;
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
  return 0; } /* End of create_tables_handles definition */

```

32. This is what's compiled. [LDF 2013.09.03.]

```
using namespace std;
⟨ Include files 3 ⟩
using namespace gwrdifpk;
⟨ create_tables_handles definition 28 ⟩
```

33. This is what's written to the header file `stpcdhd1.h`. [LDF 2013.09.03.]

```
⟨ stpcdhd1.h 33 ⟩ ≡
#ifndef STPCDHDL_H
#define STPCDHDL_H 1
⟨ create_tables_handles declaration 27 ⟩
#endif
```

34. Include files. [LDF 2013.09.04.]

```
<Include files 3> +≡
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h> /* POSIX threads */
#include <gcrypt.h> /* for gcry_control */
#include <gnutls/gnutls.h>
#include <gnutls/x509.h>
#include <mysql.h>
#include <expat.h>
#include <iomanip>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <getopt.h>
#include <map>
#include <vector>
#include <deque>
#include <set>
#include <stack>
#include <utility>
#if HAVE_CONFIG_H
#include <config.h>
#endif
#include "rspercds.h++"
#include "glblcnst.h++"
#include "glblvrbl.h++"
#include "excptntp.h++"
#include "utilfncts.h++"
#include "groupntp.h++"
#include "hndlvltp.h++"
#include "irdsavtp.h++"
#include "helper.h++"
#include "tanfncts.h++"
#include "pidfncts.h++"
#include "parser.h++"
#include "scanner.h++"
#include "rspnstp.h++"
#include "irdsobtp.h++"
#include "hdltype.h++"
#include "dcmtddtp.h++"
#include "x509cert.h++"
#include "dstngnmt.h++"
#include "usrinftp.h++"
#include "scprpmtp.h++"
#include "stpclopt.h++"
```

35. Create server-side database tables (gwirdsif). [LDF 2013.09.04.]

$\langle \text{create_tables_gwirdsif} \text{ declaration } 35 \rangle \equiv$
`int create_tables_gwirdsif(Scan_Parse_Parameter_Type ¶m);`

This code is used in section 66.

36.

```

        cerr << iter->first << ":" << iter->second << endl;
    }      /* for */
    cerr << endl;
}      /* if (DEBUG) */
#endif  /* DEBUG_COMPILE */
comma_str = ",\n";
int i = 1;
for (vector<pair<string, string>>::iterator iter = institute_vector.begin();
     iter != institute_vector.end(); ++iter) {
    sql strm << comma_str << "(" << i++ << ",_1," << iter->first << ",_1" << ","
        iter->second << ",_1)";
}      /* for */
comma_str = "";
}      /* if (institute_vector.size() > 0) */
query_vector.push_back(sql strm.str());
row_ctr_vector.push_back(0);
field_ctr_vector.push_back(0);
affected_rows_vector.push_back(new long int(0L));
comment_vector.push_back("insert_Institutes");
sql strm.str("");

```

See also sections 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, and 63.

This code is used in section 65.

37. Prefixes. [LDF 2013.09.04.]

Log

[LDF 2013.09.04.] Added this section.

```

< create_tables_gwirdsif definition 36 > +≡
sql strm << "create_table_Prefixes" << endl << "(" << endl <<
"prefix_id_int_primary_key," << endl << "prefix_varchar(16)_unique_not_null," << endl <<
"enabled_boolean_not_null_default_1" << endl << ")";
query_vector.push_back(sql strm.str());
row_ctr_vector.push_back(0);
field_ctr_vector.push_back(0);
affected_rows_vector.push_back(0);
sql strm.str("");

```

38.

```

⟨ create_tables_gwirdsif definition 36 ⟩ +≡
sql_strm << "insert_into_Prefixes(prefix_id, prefix, enabled)" << endl << "values" << endl <
    "(0, 'NULL_PREFIX', 0)";
comma_str = ",\n";
int i = 1;
for (vector<string>::iterator iter = prefix_vector.begin(); iter != prefix_vector.end(); ++iter) {
    sql_strm << comma_str << "(" << i++ << ", " << *iter << ', 1)';
}
query_vector.push_back(sql_strm.str());
row_ctr_vector.push_back(0);
field_ctr_vector.push_back(0);
affected_rows_vector.push_back(new long int(0L));
comment_vector.push_back("insert_Prefixes");
sql_strm.str("");

```

39.

```

⟨ create_tables_gwirdsif definition 36 ⟩ +≡
sql_strm << "create_table_Users" << endl << "(" << endl << "user_id int primary key," <<
endl << "username varchar(128) unique not null," << endl <
"irods_password_encrypted varchar(2048), /* 2^11 */" << endl <
"irods_password_encrypted_timestamp timestamp default 0," << endl <
"Distinguished_Name varchar(256)," << endl << "irods_homedir varchar(128)," << endl <
"irods_zone varchar(128)," << endl << "irods_default_resource varchar(128)," <<
endl << "handle_username varchar(128) default ''," << endl <
"handle_password_encrypted varchar(32) default ''," << endl <
"default_institute_id int references Institutes(institute_id)," <<
endl << "default_prefix_id int references Prefixes(prefix_id)," << endl <
"public_key_id char(8) not null default ''" << ")" << endl << endl;
Users_table_create_str = sql_strm.str();
query_vector.push_back(sql_strm.str());
row_ctr_vector.push_back(0);
field_ctr_vector.push_back(0);
affected_rows_vector.push_back(0);
sql_strm.str("");

```

40.

```

⟨ create_tables_gwirdsif definition 36 ⟩ +≡
sql_strm << "insert_into_Users" << endl << "(user_id, " << "username, "
    "irods_password_encrypted, " << "irods_password_encrypted_timestamp, "
    "Distinguished_Name," << endl << "irods_homedir, " << "irods_zone, "
    "irods_default_resource, " << "handle_username, " << endl <<
    "handle_password_encrypted, " << "default_institute_id, "
    "default_prefix_id, " << "public_key_id)" << endl << "values" << endl <<
    "(" << "0, " 'NULL_USER', "NULL_IRODS_PASSWORD_ENCRYPTED', "
    "NULL_DISTINGUISHED_NAME', "NULL_IRODS_HOMEDIR', "NULL_IRODS_ZONE', "
    "NULL_DEFAULT_RESOURCE', "NULL_HANDLE_USER', "
    "'NULL_HANDLE_PASSWORD_ENCRYPTED', " << endl << "0, 0, 00000000')";

i = 1;
comma_str = ",\n";
for (vector<string>::iterator iter = admin_vector.begin(); iter != admin_vector.end(); ++iter) {
    /* public_key_id will have to be added later, by hand. [LDF 2013.09.20.] */
    sql_strm << comma_str << "(" << i++ << ", " /* user_id */
    << ", " << *iter << ", " /* username */
    << ", " /* irods_password_encrypted */
    << "0, " /* irods_password_encrypted_timestamp */
    << ", " /* Distinguished_Name */
    << "/tempZone/home/" /* irods_homedir */
    << *iter << ", " /* tempZone */
    << "demoResc", " /* irods_default_resource */
    << ", " << *iter << ", " /* handle_username */
    << ", " /* handle_password_encrypted */
    << "0, " /* default_institute_id */
    << "0, " /* default_prefix_id */
    << "); " /* public_key_id */
}
/* for */

```

41. If there are no admins, set $i = 2$ so that an admin with $user_id \equiv 1$ can be added by hand later. There's no particular need for this, but I think it's a useful convention. [LDF 2013.09.06.]

Log

[LDF 2013.09.06.] Added this section.

```

⟨ create_tables_gwirdsif definition 36 ⟩ +≡
if (i ≡ 1) ++i;

```

42.

```

⟨ create_tables_gwirdsif definition 36 ⟩ +≡
for (vector<string>::iterator iter = user_vector.begin(); iter ≠ user_vector.end(); ++iter) {
    /* public_key_id will have to be added later, by hand. [LDF 2013.09.20.] */
    sql_strm << comma_str << "(" << i++ << ", " /* user_id */ 
    << "," << *iter << ", " /* username */ 
    << "," << " " /* irods_password_encrypted */ 
    << "0, " /* irods_password_encrypted_timestamp */ 
    << "," /* Distinguished_Name */ 
    << "/tempZone/home/" /* irods_homedir */ 
    << *iter << ", " << "tempZone", " /* irods_zone */ 
    << "'demoResc', " /* irods_default_resource */ 
    << "," << *iter << ", " /* handle_username */ 
    << " ", " /* handle_password_encrypted */ 
    << "0, " /* default_institute_id */ 
    << "0, " /* default_prefix_id */ 
    << "); " /* public_key_id */ 
} /* for */
Users_table_insert_str = sql_strm.str();
query_vector.push_back(sql_strm.str());
row_ctr_vector.push_back(0);
field_ctr_vector.push_back(0);
affected_rows_vector.push_back(new long int(0L));
comment_vector.push_back("insert_Users");
sql_strm.str("");

```

43. Users_Prefixes. [LDF 2013.09.04.]

Log

[LDF 2013.09.04.] Added this section.

```

⟨ create_tables_gwirdsif definition 36 ⟩ +≡
sql_strm << "create_table_Users_Prefixes" << endl << "(" <<
endl << "_user_id_int_references_Users(user_id)," << endl <<
"_prefix_id_int_references_Prefixes(prefix_id))";
query_vector.push_back(sql_strm.str());
row_ctr_vector.push_back(0);
field_ctr_vector.push_back(0);
affected_rows_vector.push_back(0);
sql_strm.str("");
sql_strm << "insert_into_Users_Prefixes_(user_id,_prefix_id)" << endl << "values" << endl <<
"(0,_0)";

```

44. If at least one user and at least one prefix has been specified, assign the first prefix to the first user.
[LDF 2013.09.04.]

```

⟨ create_tables_gwirdsif definition 36 ⟩ +≡
if (admin_vector.size() > 0 ∧ prefix_vector.size() > 0) {
    sql_strm << comma_str << "(1,_1)";
}

```

45.

```
< create_tables_gwirdsif definition 36 > +≡
query_vector.push_back(sql_strm.str());
row_ctr_vector.push_back(0);
field_ctr_vector.push_back(0);
affected_rows_vector.push_back(new long int(0L));
comment_vector.push_back("insert_Users_Prefixes");
sql_strm.str("");
```

46. Certificates. [LDF 2013.09.05.]**Log**

[LDF 2013.09.05.] Added this section.

```
< create_tables_gwirdsif definition 36 > +≡
sql_strm << "create_table_Certificates" << endl << "(" <<
"certificate_id_int_primary_key," << "user_id_int_references_Users(user_id)," <<
endl << "issuer_cert_id_int_references_Certificates(certificate_id)," <<
"is_ca_boolean_not_null," << "is_proxy_boolean_not_null," << endl <<
"serialNumber bigint_unsigned," << endl;
/* Fields from here on use names from the X.509 specifications */ /* LDF 2009.12.23. */
sql_strm << "organization_varchar(64)," << "organizationalUnitName_varchar(64)," <<
"commonName_varchar(64)," << "countryName_char(2)," << "localityName_varchar(64)," <<
"stateOrProvinceName_varchar(64)," << "Validity_notBefore_datetime," <<
"Validity_notAfter_datetime" << endl << ")";
query_vector.push_back(sql_strm.str());
row_ctr_vector.push_back(0);
field_ctr_vector.push_back(0);
affected_rows_vector.push_back(0);
sql_strm.str("");
```

47.

```
< create_tables_gwirdsif definition 36 > +≡
sql_strm << "insert_into_Certificates" << endl << "(" << endl << "certificate_id," <<
"user_id," << "issuer_cert_id," << "is_ca," << "is_proxy," << endl <<
"serialNumber," << "organization," << "organizationalUnitName," << "commonName," <<
"countryName," << endl << "localityName," << "stateOrProvinceName," <<
"Validity_notBefore," << "Validity_notAfter" << endl << ")" << "values" << "(" << "0," <<
"0," << "NULL," << "false," << endl << "false," << "0," << "NULL," << "NULL," <<
"NULL," << endl << "NULL," << "NULL," << "NULL," << "NULL" << endl << ")";
query_vector.push_back(sql_strm.str());
row_ctr_vector.push_back(0);
field_ctr_vector.push_back(0);
affected_rows_vector.push_back(new long int(0L));
comment_vector.push_back("insert_Certificates");
sql_strm.str("");
```

48. Privileges. [LDF 2013.09.05.]

Log

[LDF 2013.09.05.] Added this section.

```

< create_tables_gwirdsif definition 36 > +=

sql_strm << "create_table_Privileges" << endl << "(" << endl <<
    "    user_id int primary key unique not null references Users(user_id)," <<
    endl << "    superuser boolean not null default 0," <<
    endl << "    delegate boolean not null default 0," << endl <<
    "    add_groups boolean not null default 0," << endl <<
    "    delete_groups boolean not null default 0," << endl <<
    "    delete_handles boolean not null default 0," << endl <<
    "    delete_handle_values boolean not null default 0," << endl <<
    "    delete_hs_admin_handle_values boolean not null default 0," << endl <<
    "    delete_last_hs_admin_handle_value boolean not null default 0," <<
    endl << "    undelete_handle_values boolean not null default 0," <<
    endl << "    show_user_info boolean not null default 0," <<
    endl << "    show_groups boolean not null default 0," << endl <<
    "    show_certificates boolean not null default 0," << endl <<
    "    show_distinguished_names boolean not null default 0," << endl <<
    "    show_privileges boolean not null default 0" << endl << ")";
query_vector.push_back(sql_strm.str());
row_ctr_vector.push_back(0);
field_ctr_vector.push_back(0);
affected_rows_vector.push_back(0);
sql_strm.str("");

```

49.

```

< create_tables_gwirdsif definition 36 > +≡
  if (admin_vector.size() ≡ 0 ∧ user_vector.size() ≡ 0) {
    cerr ≪ "NOTICE: In 'create_tables_gwirdsif':" ≪ endl ≪
      "No admins or users specified. Not inserting into 'Privileges' table." ≪ endl;
  }
  else { sql_strm ≪ "insert into Privileges (user_id, superuser, delegate, show_user_info," ≪
    endl ≪ "show_groups, " ≪ "add_groups, " ≪ "delete_groups," ≪ endl ≪
    "delete_handles, " ≪ "delete_handle_values, " ≪ "delete_hs_admin_handle_values, " ≪
    "delete_last_hs_admin_handle_value, " ≪ "undelete_handle_values," ≪ endl ≪
    "show_certificates, " ≪ "show_distinguished_names, " ≪ "show_privileges)" ≪ endl ≪
    "values" ≪ endl;
  comma_str = "";
  int i = 1;
  for (vector<string>::iterator iter = admin_vector.begin(); iter ≠ admin_vector.end(); ++iter) {
    sql_strm ≪ comma_str ≪ "(" ≪ i ≪ ", " ≪ "true, true, true, true, true, "
      "true, true, true, true, true, true, true, true)";
    comma_str = ", \n";
    ++i;
  } /* for */

```

50. If there are no admins, the first user, if any, has *user_id* \equiv 2. See explanation for this above (insert into `Users` table). [LDF 2013.09.06.]

Log

[LDF 2013.09.06.] Added this section.

```
< create_tables_gwirdsif definition 36 > +≡
  if (i ≡ 1) i = 2;
```

51.

```
< create_tables_gwirdsif definition 36 > +≡
  for (vector<string>::iterator iter = user_vector.begin(); iter ≠ user_vector.end(); ++iter) {
    sql_strm ≪ comma_str ≪ "(" ≪ i ≪ ", " ≪ "false, false, false, false, false, false, false, "
      "false, false, false, false, false, false, false, false)" ;
    comma_str = ",\n";
    ++i;
  } /* for */
  query_vector.push_back(sql_strm.str());
  row_ctr_vector.push_back(0);
  field_ctr_vector.push_back(0);
  affected_rows_vector.push_back(new long int(0L));
  comment_vector.push_back("insert_Privileges");
  sql_strm.str(""); } /* else */
```

52. `User_Info`. [LDF 2013.09.05.]

Log

[LDF 2013.09.05.] Added this section.

```
< create_tables_gwirdsif definition 36 > +≡
  sql_strm ≪ "create_view_User_Info_as_select" ≪ endl ≪ "U.user_id, U.username, "
    "C.certificate_id, C.serialNumber, C.commonName, C.organization," ≪ endl ≪
    "C.organizationalUnitName, " ≪ "P.superuser, P.delegate, P.show_user_in\
fo, P.show_groups," ≪ endl ≪ "P.show_certificates, P.show_distinguished_names, "
    "P.show_privileges" ≪ endl ≪ "from_Users as U, Certificates as C, Privileges as P" ≪
    endl ≪ "where_U.user_id=C.user_id and_U.user_id=P.user_id" ≪ endl ≪
    "order_by_U.user_id";
  query_vector.push_back(sql_strm.str());
  row_ctr_vector.push_back(0);
  field_ctr_vector.push_back(0);
  affected_rows_vector.push_back(0);
  sql_strm.str("");
```

53. TANs. [LDF 2013.09.05.]

Log

[LDF 2013.09.05.] Added this section.

```
< create_tables_gwirdsif definition 36 > +≡
sql_strm << "create_table_TANs" << endl << "(" << endl <<
    "user_id int default 0 references Users(user_id)," << endl <<
    "TAN varchar(64) primary key," << endl << "expiration_timestamp default 0" <<
    endl << ")";
query_vector.push_back(sql_strm.str());
row_ctr_vector.push_back(0);
field_ctr_vector.push_back(0);
affected_rows_vector.push_back(0);
sql_strm.str("");
```

54.

```
< create_tables_gwirdsif definition 36 > +≡
sql_strm << "insert into TANs(TAN)" << endl << "values" << endl <<
    "('xEzo_9Xd9_ujBWT_F0zJ_K0Fs_Ssmt')," << endl << "('gQpg_vz5B_xBPP_wB2p_WQNJ_gjgv')," <<
    endl << "('Etku_40cQ_DA1_wgGxN_Wrcm_tN2Z')," << endl <<
    "('8TRf_0dsX_UdP5_ZFh1_tRuY_aTMi')," << endl << "('6gGs_ZWm8_yKgg_2o3U_NvZZ_lF1e')," <<
    endl << "('bqq6_Lczp_kv2T_upSqu_hNbb_JBCg')," << endl <<
    "('zJtG_lwJb_Reww_11fg_tgtD_SeIk')," << endl << "('QioT_R4za_6ZCp_MGEZ_v7z0_TZ8y')";
query_vector.push_back(sql_strm.str());
row_ctr_vector.push_back(0);
field_ctr_vector.push_back(0);
affected_rows_vector.push_back(new long int(0L));
comment_vector.push_back("insert_TANs");
sql_strm.str("");
```

55. Session_Data. [LDF 2013.09.05.]

Log

[LDF 2013.09.05.] Added this section.

```
< create_tables_gwirdsif definition 36 > +≡
sql_strm << "create_table_Session_Data" << endl << "(" << endl << "session_id varchar(\n"
    256)_not null," << "user_id int not null references Users(user_id)," <<
    endl << "effective_user_id int references Users(user_id)," <<
    "user_name varchar(128)_references_Users(user_name)," << endl <<
    "effective_user_name varchar(128)_references_Users(user_name)," <<
    "timestamp timestamp default 0" << endl << ")";
query_vector.push_back(sql_strm.str());
row_ctr_vector.push_back(0);
field_ctr_vector.push_back(0);
affected_rows_vector.push_back(0);
sql_strm.str("");
```

56. Groups. [LDF 2013.09.05.]

Log

[LDF 2013.09.05.] Added this section.

```

⟨ create_tables_gwirdsif definition 36 ⟩ +≡
sql strm ≪ "create_table_Groups" ≪ endl ≪ "(" ≪ endl ≪
    "group_id int primary key unique not null," ≪ endl ≪
    "creator_id int not null references Users(user_id)," ≪ endl ≪
    "name varchar(64) unique not null," ≪ endl ≪ "created datetime" ≪ endl ≪ ")";
query_vector.push_back(sql strm.str());
row_ctr_vector.push_back(0);
field_ctr_vector.push_back(0);
affected_rows_vector.push_back(0);
sql strm.str("");
sql strm ≪ "insert into Groups(group_id, creator_id, name, created)" ≪ endl ≪ "values" ≪
    endl ≪ "(0, 0, 'NULL_GROUP', 0)";
comma_str = ",\n";
i = 1;
for (vector<string>::iterator iter = group_vector.begin(); iter ≠ group_vector.end(); ++iter) {
    sql strm ≪ comma_str ≪ "(" ≪ i++ ≪ ", 1, " ≪ *iter ≪ ", now())";
}
query_vector.push_back(sql strm.str());
row_ctr_vector.push_back(0);
field_ctr_vector.push_back(0);
affected_rows_vector.push_back(new long int(0L));
comment_vector.push_back("insert Groups");
sql strm.str("");

```

57.

```

⟨ create_tables_gwirdsif definition 36 ⟩ +≡
sql strm ≪ "create_table_Groups_Users" ≪ endl ≪ "(" ≪ endl ≪
    "group_id int not null references Groups(group_id)," ≪
    endl ≪ "user_id int not null references Users(user_id)," ≪
    endl ≪ "add_user_priv boolean not null default 0," ≪ endl ≪
    "delete_user_priv boolean not null default 0," ≪ endl ≪
    "delete_group_priv boolean not null default 0" ≪ endl ≪ ")";
query_vector.push_back(sql strm.str());
row_ctr_vector.push_back(0);
field_ctr_vector.push_back(0);
affected_rows_vector.push_back(0);
sql strm.str("");

```

58.

```

⟨ create_tables_gwirdsif definition 36 ⟩ +≡
sql_strm << "insert into Groups_Users(group_id, user_id, "
    "add_user_priv, delete_user_priv, delete_group_priv)" << endl << "values" << endl <<
    "(0, 0, 0, 0)";
query_vector.push_back(sql_strm.str());
row_ctr_vector.push_back(0);
field_ctr_vector.push_back(0);
affected_rows_vector.push_back(new long int(0L));
comment_vector.push_back("insert Groups_Users");
sql_strm.str("");
sql_strm << "create view Group_Info as select" << endl << "GU.group_id, G.n\"
    ame as group_name," << endl << "GU.user_id, U.username as 'username', " <<
    endl << "GU.add_user_priv, GU.delete_user_priv, GU.delete_group_priv, " <<
    endl << "G.creator_id, UU.username as creator_name, G.created" << endl <<
    "from Groups as G, Users as U, Groups_Users as GU, Users as UU" << endl <<
    "where G.group_id = GU.group_id and U.user_id = GU.user_id" << endl <<
    "and G.creator_id = UU.user_id" << endl << "order by G.group_id, GU.user_id";
query_vector.push_back(sql_strm.str());
row_ctr_vector.push_back(0);
field_ctr_vector.push_back(0);
affected_rows_vector.push_back(0);
sql_strm.str("");

```

59.

```

⟨ create_tables_gwirdsif definition 36 ⟩ +≡
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        cerr << "query_vector.size() == " << query_vector.size() << endl;
        for (vector<string>::iterator iter = query_vector.begin(); iter != query_vector.end(); ++iter) {
            cerr << *iter << endl;
        } /* for */
        cerr << endl;
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

60.

Log

[LDF 2013.09.04.] Added this section.

```

⟨ create_tables_gwirdsif definition 36 ⟩ +≡
for (vector<string>::iterator iter = query_vector.begin(); iter != query_vector.end(); ++iter) {
    out_strm << *iter << ";" << endl << endl;
}

```

61.

```

⟨ create_tables_gwirdsif definition 36 ⟩ +≡
status = param.submit_mysql_queries(query_vector, result_array, row_ctr_vector, field_ctr_vector,
    affected_rows_vector, false);
if (status ≠ 0) {
    cerr ≪ "ERROR! In 'create_tables_gwirdsif': 'submit_mysql_queries' failed, "
    ≪ "returning " ≪ status ≪ "." ≪ endl ≪ "Exiting 'create_tables_gwirdsif' unsuccessfully"
    ≪ endl;
    delete_and_clear(result_array, result_array_size, row_ctr_vector, field_ctr_vector, affected_rows_vector,
        query_vector);
    out_strm ≪ "/* Failed */" ≪ endl ≪ endl ≪ "/* **** */" ≪ endl ≪ endl ≪ endl;
    ***" ≪ "*****" ≪ endl ≪ endl;
    return 1;
} /* if (status ≠ 0) */

```

62.

```

⟨ create_tables_gwirdsif definition 36 ⟩ +≡
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        cerr ≪ "In 'create_tables_gwirdsif': 'submit_mysql_queries' succeeded,"
            ≪ endl ≪ "Created tables in " ≪ gwirdsif_database_name ≪
            "' database successfully." ≪ endl;
        int i = 0;
        int j = 0;
        for (vector<long int *>::iterator iter = affected_rows_vector.begin();
            iter ≠ affected_rows_vector.end(); ++iter) {
            if (*iter ≠ 0) {
                temp_strm.str("");
                temp_strm ≪ "[" ≪ i ≪ "]";
                cerr ≪ "*affected_rows_vector" ≪ setw(4) ≪ left ≪ temp_strm.str() ≪ " == "
                    ≪ *affected_rows_vector[i];
                if (comment_vector.size() > j) cerr ≪ " " ≪ comment_vector[j++];
                cerr ≪ endl;
            }
            ++i;
        } /* for */
        temp_strm.str("");
        cerr ≪ endl;
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */

```

63.

```

⟨ create_tables_gwirdsif definition 36 ⟩ +≡
out_strm << "/*_Succeeded/*" << endl << endl <<
"/*_*****" << endl << endl << /*_!_TODO:_Assign_prefixes_to_users_by_inserting_rows_*/
"into_the_‘Users_Prefixes’_table.uu" << endl << "uuuuuuuuuuuuAssign_\
groups_to_users_by_inserting_rows_into_the_u" << "‘Groups_Users’_table." <<
endl << "uuuuuuuuuuuAssign_privileges_by_inserting_rows_into_the_u" <<
"‘Privileges’_table.uu" << /*_*/ << endl << endl;
delete_and_clear(result_array, result_array_size, row_ctr_vector, field_ctr_vector, affected_rows_vector,
query_vector);
sql_strm.str("");
#ifndef DEBUG_COMPILE
if (DEBUG) {
    cerr << "Exiting_‘create_tables_gwirdsif’_successfully_with_return_value_0." << endl;
} /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
return 0; } /* End of create_tables_gwirdsif definition */

```

64. Garbage. This section needs to exist, because it's included below. It can be used for temporarily (or permanently) commenting-out code. [LDF 2013.09.05.]

```

⟨ Garbage 64 ⟩ ≡
#ifndef 0 /* Empty */
#endif

```

This code is used in section 65.

65. This is what's compiled. [LDF 2013.09.03.]

```

using namespace std;
⟨ Include files 3 ⟩
using namespace gwrdifpk;
⟨ create_tables_gwirdsif definition 36 ⟩
#ifndef 0
⟨ Garbage 64 ⟩
#endif

```

66. This is what's written to the header file `stpcdsif.h`. [LDF 2013.09.03.]

```

⟨ stpcdsif.h 66 ⟩ ≡
#ifndef STPCDSIF_H
#define STPCDSIF_H 1
⟨ create_tables_gwirdsif declaration 35 ⟩
#endif

```

67. Include files. [LDF 2013.09.02.]

```
<Include files 3> +≡
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>      /* POSIX threads */
#include <gcrypt.h>        /* for gcry_control */
#include <gnutls/gnutls.h>
#include <gnutls/x509.h>
#include <mysql.h>
#include <expat.h>
#include <iomanip>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <getopt.h>
#include <map>
#include <vector>
#include <deque>
#include <set>
#include <stack>
#include <utility>
#if HAVE_CONFIG_H
#include <config.h>
#endif
#include "rspercds.h++"
#include "glblcnst.h++"
#include "glblvrbl.h++"
#include "excptntp.h++"
#include "utilfncs.h++"
#include "grouptp.h++"
#include "hndlvltp.h++"
#include "irdsavtp.h++"
#include "helper.h++"
#include "tanfncs.h++"
#include "pidfncs.h++"
#include "parser.h++"
#include "scanner.h++"
#include "rspnstp.h++"
#include "irdsobtp.h++"
#include "hdltype.h++"
#include "dcmtddtp.h++"
#include "x509cert.h++"
#include "dstngnmt.h++"
#include "usrinftp.h++"
#include "scprpmtp.h++"
```

68. Global variables. [LDF 2013.09.02.]

Log

[LDF 2013.09.04.] Added **bool** *standalone_handle* = *true*, *set* < **pair**<**string**, **string**> > *institute_set*, *set* < **pair**<**string**, **int**> > *user_set* and **ofstream** *out_strm*.

[LDF 2013.09.05.] Changed the *sets* *user_set*, *prefix_set* and *institute_set* to **vectors** because I don't want the elements to be sorted.

[LDF 2013.09.05.] Added **ifstream** *license_strm* and **vector**<**string**> *group_vector*.

[LDF 2013.09.06.] Changed type of *user_vector* from **vector**<**pair**<**string**, **int**>> to **vector**<**string**>. Added **vector**<**string**> *admin_vector*.

[LDF 2013.09.06.] Added **int** *irods_server_port* = 1247, **string** *nas_table_create_str*, **string** *handles_table_create_str*, **string** *Users_table_create_str*, **string** *nas_table_insert_str*, **string** *Users_table_insert_str* and **string** *handles_table_alter_str*.

[LDF 2013.09.06.] Removed **bool** *standalone_handles* = *true*. **bool** *standalone_handle* (without trailing "s") is already declared in *glblvrbl.web*.

```
⟨ setupdbs global variables 68 ⟩ ≡
  string handles_database_name = "handlesystem_standalone";
  string gwirdsif_database_name = "gwirdsif";
  string gwirdcli_database_name = "gwirdcli";
  bool handles_database_exists = false;
  bool gwirdsif_database_exists = false;
  bool gwirdcli_database_exists = false;
  bool handles_database_created = false;
  bool gwirdsif_database_created = false;
  bool gwirdcli_database_created = false;
  bool drop = false; /* true */
;
  vector<string> prefix_vector;
  vector<pair<string, string>> institute_vector;
  vector<string> admin_vector;
  vector<string> user_vector;
  vector<string> group_vector;
  ofstream out_strm;
  ifstream license_strm;
  int irods_server_port = 1247;
  string nas_table_create_str;
  string handles_table_create_str;
  string Users_table_create_str;
  string handles_table_alter_str[2];
  string nas_table_insert_str;
  string Users_table_insert_str;
```

This code is used in section 93.

69. **extern** declarations for global variables [LDF 2013.09.03.]**Log**

[LDF 2013.09.03.] Added this section.

```
( setupdbs extern declarations for global variables 69 ) ≡
extern string handles_database_name;
extern string gwirdsif_database_name;
extern string gwirdcli_database_name;
extern bool handles_database_exists;
extern bool gwirdsif_database_exists;
extern bool gwirdcli_database_exists;
extern bool handles_database_created;
extern bool gwirdsif_database_created;
extern bool gwirdcli_database_created;
extern bool drop;
extern vector<string> prefix_vector;
extern vector<pair<string, string>> institute_vector;
extern vector<string> admin_vector;
extern vector<string> user_vector;
extern vector<string> group_vector;
extern ofstream out_strm;
extern ifstream license_strm;
extern int irods_server_port;
extern string nas_table_create_str;
extern string handles_table_create_str;
extern string Users_table_create_str;
extern string handles_table_alter_str[2];
extern string nas_table_insert_str;
extern string Users_table_insert_str;
```

This code is used in section 94.

70. Process command line options. [LDF 2013.09.02.]

```
( setupdbs process_command_line_options declaration 70 ) ≡
int process_command_line_options(int argc, char *argv[]);
```

This code is used in section 94.

71.

```
( setupdbs process_command_line_options definition 71 ) ≡
int process_command_line_options(int argc, char *argv[]){ bool DEBUG = false; /* true */
    int status = 0;
    int option_ctr;
    int digit_optind = 0;
```

See also sections 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, and 90.

This code is used in section 93.

72. Index constants. [LDF 2013.09.02.]

Log

[LDF 2013.09.03.] Added code for the --prefix option.
 [LDF 2013.09.04.] Added code for the --no-standalone-handle option.
 [LDF 2013.09.06.] Added code for the --irods-server-port option.

```
( setupdbs process_command_line_options definition 71 ) +≡
const unsigned short VERSION_INDEX = 0;
const unsigned short HELP_INDEX = 1;
const unsigned short DROP_INDEX = 2;
const unsigned short ADMIN_INDEX = 3;
const unsigned short USER_INDEX = 4;
const unsigned short GROUP_INDEX = 5;
const unsigned short HANDLES_DATABASE_NAME_INDEX = 6;
const unsigned short SERVER_DATABASE_NAME_INDEX = 7;
const unsigned short GWIRDSIF_DATABASE_NAME_INDEX = 8;
const unsigned short CLIENT_DATABASE_NAME_INDEX = 9;
const unsigned short GWIRDCLI_DATABASE_NAME_INDEX = 10;
const unsigned short PREFIX_INDEX = 11;
const unsigned short INSTITUTE_INDEX = 12;
const unsigned short NO_STANDALONE_HANDLE_INDEX = 13;
const unsigned short IRODS_SERVER_PORT_INDEX = 14;
```

73. Option struct. [LDF 2012.07.02.]

```
( setupdbs process_command_line_options definition 71 ) +≡
static struct option long_options[] = {{"version", 0, 0, 0}, {"help", 0, 0, 0}, {"drop", 0, 0, 0}, {"admin", 1, 0, 0}, {"user", 1, 0, 0}, {"group", 1, 0, 0}, {"handles-database-name", 1, 0, 0}, {"server-database-name", 1, 0, 0}, {"gwirdsif-database-name", 1, 0, 0}, {"client-database-name", 1, 0, 0}, {"gwirdcli-database-name", 1, 0, 0}, {"prefix", 1, 0, 0}, {"institute", 1, 0, 0}, {"no-standalone-handle", 0, 0, 0}, {"irods-server-port", 1, 0, 0}, {0, 0, 0, 0}}};
int option_index = 0;
int this_option_optind = optind ? optind : 1; while (1) { option_ctr = getopt_long_only(argc, argv, "", long_options, &option_index);
#if DEBUG_COMPILE
  if (DEBUG) {
    cerr << "option_ctr=" << option_ctr << endl << "option_index=" << option_index << endl;
    if (optarg) {
      cerr << "optarg=" << optarg << endl;
    } /* if (optarg) */
    /* if (DEBUG) */
  }
#endif /* DEBUG_COMPILE */
  if (option_ctr == -1) {
#if DEBUG_COMPILE
    if (DEBUG) {
      cerr << "No more option arguments." << endl;
    }
#endif /* DEBUG_COMPILE */
    break;
  }
}
```

74. Option. [LDF 2013.09.02.]

```
(setupdbs process_command_line_options definition 71) +≡
  else if (option_ctr == 0) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    cerr << "option_" << long_options[option_index].name;
    if (optarg) {
      cerr << "with_arg_" << optarg;
    }
    cerr << endl;
  }
#endif /* DEBUG_COMPILE */
```

75. version. [LDF 2013.09.04.]

Log

[LDF 2013.09.04.] Added this section.

```
(setupdbs process_command_line_options definition 71) +≡
  if (option_index == VERSION_INDEX) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    cerr << "'option_index'" << "==" << 'VERSION_INDEX' << endl;
  }
#endif /* DEBUG_COMPILE */
  cout << "setupdbs" << PACKAGE_VERSION << endl << "setupdbs is part of" <<
    PACKAGE_NAME << " " << PACKAGE_VERSION << endl << "Copyright (C) 2013" <<
    "Gesellschaft fuer wissenschaftliche Datenverarbeitung mbH Goettingen, Germany" <<
    endl << "Author: Laurence D. Finston" << endl;
  exit(0);
} /* if (option_index == VERSION_INDEX) */
```

76. help. [LDF 2013.09.04.]

Log

[LDF 2013.09.04.] Added this section.

```

⟨ setupdbs process_command_line_options definition 71 ⟩ +≡
  if (option_index ≡ HELP_INDEX) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    cerr << "option_index" << "==" << "HELP_INDEX" << endl;
  }
#endif /* DEBUG_COMPILE */
  cout << "setupdbs sets up the databases needed for " << PACKAGE_NAME <<
        PACKAGE_VERSION << endl << "Usage: setupdbs [OPTION...]" <<
        endl << "--version: Print out the package name and version and exit" <<
        endl << "--help: Print out this help message and exit" << endl <<
        "--drop: Databases will be dropped and recreated, if they already exist" <<
        endl << "--handles-database-name STRING: Name for handles database" << endl <<
        "--handlesystem-standalone (default: \\"handlesystem_standalone\\") for " <<
        "--standalone_handle_service" << endl << "standalone handle service" or \\"handlesystem\\"
        " for " << "non-standalone handle service" << endl << "--server-database-n-
ame: NAME for server-side database" << "(default: \wgwirdsif)" << endl <<
        "--gwirdsif-database-name: SYNONYM for previous option" << endl <<
        "--client-database-name: NAME for client-side database (default: \wgwirdcli)" <<
        endl << "--gwirdcli-database-name: SYNONYM for previous option" <<
        endl << "--prefix STRING: Handle prefix (default: \12345)" << endl <<
        "--institute: STRING: STRING, e.g., \\"ZZZZ:Test_Institute\"" << endl <<
        "--no-standalone-handle: Create database for a handle service" <<
        "registered with CNRI" << " (Corporation of National Research Initiatives)" << endl;
  exit(0);
} /* if (option_index ≡ HELP_INDEX) */

```

77. drop. [LDF 2013.09.02.]

Log

[LDF 2013.09.02.] Added this section.

```

⟨ setupdbs process_command_line_options definition 71 ⟩ +≡
  else
    if (option_index ≡ DROP_INDEX) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
      cerr << "option_index" << "==" << "DROP_INDEX" << endl;
    }
#endif /* DEBUG_COMPILE */
    drop = true;
} /* else if (option_index ≡ DROP_INDEX) */

```

78. admin. [LDF 2013.09.04.]

Log

[LDF 2013.09.04.] Added this section.

```
(setupdbs process_command_line_options definition 71) +≡
else
    if (option_index ≡ ADMIN_INDEX) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        cerr << "option_index'" << "==" << 'ADMIN_INDEX' << endl << "optarg==" << optarg << endl;
    }
#endif /* DEBUG_COMPILE */
    admin_vector.push_back(string(optarg));
} /* else if (option_index ≡ ADMIN_INDEX) */
```

79. user. [LDF 2013.09.04.]

Log

[LDF 2013.09.04.] Added this section.

```
(setupdbs process_command_line_options definition 71) +≡
else
    if (option_index ≡ USER_INDEX) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        cerr << "option_index'" << "==" << 'USER_INDEX' << endl << "optarg==" << optarg << endl;
    }
#endif /* DEBUG_COMPILE */
    user_vector.push_back(string(optarg));
} /* else if (option_index ≡ USER_INDEX) */
```

80. group. [LDF 2013.09.05.]

Log

[LDF 2013.09.05.] Added this section.

```
(setupdbs process_command_line_options definition 71) +≡
else
    if (option_index ≡ GROUP_INDEX) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        cerr << "option_index'" << "==" << 'GROUP_INDEX' << endl << "optarg==" << optarg << endl;
    }
#endif /* DEBUG_COMPILE */
    group_vector.push_back(string(optarg));
} /* else if (option_index ≡ GROUP_INDEX) */
```

81. handles-database-name. [LDF 2013.09.02.]

Log

[LDF 2013.09.02.] Added this section.

```
(setupdbs process-command-line-options definition 71) +≡
else
  if (option_index ≡ HANDLES_DATABASE_NAME_INDEX) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    cerr << "'option_index'" << "==" << 'HANDLES_DATABASE_NAME_INDEX' << endl;
    if (optarg) cerr << "optarg" << optarg << endl;
  }
#endif /* DEBUG_COMPILE */
  handles_database_name = optarg;
} /* else if (option_index ≡ HANDLES_DATABASE_NAME_INDEX) */
```

82. server-database-name or gwirdsif-database-name. [LDF 2013.09.02.]

Log

[LDF 2013.09.02.] Added this section.

```
(setupdbs process-command-line-options definition 71) +≡
else
  if (option_index ≡ SERVER_DATABASE_NAME_INDEX ∨ option_index ≡ GWIRDSIF_DATABASE_NAME_INDEX)
  {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    cerr << "'option_index'" << "==" << 'SERVER_DATABASE_NAME_INDEX' << "or" <<
        "==" << 'GWIRDSIF_DATABASE_NAME_INDEX' << endl;
    if (optarg) cerr << "optarg" << optarg << endl;
  }
#endif /* DEBUG_COMPILE */
  gwirdsif_database_name = optarg;
} /* else if */
```

83. client-database-name or gwirdcli-database-name. [LDF 2013.09.02.]

Log

[LDF 2013.09.02.] Added this section.

```
( setupdbs process_command_line_options definition 71 ) +≡
else
  if (option_index ≡ CLIENT_DATABASE_NAME_INDEX ∨ option_index ≡ GWIRDCLI_DATABASE_NAME_INDEX)
  {
#if DEBUG_COMPILE
    if (DEBUG) {
      cerr << "'option_index'" << "==" << 'CLIENT_DATABASE_NAME_INDEX' << "or" <<
      "'GWIRDCLI_DATABASE_NAME_INDEX' ." << endl;
      if (optarg) cerr << " optarg==" << optarg << endl;
    }
#endif /* DEBUG_COMPILE */
    gwirdcli_database_name = optarg;
  } /* else if */
```

84. prefix. [LDF 2013.09.03.]

Log

[LDF 2013.09.03.] Added this section.

```
( setupdbs process_command_line_options definition 71 ) +≡
else
  if (option_index ≡ PREFIX_INDEX) {
#if DEBUG_COMPILE
    if (DEBUG) {
      cerr << "'option_index'" << "==" << 'PREFIX_INDEX' . " << endl;
      if (optarg) cerr << " optarg==" << optarg << endl;
    }
#endif /* DEBUG_COMPILE */
    prefix_vector.push_back(string(optarg));
  } /* else if (option_index ≡ PREFIX_INDEX) */
```

85. institute. [LDF 2013.09.03.]

Log

[LDF 2013.09.03.] Added this section.

```
( setupdbs process_command_line_options definition 71 ) +≡
else
  if (option_index ≡ INSTITUTE_INDEX) {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    cerr ≪ "option_index" ≪ "==" ≪ 'INSTITUTE_INDEX' . " ≪ endl;
    if (optarg) cerr ≪ "optarg" ≪ optarg ≪ endl;
  }
#endif /* DEBUG_COMPILE */
  string temp_str = optarg;
  string temp_str_1;
  string temp_str_2;
  size_t pos = temp_str.find(":");
  if (pos ≡ string::npos ∨ pos ≡ 0 ∨ pos ≡ temp_str.size() - 1) {
    cerr ≪ "[setupdbs] WARNING! Invalid argument to '--institute' option: " ≪ optarg ≪
      endl ≪ "Not adding institute." ≪ endl ≪ "Usage: --institute STRING:STRING" ≪
      endl ≪ "Continuing." ≪ endl;
  }
  else {
    temp_str_1 = temp_str;
    temp_str_1.erase(0, pos + 1);
    temp_str.erase(pos);
#endif /* DEBUG_COMPILE */
  if (DEBUG) {
    cerr ≪ "temp_str" ≪ endl ≪ temp_str ≪ endl ≪ "temp_str_1" ≪ endl ≪ temp_str_1 ≪ endl;
  }
#endif /* DEBUG_COMPILE */
  pos = temp_str.find(":");
  if (pos ≡ string::npos) pos = temp_str_1.find(":");
  if (pos ≠ string::npos)
    cerr ≪ "[setupdbs] WARNING! Invalid argument to '--institute' option: " ≪
      optarg ≪ endl ≪ "Not adding institute." ≪ endl ≪
      "Usage: --institute STRING:STRING" ≪ endl ≪ "Continuing." ≪ endl;
  else {
    institute_vector.push_back(make_pair(temp_str, temp_str_1));
  }
} /* else */
} /* else if (option_index ≡ INSTITUTE_INDEX) */
```

86. no-standalone-handle. [LDF 2013.09.04.]**Log**

[LDF 2013.09.04.] Added this section.

```
(setupdbs process_command_line_options definition 71) +≡
else
    if (option_index ≡ NO_STANDALONE_HANDLE_INDEX) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        cerr << "option_index'"_u" << "=="_u"NO_STANDALONE_HANDLE_INDEX'" . " << endl;
    }
#endif /* DEBUG_COMPILE */
    standalone_handle = false;
} /* else if (option_index ≡ NO_STANDALONE_HANDLE_INDEX) */
```

87. irods-server-port. [LDF 2013.09.06.]**Log**

[LDF 2013.09.06.] Added this section.

```
(setupdbs process_command_line_options definition 71) +≡
else
    if (option_index ≡ IRODS_SERVER_PORT_INDEX) {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        cerr << "option_index'"_u" << "=="_u"IRODS_SERVER_PORT_INDEX'" . " << endl;
        if (optarg) cerr << "optarg"_u" << optarg << endl;
    }
#endif /* DEBUG_COMPILE */
    irods_server_port = atoi(optarg);
} /* else if (option_index ≡ IRODS_SERVER_PORT_INDEX) */
```

88. Invalid option_index value. [LDF 2013.09.02.]

```
(setupdbs process_command_line_options definition 71) +≡
else {
    cerr << "WARNING! In 'process_command_line_options': " << endl <<
        "'option_index'_has_invalid_value:_u" << option_index << endl <<
        "Will try to continue." << endl;
}
} /* else if (option_ctr ≡ 0) */
```

89. Ambiguous option. [LDF 2013.09.02.]

```
(setupdbs process_command_line_options definition 71) +≡
else
    if (option_ctr ≡ '?') {
        cerr << "WARNING! In 'process_command_line_options': " << endl <<
            "'getopt_long_only'_returned_ambiguous_match._u" << "Breaking." << endl;
        break;
} /* else if (option_ctr ≡ '?') */
```

90. Invalid option. [LDF 2013.09.02.]

```
< setupdbs process_command_line_options definition 71 > +≡
  else {
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    cerr << "'getopt_long_only'\u2014returned\u2014invalid\u2014option." << endl;
  }
#endif /* DEBUG_COMPILE */
}
} /* while */
return 0; } /* End of process_command_line_options definition */
```

91. Delete pointers and clear vectors (*delete_and_clear*). [LDF 2013.09.04.]

Log

[LDF 2013.09.04.] Added this function.

```
< delete_and_clear declaration 91 > ≡
void delete_and_clear(MYSQL_RES **result_array, size_t result_array_size, vector<unsigned
int *> &row_ctr_vector, vector<unsigned int *> &field_ctr_vector, vector<long int *>
&affected_rows_vector, vector<string> &query_vector);
```

This code is used in section 94.

92.

```

⟨ delete_and_clear definition 92 ⟩ ≡
void delete_and_clear(MYSQL_RES **result_array, size_t result_array_size, vector<⟨unsigned int
    *⟩ &row_ctr_vector, vector<⟨unsigned int *⟩ &field_ctr_vector, vector<⟨long int *⟩
    &affected_rows_vector, vector<⟨string⟩ &query_vector)
{
    for (int i = 0; i < result_array_size; ++i) {
        if (result_array[i]) {
            mysql_free_result(result_array[i]);
            result_array[i] = 0;
        }
    }
    for (vector<⟨unsigned int *⟩::iterator iter = row_ctr_vector.begin(); iter ≠ row_ctr_vector.end();
          ++iter) {
        delete *iter;
        *iter = 0;
    }
    row_ctr_vector.clear();
    for (vector<⟨unsigned int *⟩::iterator iter = field_ctr_vector.begin(); iter ≠ field_ctr_vector.end();
          ++iter) {
        delete *iter;
        *iter = 0;
    }
    field_ctr_vector.clear();
    for (vector<⟨long int *⟩::iterator iter = affected_rows_vector.begin(); iter ≠ affected_rows_vector.end(); ++iter) {
        delete *iter;
        *iter = 0;
    }
    affected_rows_vector.clear();
    query_vector.clear();
    return;
}      /* End of delete_and_clear definition */

```

This code is used in section 93.

93. This is what's compiled. [LDF 2013.09.03.]

```

using namespace std;
⟨ Include files 3 ⟩
using namespace gwrdifpk;
⟨ setupdbs global variables 68 ⟩
⟨ setupdbs process_command_line_options definition 71 ⟩
⟨ delete_and_clear definition 92 ⟩

```

94. This is what's written to the header file **stpclopt.h**. [LDF 2013.09.03.]

```

⟨ stpclopt.h 94 ⟩ ≡
#ifndef STPCLOPT_H
#define STPCLOPT_H 1
⟨ setupdbs process_command_line_options declaration 70 ⟩
⟨ setupdbs extern declarations for global variables 69 ⟩
⟨ delete_and_clear declaration 91 ⟩
#endif

```

95. Include files. [LDF 2013.09.04.]

```
<Include files 3> +≡
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>    /* POSIX threads */
#include <gcrypt.h>      /* for gcry-control */
#include <gnutls/gnutls.h>
#include <gnutls/x509.h>
#include <mysql.h>
#include <expat.h>
#include <iomanip>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <getopt.h>
#include <map>
#include <vector>
#include <deque>
#include <set>
#include <stack>
#include <utility>
#if HAVE_CONFIG_H
#include <config.h>
#endif
#include "rspercds.h++"
#include "glblcnst.h++"
#include "glblvrbl.h++"
#include "excptntp.h++"
#include "utilfncts.h++"
#include "groupntp.h++"
#include "hndlvltp.h++"
#include "irdsavtp.h++"
#include "helper.h++"
#include "tanfncts.h++"
#include "pidfncts.h++"
#include "parser.h++"
#include "scanner.h++"
#include "rspnstp.h++"
#include "irdsobtp.h++"
#include "hdltype.h++"
#include "dcmtddtp.h++"
#include "x509cert.h++"
#include "dstngnmt.h++"
#include "usrinftp.h++"
#include "scprpmtp.h++"
#include "stpclopt.h++"
```

96. Create databases (*create_databases*). [LDF 2013.09.04.]

Log

[LDF 2013.09.04.] Added this function.

$\langle \text{create_databases} \text{ declaration } 96 \rangle \equiv$
int *create_databases*(Scan_Parse_Parameter_Type &*param*);

This code is used in section 111.

97.

```

⟨ create_databases definition 97 ⟩ ≡
int create_databases(Scan_Parse_Parameter_Type &param){ bool DEBUG = false;      /* true */
    set_debug_level(DEBUG, 0, 0);
    int status;
    stringstream sql_strm;
    stringstream temp_strm;
    string temp_str;
    size_t pos = 0;
    MYSQL_RES *result = 0;
    MYSQL_ROW curr_row;
    size_t result_array_size = 16;
    MYSQL_RES *result_array[16] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
    unsigned int row_ctr = 0;
    unsigned int field_ctr = 0;
    long int affected_rows = 0;
    vector<string> query_vector;
    vector<unsigned int *> row_ctr_vector;
    vector<unsigned int *> field_ctr_vector;
    vector<long int *> affected_rows_vector;
    string comma_str;
#define DEBUG_COMPILE
    if (DEBUG) {
        cerr << "Entering 'create_databases'." << endl;
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    sql_strm << "select schema_name from information_schema.schemata" <<
        "where schema_name = " << handles_database_name << ',' << "or schema_name = " <<
        gwirdsif_database_name << ',' << "or schema_name = " << gwirdcli_database_name << '';
#define DEBUG_COMPILE
    if (DEBUG) {
        cerr << "In 'create_databases': " << sql_strm.str() << " == " << sql_strm.str() << endl;
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    status = param.submit_mysql_query(sql_strm.str(), result, &row_ctr, &field_ctr);
    if (status != 0) {
        cerr << "ERROR! In 'create_databases': Scan_Parse_Parameter_Type:\\" <<
            "submit_mysql_query" << " failed, returning " << status <<
            "." << endl << "Failed to query for database names." << endl <<
            "Exiting 'create_databases' unsuccessfully with return value 1." << endl;
        if (result) {
            mysql_free_result(result);
            result = 0;
        }
        return 1;
    } /* if (status != 0) */

```

See also sections 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, and 109.

This code is used in section 110.

98.

```
< create_databases definition 97 > +≡
#ifndef DEBUG_COMPILE
    else
        if (DEBUG) {
            cerr << "In 'create_databases': Scan_Parse_Parameter_Type::submit_mysql_query' <<
                "succeeded, returning 0." << endl << "Queried for database names successfully." <<
                endl << "'row_ctr' == " << row_ctr << endl;
        } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

99.

```
< create_databases definition 97 > +≡
temp_strm.str("");
if (row_ctr == 0) {
    cerr << "Databases don't already exist. Will create." << endl;
    handles_database_created = gwirdsif_database_created = gwirdcli_database_created = true;
} /* if (row_ctr == 0) */
```

100.

```

⟨ create_databases definition 97 ⟩ +≡
  else {
#ifndef DEBUG_COMPILE
    if (DEBUG) {
      cerr << "At least one database already exists. Will check." << endl;
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
    for (int i = 0; i < row_ctr; ++i) {
      if ((curr_row = mysql_fetch_row(result)) == 0) {
        cerr << "ERROR! In 'create_databases': mysql_fetch_row failed:" <<
              endl << mysql_error(param.mysql_ptr) << endl <<
              "Exiting 'create_databases' unsuccessfully with return value 1." << endl;
        mysql_free_result(result);
        return 1;
      } /* if (curr_row = mysql_fetch_row(result)) == 0 */
#endif DEBUG_COMPILE
    else
      if (DEBUG) {
        cerr << "In 'create_databases': mysql_fetch_row succeeded." << endl;
      } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
    if (curr_row[0] & strlen(curr_row[0])) temp_strm << curr_row[0] << " ";
    } /* for */
    if (temp_strm.str().empty()) /* This should never happen. [LDF 2013.09.03.] */
    {
      cerr << "Databases don't already exist. Will create." << endl;
      handles_database_created = gwirdsif_database_created = gwirdcli_database_created = true;
    }
  else {
    temp_str = temp_strm.str();
    pos = temp_str.find(handles_database_name);
    if (pos != string::npos) handles_database_exists = true;
    pos = temp_str.find(gwirdsif_database_name);
    if (pos != string::npos) gwirdsif_database_exists = true;
    pos = temp_str.find(gwirdcli_database_name);
    if (pos != string::npos) gwirdcli_database_exists = true;
  } /* else */
} /* else */
temp_strm.str("");
sql_strm.str("");
mysql_free_result(result);
result = 0;

```

101.

```

⟨ create_databases definition 97 ⟩ +≡
  if (handles_database_exists) {
    cerr ≪ "Handles_database\"" ≪ handles_database_name ≪ "\"already_exists." ≪ endl;
  }
  else {
    cerr ≪ "Handles_database\"" ≪ handles_database_name ≪ "\"doesn't already_exist." ≪
        endl;
    temp_strm ≪ "create_database" ≪ handles_database_name;
    query_vector.push_back(temp_strm.str());
    row_ctr_vector.push_back(0);
    field_ctr_vector.push_back(0);
    affected_rows_vector.push_back(0);
    temp_strm.str("");
    handles_database_created = true;
  }
  if (gwirdsif_database_exists) {
    cerr ≪ "Server-side_database\"" ≪ gwirdsif_database_name ≪ "\"already_exists." ≪ endl;
  }
  else {
    cerr ≪ "Server-side_database\"" ≪ gwirdsif_database_name ≪ "\"doesn't already_exist." ≪
        endl;
    temp_strm ≪ "create_database" ≪ gwirdsif_database_name;
    query_vector.push_back(temp_strm.str());
    row_ctr_vector.push_back(0);
    field_ctr_vector.push_back(0);
    affected_rows_vector.push_back(0);
    temp_strm.str("");
    gwirdsif_database_created = true;
  }
  if (gwirdcli_database_exists) {
    cerr ≪ "Client-side_database\"" ≪ gwirdcli_database_name ≪ "\"already_exists." ≪ endl;
  }
  else {
    cerr ≪ "Client-side_database\"" ≪ gwirdcli_database_name ≪ "\"doesn't already_exist." ≪
        endl;
    temp_strm ≪ "create_database" ≪ gwirdcli_database_name;
    query_vector.push_back(temp_strm.str());
    row_ctr_vector.push_back(0);
    field_ctr_vector.push_back(0);
    affected_rows_vector.push_back(0);
    temp_strm.str("");
    gwirdcli_database_created = true;
  }
}

```

102.

```

⟨ create_databases definition 97 ⟩ +≡
  if (drop ∧ (handles_database_exists ∨ gwirdsif_database_exists ∨ gwirdcli_database_exists)) {
    temp_strm.str("");
    if (handles_database_exists) {
      cerr << "Handles_database\" " << handles_database_name << "\"_exists." << "Will_drop." <<
        endl;
      temp_strm << "drop_database" << handles_database_name;
      query_vector.push_back(temp_strm.str());
      row_ctr_vector.push_back(0);
      field_ctr_vector.push_back(0);
      affected_rows_vector.push_back(0);
      temp_strm.str("");
      temp_strm << "create_database" << handles_database_name;
      query_vector.push_back(temp_strm.str());
      row_ctr_vector.push_back(0);
      field_ctr_vector.push_back(0);
      affected_rows_vector.push_back(0);
      temp_strm.str("");
      handles_database_created = true;
    }
    if (gwirdsif_database_exists) {
      cerr << "Server-side_database\" " << gwirdsif_database_name << "\"_exists." <<
        "Will_drop." << endl;
      temp_strm << "drop_database" << gwirdsif_database_name;
      query_vector.push_back(temp_strm.str());
      row_ctr_vector.push_back(0);
      field_ctr_vector.push_back(0);
      affected_rows_vector.push_back(0);
      temp_strm.str("");
      temp_strm << "create_database" << gwirdsif_database_name;
      query_vector.push_back(temp_strm.str());
      row_ctr_vector.push_back(0);
      field_ctr_vector.push_back(0);
      affected_rows_vector.push_back(0);
      temp_strm.str("");
      gwirdsif_database_created = true;
    }
    if (gwirdcli_database_exists) {
      cerr << "Client-side_database\" " << gwirdcli_database_name << "\"_exists." <<
        "Will_drop." << endl;
      temp_strm << "drop_database" << gwirdcli_database_name;
      query_vector.push_back(temp_strm.str());
      row_ctr_vector.push_back(0);
      field_ctr_vector.push_back(0);
      affected_rows_vector.push_back(0);
      temp_strm.str("");
      temp_strm << "create_database" << gwirdcli_database_name;
      query_vector.push_back(temp_strm.str());
      row_ctr_vector.push_back(0);
      field_ctr_vector.push_back(0);
      affected_rows_vector.push_back(0);
    }
  }
}

```

```

temp_strm.str("");
gwirdcli_database_created = true;
}

```

103.

```

⟨ create_databases definition 97 ⟩ +≡
}      /* if (drop) */

```

104.

```

⟨ create_databases definition 97 ⟩ +≡
else      /* drop ≡ false */
{
    temp_strm.str("");
    if (handles_database_exists) {
        cerr << "Handles_database\""
             << handles_database_name << "\"already_exists."
             << "Not_dropping_and_recreating." << endl;
    }
    else {
        cerr << "Handles_database\""
             << handles_database_name << "\"doesn't_already_exist."
             << "Creating." << endl;
    }
    if (gwirdsif_database_exists) {
        cerr << "Server-side_database\""
             << gwirdsif_database_name << "\"already_exists."
             << "Not_dropping_and_recreating." << endl;
    }
    else {
        cerr << "Server-side_database\""
             << gwirdsif_database_name << "\""
             << "doesn't_already_exist_Creating." << endl;
    }
    if (gwirdcli_database_exists) {
        cerr << "Client-side_database\""
             << gwirdcli_database_name << "\"already_exists."
             << "Not_dropping_and_recreating." << endl;
    }
    else {
        cerr << "Client-side_database\""
             << gwirdcli_database_name << "\""
             << "doesn't_already_exist_Creating." << endl;
    }
}      /* else (drop ≡ false) */

```

105.

```
<create_databases definition 97> +≡
  if (query_vector.size() == 0) /* Nothing to do. Exit. [LDF 2013.09.03.] */
  {
    cerr << "In 'create_databases': query_vector.size() == 0. No SQL\"
         commands to execute." << endl;
  if (drop == true)
    cerr << "'drop' == 'true'. This should never happen." << endl <<
        "Exiting 'create_databases' unsuccessfully with return value 1." << endl;
    delete_and_clear(result_array, result_array_size, row_ctr_vector, field_ctr_vector, affected_rows_vector,
                     query_vector);
    return 1;
  }
  else {
    cerr << "Exiting 'create_databases' with return value 2." << endl;
    delete_and_clear(result_array, result_array_size, row_ctr_vector, field_ctr_vector, affected_rows_vector,
                     query_vector);
    return 2;
  }
} /* if (query_vector.size() == 0) */
```

106.

```
<create_databases definition 97> +≡
#ifndef DEBUG_COMPILE
  if (DEBUG) {
    cerr << "query_vector.size() == " << query_vector.size() << endl << "'query_vector':"
    for (vector<string>::iterator iter = query_vector.begin(); iter != query_vector.end(); ++iter) {
      cerr << *iter << endl;
    }
    cerr << endl;
  } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
```

107.

Log

[LDF 2013.09.04.] Added this section.

```
<create_databases definition 97> +≡
  for (vector<string>::iterator iter = query_vector.begin(); iter != query_vector.end(); ++iter) {
    out_strm << *iter << ";" << endl << endl;
  }
  out_strm << endl;
```

108.

```
< create_databases definition 97 > +≡
status = param.submit_mysql_queries(query_vector, result_array, row_ctr_vector, field_ctr_vector,
    affected_rows_vector, false);
if (status ≠ 0) {
    cerr ≪ "ERROR! In 'create_databases': 'submit_mysql_queries' failed, returning " ≪
        status ≪ ". " ≪ endl ≪ "Exiting 'create_databases' unsuccessfully with return value \
        1." ≪ endl;
    delete_and_clear(result_array, result_array_size, row_ctr_vector, field_ctr_vector, affected_rows_vector,
        query_vector);
    out_strm ≪ "/* Failed */" ≪ endl;
    return 1;
} /* if (status ≠ 0) */
```

109.

```
< create_databases definition 97 > +≡
#ifndef DEBUG_COMPILE
else
    if (DEBUG) {
        cerr ≪ "In 'create_databases': 'submit_mysql_queries' succeeded, returning 0. " ≪
            endl ≪ "Dropped (possibly) and created databases successfully." ≪ endl;
    } /* else if (DEBUG) */
#endif /* DEBUG_COMPILE */
out_strm ≪ "/* Succeeded */" ≪ endl ≪ endl ≪
    "*****" ≪ endl ≪ endl;
delete_and_clear(result_array, result_array_size, row_ctr_vector, field_ctr_vector, affected_rows_vector,
    query_vector);
if (result) {
    mysql_free_result(result);
    result = 0;
}
sql_strm.str("");
#ifndef DEBUG_COMPILE
    if (DEBUG) {
        cerr ≪ "Exiting 'create_databases' successfully with return value 1." ≪ endl;
    } /* if (DEBUG) */
#endif /* DEBUG_COMPILE */
return 0; } /* End of create_databases definition */
```

110. This is what's compiled. [LDF 2013.09.04.]

```
using namespace std;
< Include files 3 >
using namespace gwrdifpk;
< create_databases definition 97 >
```

111. This is what's written to the header file `stpcrdbs.h`. [LDF 2013.09.04.]

```
<stpcrdbs.h 111>≡
#ifndef STPCRDBS_H
#define STPCRDBS_H 1
    <create_databases declaration 96>
#endif
```

112. Index.

`ADMIN_INDEX:` 72, 78.
`admin_vector:` 40, 44, 49, 68, 69, 78.
`affected_rows:` 20, 28, 36, 97.
`affected_rows_vector:` 20, 21, 22, 23, 28, 29, 31, 36, 37, 38, 39, 42, 43, 45, 46, 47, 48, 51, 52, 53, 54, 55, 56, 57, 58, 61, 62, 63, 91, 92, 97, 101, 102, 105, 108, 109.
`argc:` 5, 6, 70, 71, 73.
`argv:` 5, 6, 70, 71, 73.
`atexit:` 5.
`atoi:` 87.
`begin:` 8, 21, 22, 28, 36, 38, 40, 42, 49, 51, 56, 59, 60, 62, 92, 106, 107.
`buffer:` 5, 7.
`cerr:` 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 16, 20, 21, 22, 23, 28, 29, 30, 31, 36, 49, 59, 61, 62, 63, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 97, 98, 99, 100, 101, 102, 104, 105, 106, 108, 109.
`clear:` 92.
`CLIENT_DATABASE_NAME_INDEX:` 72, 83.
`close:` 4, 7.
`comma_str:` 20, 28, 36, 38, 40, 42, 44, 49, 51, 56, 97.
`comment_vector:` 20, 21, 22, 36, 38, 42, 45, 47, 51, 54, 56, 58, 62.
`cout:` 75, 76.
`create_databases:` 9, 96, 97, 109.
`create_tables_archive:` 12.
`create_tables_dublin_core:` 13.
`create_tables_gwirdcli:` 14, 19, 20, 23.
`create_tables_gwirdsif:` 11, 35, 36, 63.
`create_tables_handles:` 10, 27, 28, 31.
`curr_row:` 20, 28, 36, 97, 100.
`DEBUG:` 5, 6, 7, 8, 9, 20, 21, 22, 23, 28, 29, 30, 31, 36, 59, 62, 63, 71, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 90, 97, 98, 100, 106, 109.
`DEBUG_COMPILE:` 5, 6, 7, 8, 9, 20, 21, 22, 23, 28, 29, 30, 31, 36, 59, 62, 63, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 90, 97, 98, 100, 106, 109.
`default_institute_id:` 40, 42.

`default_prefix_id:` 40, 42.
`delete_and_clear:` 21, 23, 29, 31, 61, 63, 91, 92, 105, 108, 109.
`digit_optind:` 71.
`Distinguished_Name:` 40, 42.
`drop:` 68, 69, 77, 102, 103, 104, 105.
`DROP_INDEX:` 72, 77.
`empty:` 100.
`end:` 8, 21, 22, 28, 36, 38, 40, 42, 49, 51, 56, 59, 60, 62, 92, 106, 107.
`endl:` 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 16, 20, 21, 22, 23, 28, 29, 30, 31, 36, 37, 38, 39, 40, 43, 46, 47, 48, 49, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 97, 98, 99, 100, 101, 102, 104, 105, 106, 107, 108, 109.
`eof:` 7.
`erase:` 85.
`exit:` 5, 6, 7, 9, 10, 11, 12, 13, 14, 16, 29, 75, 76.
`false:` 5, 20, 21, 28, 29, 36, 61, 68, 71, 86, 97, 104, 108.
`field_ctr:` 20, 28, 36, 97.
`field_ctr_vector:` 20, 21, 23, 28, 29, 31, 36, 37, 38, 39, 42, 43, 45, 46, 47, 48, 51, 52, 53, 54, 55, 56, 57, 58, 61, 63, 91, 92, 97, 101, 102, 105, 108, 109.
`find:` 85, 100.
`finish:` 4, 5.
`first:` 36.
`gcry_control:` 3, 18, 26, 34, 67, 95.
`get_datestamp:` 7.
`getenv:` 7.
`getopt_long_only:` 73.
`good:` 7.
`GROUP_INDEX:` 72, 80.
`group_vector:` 56, 68, 69, 80.
`gwirdcli_database_created:` 14, 68, 69, 99, 100, 101, 102.
`gwirdcli_database_exists:` 68, 69, 100, 101, 102, 104.
`gwirdcli_database_name:` 14, 20, 22, 68, 69, 83, 97, 100, 101, 102, 104.
`GWIRDCLI_DATABASE_NAME_INDEX:` 72, 83.
`gwirdsif_database_created:` 11, 13, 68, 69, 99,

100, 101, 102.
gwirdsif_database_exists: 68, 69, 100, 101, 102, 104.
gwirdsif_database_name: 11, 12, 13, 36, 62, 68, 69, 82, 97, 100, 101, 102, 104.
GWIRDSIF_DATABASE_NAME_INDEX: 72, 82.
gwrdifpk: 17, 24, 32, 65, 93, 110.
handle_password_encrypted: 40, 42.
handle_username: 40, 42.
handles_database_created: 10, 68, 69, 99, 100, 101, 102.
handles_database_exists: 68, 69, 100, 101, 102, 104.
handles_database_name: 10, 28, 30, 68, 69, 81, 97, 100, 101, 102, 104.
HANDLES_DATABASE_NAME_INDEX: 72, 81.
handles_table_alter_str: 21, 28, 68, 69.
handles_table_create_str: 21, 28, 68, 69.
HAVE_CONFIG_H: 3, 18, 26, 34, 67, 95.
HELP_INDEX: 72, 76.
i: 22, 36, 38, 49, 62, 92, 100.
ifstream: 68, 69.
INSTITUTE_INDEX: 72, 85.
institute_set: 68.
institute_vector: 36, 68, 69, 85.
ios_base: 7.
irods_default_resource: 40, 42.
irods_homedir: 40, 42.
irods_password_encrypted: 40, 42.
irods_password_encrypted_timestamp: 40, 42.
irods_server_port: 68, 69, 87.
IRODS_SERVER_PORT_INDEX: 72, 87.
irods_zone: 40, 42.
is_open: 4, 7.
iter: 8, 21, 22, 28, 36, 38, 40, 42, 49, 51, 56, 59, 60, 62, 92, 106, 107.
iterator: 8, 21, 22, 28, 36, 38, 40, 42, 49, 51, 56, 59, 60, 62, 92, 106, 107.
j: 22, 62.
left: 22, 62.
license strm: 7, 68, 69.
long_options: 73, 74.
main: 5, 16.
make_pair: 85.
memset: 5, 7.
mysql_error: 100.
mysql_fetch_row: 100.
mysql_free_result: 92, 97, 100, 109.
mysql_ptr: 100.
MYSQL_RES: 20, 28, 36, 91, 92, 97.
MYSQL_ROW: 20, 28, 36, 97.
name: 74.
nas_table_create_str: 21, 28, 68, 69.
nas_table_insert_str: 21, 28, 68, 69.

NO_STANDALONE_HANDLE_INDEX: 72, 86.
npos: 85, 100.
ofstream: 68, 69.
open: 7.
optarg: 73, 74, 78, 79, 80, 81, 82, 83, 84, 85, 87.
optind: 73.
option: 73.
option_ctrl: 71, 73, 74, 88, 89.
option_index: 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88.
out_strm: 4, 7, 60, 61, 63, 68, 69, 107, 108, 109.
PACKAGE_NAME: 75, 76.
PACKAGE_VERSION: 75, 76.
pair: 36, 68, 69.
param: 9, 10, 11, 12, 13, 14, 19, 20, 21, 27, 28, 29, 35, 36, 61, 96, 97, 100, 108.
pos: 85, 97, 100.
PREFIX_INDEX: 72, 84.
prefix_set: 68.
prefix_vector: 8, 28, 38, 44, 68, 69, 84.
process_command_line_options: 6, 70, 71, 90.
public_key_id: 40, 42.
push_back: 8, 20, 21, 28, 36, 37, 38, 39, 42, 43, 45, 46, 47, 48, 51, 52, 53, 54, 55, 56, 57, 58, 78, 79, 80, 84, 85, 101, 102.
query_vector: 20, 21, 23, 28, 29, 31, 36, 37, 38, 39, 42, 43, 45, 46, 47, 48, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 63, 91, 92, 97, 101, 102, 105, 106, 107, 108, 109.
read: 7.
result: 20, 28, 36, 97, 100, 109.
result_array: 20, 21, 23, 28, 29, 31, 36, 61, 63, 91, 92, 97, 105, 108, 109.
result_array_size: 20, 21, 23, 28, 29, 31, 36, 61, 63, 91, 92, 97, 105, 108, 109.
row_ctr: 20, 28, 36, 97, 98, 99, 100.
row_ctr_vector: 20, 21, 23, 28, 29, 31, 36, 37, 38, 39, 42, 43, 45, 46, 47, 48, 51, 52, 53, 54, 55, 56, 57, 58, 61, 63, 91, 92, 97, 101, 102, 105, 108, 109.
Scan_Parse_Parameter_Type: 9, 19, 20, 27, 28, 35, 36, 96, 97.
second: 36.
SERVER_DATABASE_NAME_INDEX: 72, 82.
set: 68.
set_debug_level: 5, 20, 28, 36, 97.
sets: 68.
setw: 22, 62.
size: 8, 21, 22, 28, 29, 36, 44, 49, 59, 62, 85, 105, 106.
sql_strm: 20, 21, 23, 28, 31, 36, 37, 38, 39, 40, 42, 43, 44, 45, 46, 47, 48, 49, 51, 52, 53, 54,

55, 56, 57, 58, 63, 97, 100, 109.
standalone_handle: 28, 68, 86.
standalone_handles: 68.
status: 5, 6, 9, 10, 11, 12, 13, 14, 20, 21, 28,
29, 36, 61, 71, 97, 108.
std: 17, 24, 32, 65, 93, 110.
STPCDCLI_H: 25.
STPCDHDL_H: 33.
STPCDSIF_H: 66.
STPCLOPT_H: 94.
STPCRDBS_H: 111.
str: 20, 21, 22, 23, 28, 31, 36, 37, 38, 39, 42, 43,
45, 46, 47, 48, 51, 52, 53, 54, 55, 56, 57, 58, 62,
63, 97, 99, 100, 101, 102, 104, 109.
strcat: 7.
strcpy: 7.
string: 8, 20, 21, 28, 36, 38, 40, 42, 49, 51, 56,
59, 60, 68, 69, 78, 79, 80, 84, 85, 91, 92,
97, 100, 106, 107.
stringstream: 20, 28, 36, 97.
strlen: 7, 100.
submit_mysql_queries: 21, 29, 61, 108.
submit_mysql_query: 97.
temp_ptr: 5, 7.
temp_str: 85, 97, 100.
temp_str_1: 85.
temp_str_2: 85.
temp_strm: 20, 22, 28, 36, 62, 97, 99, 100,
101, 102, 104.
this_option_optind: 73.
true: 5, 10, 11, 13, 14, 20, 28, 36, 68, 71, 77,
97, 99, 100, 101, 102, 105.
trunc: 7.
user_id: 40, 41, 42, 50.
USER_INDEX: 72, 79.
user_set: 68.
user_vector: 42, 49, 51, 68, 69, 79.
username: 40, 42.
Users_table_create_str: 21, 39, 68, 69.
Users_table_insert_str: 21, 42, 68, 69.
vector: 8, 20, 21, 22, 28, 36, 38, 40, 42, 49, 51,
56, 59, 60, 62, 68, 69, 91, 92, 97, 106, 107.
vectors: 68.
VERSION_INDEX: 72, 75.

⟨ Garbage 64 ⟩ Used in section 65.
⟨ Include files 3, 18, 26, 34, 67, 95 ⟩ Used in sections 17, 24, 32, 65, 93, and 110.
⟨ Main 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 ⟩ Used in section 17.
⟨ setupdbs global variables 68 ⟩ Used in section 93.
⟨ setupdbs **extern** declarations for global variables 69 ⟩ Used in section 94.
⟨ setupdbs *process_command_line_options* declaration 70 ⟩ Used in section 94.
⟨ setupdbs *process_command_line_options* definition 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90 ⟩ Used in section 93.
⟨ *stpcdcli.h* 25 ⟩
⟨ *stpcdhdl.h* 33 ⟩
⟨ *stpcdsif.h* 66 ⟩
⟨ *stpclopt.h* 94 ⟩
⟨ *stpcrdbs.h* 111 ⟩
⟨ *create_databases* declaration 96 ⟩ Used in section 111.
⟨ *create_databases* definition 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109 ⟩ Used in section 110.
⟨ *create_tables_gwirdcli* declaration 19 ⟩ Used in section 25.
⟨ *create_tables_gwirdcli* definition 20, 21, 22, 23 ⟩ Used in section 24.
⟨ *create_tables_gwirdsif* declaration 35 ⟩ Used in section 66.
⟨ *create_tables_gwirdsif* definition 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63 ⟩ Used in section 65.
⟨ *create_tables_handles* declaration 27 ⟩ Used in section 33.
⟨ *create_tables_handles* definition 28, 29, 30, 31 ⟩ Used in section 32.
⟨ *delete_and_clear* declaration 91 ⟩ Used in section 94.
⟨ *delete_and_clear* definition 92 ⟩ Used in section 93.
⟨ *finish* definition 4 ⟩ Used in section 17.

GWDG Archive Interface V: Version 1.0

by Laurence D. Finston

September 2013

	Section	Page
GWDG iRODS/Handle Interface Client-Server Application	1	1
Set up databases	2	1
Index	112	60

1. Generate Passwords/Passphrases. Additionally, a checksum can be generated for each password or passphrase, using one of several checksum utilities. See below for more information. [LDF 2012.03.07.]

2. optpsgen. This program generates passwords using random data. [LDF 2013.03.27.]

3. Include header files.

```
<Include header files 3> ≡
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#if 0
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <unistd.h>
#endif
#include <getopt.h>
#include <limits.h>
#include <errno.h>
#include <string>
#include <iomanip>
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <algorithm>
using namespace std;
#include <optpsgsb.h++>
```

See also section 16.

This code is used in sections 15 and 47.

4. Main function. [LDF 2012.03.07.]

```
<Main 4> ≡
int main(int argc, char *argv[]){ bool DEBUG = false; /* true */
    int status = 0;
    if (DEBUG) cerr << "Entering 'optpsgen'." << endl;
    passwd_len = DEFAULT_PASSWD_LEN;
    status = handle_options(argc, argv);
    if (DEBUG) cerr << "handle_options returned " << status << "." << endl;
    if (status < 0) {
        cerr << "ERROR! In 'main': 'handle_options' failed, returning " << status << "."
        endl << "Exiting 'optpsgen' unsuccessfully with exit status 1." << endl;
        exit(1);
    }
    if (debug_level > 0) DEBUG = true;
```

See also sections 5, 6, 7, 8, 9, 10, 11, 12, 13, and 14.

This code is used in section 15.

5. Sort *exclude_char_vector* and erase duplicates. [LDF 2012.03.07.]

```
(Main 4) +≡
if (DEBUG) {
    cerr ≪ "Showing 'exclude_char_vector' before sorting:" ≪ endl;
    for (vector<char>::const_iterator iter = exclude_char_vector.begin(); iter ≠ exclude_char_vector.end();
        ++iter) cerr ≪ "*iter == " ≪ *iter ≪ endl;
} /* if (DEBUG) */
sort(exclude_char_vector.begin(), exclude_char_vector.end());
if (DEBUG) {
    cerr ≪ "Showing 'exclude_char_vector' after sorting:" ≪ endl;
    for (vector<char>::const_iterator iter = exclude_char_vector.begin(); iter ≠ exclude_char_vector.end();
        ++iter) cerr ≪ "*iter == " ≪ *iter ≪ endl;
} /* if (DEBUG) */
vector<char>::iterator exclude_iter = unique(exclude_char_vector.begin(), exclude_char_vector.end());
exclude_char_vector.erase(exclude_iter, exclude_char_vector.end());
if (DEBUG) {
    cerr ≪ "Showing 'exclude_char_vector' after erasing duplicates:" ≪ endl;
    for (vector<char>::const_iterator iter = exclude_char_vector.begin(); iter ≠ exclude_char_vector.end();
        ++iter) cerr ≪ "*iter == " ≪ *iter ≪ endl;
} /* if (DEBUG) */
```

6. Show *checksum_type*. [LDF 2012.03.07.]

```
(Main 4) +≡
if (DEBUG) {
    cerr ≪ "'checksum_type' == ";
    if (checksum_type ≡ 0) cerr ≪ "0" ≪ endl;
    else if (checksum_type ≡ SHA1_TYPE) cerr ≪ "SHA1_TYPE" ≪ endl;
    else if (checksum_type ≡ MD5_TYPE) cerr ≪ "MD5_TYPE" ≪ endl;
    else if (checksum_type ≡ SHA1_TYPE) cerr ≪ "SHA1_TYPE" ≪ endl;
    else if (checksum_type ≡ SHA224_TYPE) cerr ≪ "SHA224_TYPE" ≪ endl;
    else if (checksum_type ≡ SHA256_TYPE) cerr ≪ "SHA256_TYPE" ≪ endl;
    else if (checksum_type ≡ SHA384_TYPE) cerr ≪ "SHA384_TYPE" ≪ endl;
    else if (checksum_type ≡ SHA512_TYPE) cerr ≪ "SHA512_TYPE" ≪ endl;
} /* if (DEBUG) */
```

7. Set type of characters to include in password or passphrase. [LDF 2012.03.07.]

```
(Main 4) +≡
int char_type = status & PRINTABLE_TYPE;
if (char_type ≡ 0) {
    char_type = status & GRAPH_TYPE;
    if (char_type > 0 ∧ status & SPACE_TYPE) {
        if (DEBUG) {
            cerr ≪ "GRAPH_TYPE_and_SPACE_TYPE_set. Setting'char_type'" ≪
                "to'PRINTABLE_TYPE'." ≪ endl;
        } /* if (DEBUG) */
        char_type = PRINTABLE_TYPE;
    }
}
if (char_type ≡ 0) {
    char_type = status & ALPHANUM_TYPE;
}
if (char_type ≡ 0) {
    char_type = status & ALPHA_TYPE;
}
if (char_type ≡ 0) char_type = ALPHANUM_TYPE;
char_type |= status & NO_START_SPACE_TYPE;
/* These apply to PRINTABLE_TYPE and GRAPH_TYPE, too. [LDF 2012.03.07.] */
char_type |= status & NO_END_SPACE_TYPE;
if (char_type & ALPHANUM_TYPE ∨ char_type & ALPHA_TYPE) {
    char_type |= status & SPACE_TYPE;
    char_type |= status & BLANK_TYPE;
    char_type |= status & NO_TABS_TYPE;
}
if (DEBUG) {
    cerr ≪ "char_type== " ≪ char_type ≪ endl ≪ "char_type&ALPHA_TYPE==" ≪
        (char_type & ALPHA_TYPE) ≪ endl ≪ "char_type&ALPHANUM_TYPE==" ≪
        (char_type & ALPHANUM_TYPE) ≪ endl ≪ "char_type&SPACE_TYPE==" ≪
        (char_type & SPACE_TYPE) ≪ endl ≪ "char_type&BLANK_TYPE==" ≪
        (char_type & BLANK_TYPE) ≪ endl ≪ "char_type&NO_TABS_TYPE==" ≪
        (char_type & NO_TABS_TYPE) ≪ endl ≪ "char_type&NO_START_SPACE_TYPE==" ≪
        (char_type & NO_START_SPACE_TYPE) ≪ endl ≪ "char_type&NO_END_SPACE_TYPE==" ≪
        (char_type & NO_END_SPACE_TYPE) ≪ endl ≪ "delim_start==" ≪ delim_start ≪ endl ≪
        "delim_end==" ≪ delim_end ≪ endl ≪ "'min_block_size'" ≪ min_block_size ≪ endl ≪
        "'max_block_size'" ≪ max_block_size ≪ endl ≪ "'iterations'" ≪ iterations ≪ endl;
}
```

8. Open input file. Default is `/dev/urandom`. If the `--extra-random` option is used, `/dev/random` is used instead. This can cause the program to block, if not enough entropy is available in the system. A message is issued to this effect.

Another file to be read from may be specified using the `--input-filename` option. This may be useful for debugging and testing. !! PLEASE NOTE: If this file doesn't contain enough characters, the program will block. That is, there is no provision for closing and reopening it to read it multiple times. [LDF 2012.03.08.]

```
(Main 4) +≡
    ifstream in_file;
    if (in_filename.empty()) {
        if (extra_random) in_filename = "/dev/random";
        else in_filename = "/dev/urandom";
    }
    else if (extra_random) {
        cerr << "WARNING! If an input file is specified using '--input-filename', " <<
            "'--extra-random' has no effect." << endl << "Continuing." << endl;
    }
    in_file.open(in_filename.c_str());
    if (!in_file) {
        cerr << "ERROR! In 'main': Failed to open " << in_filename << "."
            "Exiting 'optpsgen' unsuccessfully with exit status 1." << endl;
        exit(1);
    }
    else if (DEBUG) {
        cerr << "In 'main': Opened " << in_filename << "' successfully." << endl;
    } /* else if (DEBUG) */
```

9. Main loop. [LDF 2012.03.07.]

```
(Main 4) +≡
    char c;
    char prev_char = 'A';
    string passwd;
    stringstream system strm; for (int j = 0; j < iterations; ++j) { char prev_char = 'A';
    passwd = "";
    block_ctr = 0;
```

10. Inner loop. [LDF 2012.03.08.]

```

⟨ Main 4 ⟩ +≡
for (int i = 0; i < passwd_len; ) {
    if (DEBUG) {
        cerr << "Beginning_loop." << endl << "block_ctr'===" << block_ctr << endl << "i==" << i <<
        endl << "passwd_len- max(min_block_size,1)==" << (passwd_len - max(min_block_size,
            1)) << endl;
    } /* if (DEBUG) */
    if (max_block_size > 0 & block_ctr == max_block_size & i < passwd_len - max(min_block_size, 1)) {
        if (DEBUG) {
            cerr << "max_block_size>0&&block_ctr==max_block_size" <<
                "i<passwd_len-(max(min_block_size,1)+1)" << endl <<
                "block_ctr'==" << block_ctr << endl << "max_block_size'==" <<
                max_block_size << endl << "'min_block_size'" << min_block_size <<
                endl << "Adding_a_space_to_passwd", resetting_block_ctr' to 0" <<
                "and continuing." << endl;
        } /* if (DEBUG) */
        passwd += ' ';
        prev_char = ' ';
        ++i;
        block_ctr = 0;
        continue;
    }
    c = in_file.get();
    exclude_iter = find(exclude_char_vector.begin(), exclude_char_vector.end(), c);
    if (exclude_iter != exclude_char_vector.end()) {
        if (DEBUG) {
            cerr << "Character " << c << "' on exclude list. Not adding to passwd." << endl <<
                "Continuing." << endl;
        } /* if (DEBUG) */
        continue;
    }
    else if (DEBUG) {
        cerr << "Character " << c << "' not on exclude list." << endl;
    } /* else if (DEBUG) */
#endif 0
    if (DEBUG) {
        cerr << "c(" << static_cast<int>(c) << ")" << i << endl;
        cerr << "i==" << i << endl;
        cerr << "passwd_len==" << passwd_len << endl;
        cerr << "isspace(c)==" << isspace(c) << endl;
    } /* if (DEBUG) */
#endif
    if (isspace(c)) {
        if (char_type & NO_START_SPACE_TYPE & i == 0) {
            if (DEBUG) {
                cerr << "Space and first character. Not adding to password." << endl;
            } /* if (DEBUG) */
            continue;
        }
        else if (char_type & NO_END_SPACE_TYPE & i == passwd_len - 1) {
            if (DEBUG) {

```

```

        cerr << "Space_and_last_character._Not_adding_to_password." << endl;
    } /* if (DEBUG) */
    continue;
}
else if (min_block_size > 0 & !isspace(prev_char) & block_ctr < min_block_size) {
    if (DEBUG) {
        cerr << "Space_and_`min_block_size'>0_and_`block_ctr'<`min_block_size'." <<
            endl << "Not_adding_space_to_password." << endl << "'block_ctr'==`" <<
            block_ctr << endl << "'min_block_size'==" << min_block_size << endl;
    } /* if (DEBUG) */
    continue;
}
/* if (isspace(c)) */
if (char_type & PRINTABLE_TYPE & isprint(c)) {
    if (DEBUG) {
        cerr << "c==" << c << "'(" << static_cast<int>(c) << ")"<isprintable." << endl <<
            "Adding_to_password." << endl;
    } /* if (DEBUG) */
    passwd += c;
    prev_char = c;
    ++i;
    if (isspace(c)) block_ctr = 0;
    else ++block_ctr;
}
/* if (char_type & PRINTABLE_TYPE & isprint(c)) */
else if (char_type & GRAPH_TYPE & isgraph(c)) {
    if (DEBUG) {
        cerr << "c==" << c << "'(" << static_cast<int>(c) << ")"<isgraphical." <<
            "Adding_to_password." << endl;
    } /* if (DEBUG) */
    passwd += c;
    prev_char = c;
    ++i;
}
/* else if (char_type & GRAPH_TYPE & isgraph(c)) */
else if (char_type & ALPHANUM_TYPE & char_type & ALPHA_TYPE) {
    if (char_type & ALPHA_TYPE & isalpha(c)) {
        if (DEBUG) {
            cerr << "c==" << c << "'(" << static_cast<int>(c) << ")"<isalphabetical." <<
                "Adding_to_password." << endl;
        } /* if (DEBUG) */
        passwd += c;
        prev_char = c;
        ++i;
        ++block_ctr;
        continue;
    } /* if (char_type & ALPHA_TYPE & isalpha(c)) */
    else if (char_type & ALPHANUM_TYPE & isalnum(c)) {
        if (DEBUG) {
            cerr << "c==" << c << "'(" << static_cast<int>(c) << ")"<isalphanumeric." <<
                "Adding_to_password." << endl;
        } /* if (DEBUG) */
        passwd += c;
        prev_char = c;
    }
}
```

```

++i;
++block_ctr;
continue;
} /* else if (char_type & ALPHANUM_TYPE & isalnum(c)) */
else if (isblank(c)) {
    if (char_type & BLANK_TYPE) {
        if (char_type & NO_TABS_TYPE & c == '\t') {
            if (DEBUG) {
                cerr << "c==" << c << "'(" << static_cast<int>(c) << ")is a tab." <<
                    "Not allowing tabs. Not adding to password." << endl;
            } /* if (DEBUG) */
            continue;
        }
        else {
            if (DEBUG) {
                cerr << "c==" << c << "'(" << static_cast<int>(c) << ")" << "is a blank." <<
                    "Allowing blanks. Adding to password." << endl;
            } /* if (DEBUG) */
            passwd += c;
            prev_char = c;
            ++i;
            block_ctr = 0;
            continue;
        }
        /* else */
    } /* if (char_type & BLANK_TYPE) */
else {
    if (DEBUG) {
        cerr << "c==" << c << "'(" << static_cast<int>(c) << ")" << "is blank." <<
            "Not allowing blanks." << "Not adding to password." << endl;
    } /* if (DEBUG) */
    continue;
} /* else if (isblank(c)) */
else if (isspace(c)) {
    if (char_type & SPACE_TYPE) {
        if (DEBUG) {
            cerr << "c==" << c << "'(" << static_cast<int>(c) << ")" << "is space." <<
                "Allowing spaces. Adding to password." << endl;
        } /* if (DEBUG) */
        passwd += c;
        prev_char = c;
        ++i;
        block_ctr = 0;
        continue;
    } /* if (char_type & SPACE_TYPE) */
else {
    if (DEBUG) {
        cerr << "c==" << c << "'(" << static_cast<int>(c) << ")" << "is space." <<
            "Not allowing spaces." << "Not adding to password." << endl;
    } /* if (DEBUG) */
    continue;
} /* else */
}

```

```

    } /* else if (isspace(c)) */
} /* else if (char_type & ALPHANUM_TYPE ∨ char_type & ALPHA_TYPE) */
else {
    if (DEBUG) {
        cerr << "c==" << c << "' " << static_cast<int>(c) << ") is not right type. " <<
            "Not adding to password." << endl;
    } /* if (DEBUG) */
    continue;
}
/* Inner for */

```

11. Write password to standard output. [LDF 2012.03.08.]

```

>Main 4> +≡
if (DEBUG) {
    cerr << endl << "passwd (between '---' on each side) == " << endl << "---" << passwd <<
        "---" << endl;
} /* if (DEBUG) */
cout << delim_start << passwd << delim_end << endl;

```

12. Output checksum (optional). [LDF 2012.03.08.]

```

⟨ Main 4 ⟩ +≡
if (DEBUG ∧ checksum_type ≡ 0) {
    cerr ≪ "checksum_type==0. Not creating checksum." ≪ endl;
}
system_strm.str("");
if (checksum_type ≡ MD5_TYPE) {
    system_strm ≪ "echo-n\\"md5 checksum:\\"; echo' " ≪ passwd ≪
        "\u\|_md5sum\|_tr\--squeeze-repeats\\"\\\"|\u\|_cut\_-f1\_-d\\"\\\"";
}
else if (checksum_type ≡ SHA1_TYPE) {
    system_strm ≪ "echo-n\\"sha1 checksum:\\"; echo' " ≪ passwd ≪
        "\u\|_sha1sum\|_tr\--squeeze-repeats\\"\\\"|\u\|_cut\_-f1\_-d\\"\\\"";
}
else if (checksum_type ≡ SHA224_TYPE) {
    system_strm ≪ "echo-n\\"sha224 checksum:\\"; echo' " ≪ passwd ≪
        "\u\|_sha224sum\|_tr\--squeeze-repeats\\"\\\"|\u\|_cut\_-f1\_-d\\"\\\"";
}
else if (checksum_type ≡ SHA256_TYPE) {
    system_strm ≪ "echo-n\\"sha256 checksum:\\"; echo' " ≪ passwd ≪
        "\u\|_sha256sum\|_tr\--squeeze-repeats\\"\\\"|\u\|_cut\_-f1\_-d\\"\\\"";
}
else if (checksum_type ≡ SHA384_TYPE) {
    system_strm ≪ "echo-n\\"sha384 checksum:\\"; echo' " ≪ passwd ≪
        "\u\|_sha384sum\|_tr\--squeeze-repeats\\"\\\"|\u\|_cut\_-f1\_-d\\"\\\"";
}
else if (checksum_type ≡ SHA512_TYPE) {
    system_strm ≪ "echo-n\\"sha512 checksum:\\"; echo' " ≪ passwd ≪
        "\u\|_sha512sum\|_tr\--squeeze-repeats\\"\\\"|\u\|_cut\_-f1\_-d\\"\\\"";
}
if (DEBUG) {
    cerr ≪ "system_strm.str() == " ≪ system_strm.str() ≪ endl;
} /* if (DEBUG) */

```

13.

```

⟨ Main 4 ⟩ +≡
    status = system(system_strm.str().c_str());
    if (status != 0) {
        cerr << "ERROR! In 'main': 'system' failed, returning " << status << ":" << endl;
        perror("system error");
        if (WIFEXITED(status)) {
            cerr << "WEXITSTATUS(" << status << ") == " << WEXITSTATUS(status) << endl;
        }
        else {
            cerr << "Process didn't exit." << endl;
        }
    } /* if (status != 0) */
    else if (DEBUG) {
        cerr << "'system' succeeded." << endl;
    } /* else if (DEBUG) */
#ifndef 0
    if (j < iterations - 1) cerr << endl;
#endif
} /* Outer for */

```

14. Close input file and exit successfully with exit status 0. [LDF 2012.03.08.]

```

⟨ Main 4 ⟩ +≡
    in_file.close();
    if (DEBUG) {
        cerr << "Exiting 'optpsgen' successfully with exit status 0." << endl;
    } /* if (DEBUG) */
    exit(0); } /* End of main definition */

```

15. Putting `optpsgen` together. [LDF 2012.03.07.]

```

⟨ Include header files 3 ⟩
⟨ Main 4 ⟩

```

16. Include header files.

```
<Include header files 3> +≡
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#if 0
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <unistd.h>
#endif
#include <getopt.h>
#include <limits.h>
#include <errno.h>
#include <string>
#include <iomanip>
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <algorithm>
using namespace std;
```

17. Global variables. [LDF 2012.03.07.]

```
{ Global variables 17 } ≡
extern const unsigned int DEFAULT_PASSWD_LEN = 8;
unsigned int passwd_len = 0;
/* Set to DEFAULT_PASSWD_LEN at the beginning of main. [LDF 2012.03.08.] */
int debug_level = 0;
extern const unsigned int ALPHA_TYPE = 1;
extern const unsigned int ALPHANUM_TYPE = 2;
extern const unsigned int GRAPH_TYPE = 4;
extern const unsigned int PRINTABLE_TYPE = 8;
extern const unsigned int BLANK_TYPE = 16;
extern const unsigned int SPACE_TYPE = 32;
extern const unsigned int NO_TABS_TYPE = 64;
extern const unsigned int NO_START_SPACE_TYPE = 128;
extern const unsigned int NO_END_SPACE_TYPE = 256;
extern const unsigned int MD5_TYPE = 1;
extern const unsigned int SHA1_TYPE = 2;
extern const unsigned int SHA224_TYPE = 3;
extern const unsigned int SHA256_TYPE = 4;
extern const unsigned int SHA384_TYPE = 5;
extern const unsigned int SHA512_TYPE = 6;
extern const int DEFAULT_MIN_BLOCK_SIZE = 4;
extern const int DEFAULT_MAX_BLOCK_SIZE = 8;
int min_block_size = 0;
int max_block_size = 0;
unsigned int block_ctr = 0;
bool extra_random = false;
string delim_start;
string delim_end;
string in_filename;
vector<char> exclude_char_vector;
unsigned int checksum_type = 0;
unsigned int iterations = 1;
```

This code is used in section 47.

18.

```
(extern declarations for global variables 18) ≡
  extern const unsigned int DEFAULT_PASSWD_LEN;
  extern unsigned int passwd_len;
  extern int debug_level;
  extern const unsigned int ALPHA_TYPE;
  extern const unsigned int ALPHANUM_TYPE;
  extern const unsigned int GRAPH_TYPE;
  extern const unsigned int PRINTABLE_TYPE;
  extern const unsigned int BLANK_TYPE;
  extern const unsigned int SPACE_TYPE;
  extern const unsigned int NO_TABS_TYPE;
  extern const unsigned int NO_START_SPACE_TYPE;
  extern const unsigned int NO_END_SPACE_TYPE;
  extern const unsigned int MD5_TYPE;
  extern const unsigned int SHA1_TYPE;
  extern const unsigned int SHA224_TYPE;
  extern const unsigned int SHA256_TYPE;
  extern const unsigned int SHA384_TYPE;
  extern const unsigned int SHA512_TYPE;
  extern const int DEFAULT_MIN_BLOCK_SIZE;
  extern const int DEFAULT_MAX_BLOCK_SIZE;
  extern int min_block_size;
  extern int max_block_size;
  extern unsigned int block_ctr;
  extern bool extra_random;
  extern string delim_start;
  extern string delim_end;
  extern string in_filename;
  extern vector<char> exclude_char_vector;
  extern unsigned int checksum_type;
  extern unsigned int iterations;
```

This code is used in section 48.

19. Handle command-line options. [LDF 2012.03.07.]

```
(handle_options declaration 19) ≡
  int handle_options(int argc, char *argv[]);
```

This code is used in sections 47 and 48.

20.

```

⟨ handle_options definition 20 ⟩ ≡
int handle_options(int argc, char *argv[]){ bool DEBUG = false;      /* true */
    const unsigned int ALPHA_INDEX = 0;
    const unsigned int ALPHANUM_INDEX = 1;
    const unsigned int ALNUM_INDEX = 2;
    const unsigned int GRAPH_INDEX = 3;
    const unsigned int PRINTABLE_INDEX = 4;
    const unsigned int BLANK_INDEX = 5;
    const unsigned int SPACE_INDEX = 6;
    const unsigned int NO_TABS_INDEX = 7;
    const unsigned int LENGTH_INDEX = 8;
    const unsigned int HELP_INDEX = 9;
    const unsigned int EXTRA_RANDOM_INDEX = 10;
    const unsigned int MIN_BLOCK_SIZE_INDEX = 11;
    const unsigned int MAX_BLOCK_SIZE_INDEX = 12;
    const unsigned int DEBUG_LEVEL_INDEX = 13;
    const unsigned int NO_START_SPACE_INDEX = 14;
    const unsigned int NO_END_SPACE_INDEX = 15;
    const unsigned int DELIMITER_INDEX = 16;
    const unsigned int INPUT_FILENAME_INDEX = 17;
    const unsigned int EXCLUDE_CHARS_INDEX = 18;
    const unsigned int CHECKSUM_INDEX = 19;
    const unsigned int ITERATIONS_INDEX = 20;
    int return_val = 0;
    static struct option long_options[] = {{ "alpha", 0, 0, 0}, {"alphanum", 0, 0, 0}, {"alnum", 0, 0, 0},
                                           {"graph", 0, 0, 0}, {"printable", 0, 0, 0}, {"blank", 0, 0, 0}, {"space", 0, 0, 0}, {"no-tabs", 0, 0, 0},
                                           {"length", 1, 0, 0}, {"help", 0, 0, 0}, {"extra-random", 0, 0, 0}, {"min-block-size", 2, 0, 0},
                                           {"max-block-size", 2, 0, 0}, {"debug-level", 2, 0, 0}, {"no-start-space", 0, 0, 0},
                                           {"no-end-space", 0, 0, 0}, {"delimiters", 2, 0, 0}, {"input-filename", 1, 0, 0},
                                           {"exclude-chars", 1, 0, 0}, {"checksum", 2, 0, 0}, {"iterations", 1, 0, 0}, {0, 0, 0, 0}};
    int option_ctr = 0;
    int option_index = 0;
    int this_option_optind = optind ? optind : 1; while (1) { option_ctr = getopt_long_only(argc, argv,
                                              "", long_options, &option_index);
    if (DEBUG) {
        cerr << "option_ctrl=" << option_ctr << endl << "option_index=" << option_index << endl;
        if (optarg) {
            cerr << "optarg=" << optarg << endl;
        } /* if (optarg) */
    } /* if (DEBUG) */
    if (option_ctr == -1) {
        if (DEBUG) {
            cerr << "No more option arguments." << endl;
        }
        break;
    }
}

```

See also sections 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, and 46.

This code is used in section 47.

21. Option. [LDF 2012.03.07.]

```
<handle_options definition 20> +≡
  else if (option_ctr == 0) {
    if (DEBUG) {
      cerr << "option_" << long_options[option_index].name;
      if (optarg) {
        cerr << "with_arg_" << optarg;
      }
      cerr << endl;
    }
  }
```

22. alpha: Alphabetical characters only. [LDF 2012.03.07.]

```
<handle_options definition 20> +≡
  if (option_index == ALPHA_INDEX) {
    if (DEBUG) {
      cerr << "'option_index'" << "==" << 'ALPHA_INDEX' << endl;
    }
    return_val |= ALPHA_TYPE;
  } /* if (option_index == ALPHA_INDEX) */
```

23. alphanum: Alphanumeric characters only.

```
<handle_options definition 20> +≡
  else
    if (option_index == ALPHANUM_INDEX ∨ option_index == ALNUM_INDEX) {
      if (DEBUG) {
        cerr << "'option_index'" == 'ALPHANUM_INDEX' || "'ALNUM_INDEX'" << endl;
      }
      return_val |= ALPHANUM_TYPE;
    } /* else if (option_index == ALPHANUM_INDEX ∨ option_index == ALNUM_INDEX) */
```

24. graph: Graphical characters only (excluding space).

```
<handle_options definition 20> +≡
  else
    if (option_index == GRAPH_INDEX) {
      if (DEBUG) {
        cerr << "'option_index'" == 'GRAPH_INDEX' << endl;
      }
      return_val |= GRAPH_TYPE;
    } /* else if (option_index == GRAPH_INDEX) */
```

25. printable: Printable characters (including space).

```
<handle_options definition 20> +≡
  else
    if (option_index == PRINTABLE_INDEX) {
      if (DEBUG) {
        cerr << "'option_index'" == 'PRINTABLE_INDEX' << endl;
      }
      return_val |= PRINTABLE_TYPE;
    } /* else if (option_index == PRINTABLE_INDEX) */
```

26. blank: Blank characters (space or tab).

```
<handle_options definition 20> +≡
else
  if (option_index ≡ BLANK_INDEX) {
    if (DEBUG) {
      cerr ≡ "‘option_index’ == ‘BLANK_INDEX’" ≡ endl;
    }
    return_val |= BLANK_TYPE;
  } /* else if (option_index ≡ BLANK_INDEX) */
```

27. space: Allow spaces. This is meant for use with `alpha` or `alphanum`. It's not needed for `graph` or `print`, because `print` is equivalent to `graph` plus `space`. However, the latter works, too. [LDF 2012.03.07.]

```
<handle_options definition 20> +≡
else
  if (option_index ≡ SPACE_INDEX) {
    if (DEBUG) {
      cerr ≡ "‘option_index’ == ‘SPACE_INDEX’" ≡ endl;
    }
    return_val |= SPACE_TYPE;
    return_val |= BLANK_TYPE;
    /* Allowing space characters implies allowing blank characters. [LDF 2012.03.07.] */
  } /* else if (option_index ≡ SPACE_INDEX) */
```

28. --no-tabs: Don't allow tab characters. This is meant for use with `--blank`. Otherwise, it has no effect. `--blank`, in turn, only has an effect with `--alpha` and `--alphanum`. [LDF 2012.03.07.]

```
<handle_options definition 20> +≡
else
  if (option_index ≡ NO_TABS_INDEX) {
    if (DEBUG) {
      cerr ≡ "‘option_index’ == ‘NO_TABS_INDEX’" ≡ endl;
    }
    return_val |= NO_TABS_TYPE;
  } /* else if (option_index ≡ NO_TABS_INDEX) */
```

29. --length.

```
( handle_options definition 20 ) +≡
else
  if (option_index ≡ LENGTH_INDEX) {
    if (DEBUG) {
      cerr ≡ "‘option_index’ == ‘LENGTH_INDEX’" ≡ endl;
    }
    errno = 0;
    passwd_len = strtoul(optarg, 0, 0);
    if (passwd_len ≡ ULONG_MAX) {
      cerr ≡ "ERROR! In ‘handle_options’: ‘strtoul’ failed, "
            ≡ "returning ‘ULONG_MAX’: " ≡ endl;
      perror("strtoul_error:");
      cerr ≡ "Setting ‘passwd_len’ to " ≡ DEFAULT_PASSWD_LEN ≡ ". Continuing." ≡ endl;
      passwd_len = DEFAULT_PASSWD_LEN;
    } /* if (passwd_len ≡ ULONG_MAX) */
    if (DEBUG) cerr ≡ "passwd_len == " ≡ passwd_len ≡ endl;
  } /* else if (option_index ≡ LENGTH_INDEX) */
```

30. --help.

```
( handle_options definition 20 ) +≡
else
  if (option_index ≡ HELP_INDEX) {
    if (DEBUG) {
      cerr ≡ "option_index" == "HELP_INDEX" ≡ endl;
    }
    cout ≡ "gnpsswdp" ≡ endl ≡ "Copyright (C) 2012 Gesellschaft fuer"
      ≡ "wissenschaftliche Datenverarbeitung mbH Goettingen" ≡ endl ≡
      "Author: Laurence D. Finston" ≡ "Generate password or passphrase" ≡
      endl ≡ "Usage: optpsgen [OPTIONS]..." ≡ endl ≡ "Options:" ≡ endl ≡
      "--help: Print this message and exit" ≡ endl ≡
      "--alpha: Alphabetical characters only (isalpha)" ≡ endl ≡
      "--alphanum: Alphanumeric characters only (isalnum)" ≡
      endl ≡ "This is the default" ≡ endl ≡
      "--alnum: Alphanumeric characters only (synonym for)" ≡
      "--alphanum)" ≡ "(isalnum)" ≡ endl ≡
      "--graph: Graphical characters, i.e.," ≡ endl ≡
      "printable characters excluding space characters (isgraph)" ≡ endl ≡
      "--printable: Printable characters including space characters" ≡
      "(isprint)" ≡ endl ≡ "--blank: Allow blank"
      ≡ "(i.e., tab and space characters)" ≡ "(isblank)" ≡ endl ≡
      "For use with --alpha and --alphanum" ≡ endl ≡
      "--space: Allow space" ≡ endl ≡
      "For use with --alpha and --alphanum" ≡ endl ≡
      "--no-tabs: Do not use tab characters" ≡ endl ≡
      "For use with --blank" ≡ endl ≡ "--length[ARG]:"
      ≡ "Set length of password or passphrase to ARG" ≡ endl ≡
      "--extra-random: Read from /dev/random instead of /dev/urandom" ≡
      endl ≡ "This provides better quality random bytes, but"
      ≡ "may block if not enough entropy is present" ≡ endl ≡
      "in the system." ≡ endl ≡ "--min-block-size[=ARG]" ≡
      endl ≡ "--max-block-size[=ARG]: Minimum or maximum size of blocks,"
      ≡ "respectively, without intervening space" ≡ endl ≡
      "For use with --print or --alpha or --alnum with" ≡
      "--blank or --space" ≡ endl ≡ "If no argument is provided, the minimum"
      ≡ "block size is set to" ≡ DEFAULT_MIN_BLOCK_SIZE ≡ "."
      ≡ "and the maximum to" ≡ DEFAULT_MAX_BLOCK_SIZE ≡ "."
      ≡ endl ≡ "--debug-level[ARG]: Set debug level. If ARG not supplied, "
      ≡ "debug level is set to 1." ≡ "--no-start-space" ≡ endl ≡
      "--no-end-space: Prevents a space character from appearing at"
      ≡ "the beginning or end of the password or passphrase" ≡ endl ≡
      "--delimiters[ARG]: Specify delimiters for the password or"
      ≡ "passphrase, when it's output." ≡ endl ≡
      "--input-filename[ARG]: Name of file to use instead of /dev/urandom"
      ≡ "or /dev/random" ≡ endl ≡ "For testing purposes." ≡
      endl ≡ "--exclude-chars[ARG]: ARG is a list of characters to exclude"
      ≡ endl ≡ "This option may be used multiple times." ≡
      endl ≡ "--checksum[ARG]: Generate checksum." ≡ endl ≡
      "ARG may be md5, sha1, sha224, sha256, sha384" ≡
      "or sha512. If argument is not provided, sha1 will be used." ≡ endl ≡
```

```

    " --iterations INTEGER: Number of passwords or passphrases to generate" <<
    endl;
    exit(0);
} /* else if (option_index == HELP_INDEX) */

```

31. --extra-random.

```

⟨ handle_options definition 20 ⟩ +≡
else
  if (option_index == EXTRA_RANDOM_INDEX) {
    if (DEBUG) {
      cerr << "option_index' == 'EXTRA_RANDOM_INDEX'" << endl;
    }
    cerr << "Will read from /dev/random. This may block, "
    << "if not enough entropy is available in the system." << endl <<
    "If it does, do some work in another shell, move the mouse, access the "
    "file system, etc.," << endl << "in order to create more entropy." << endl;
    extra_random = true;
  } /* else if (option_index == EXTRA_RANDOM_INDEX) */

```

32. --min-block-size.

```

⟨ handle_options definition 20 ⟩ +≡
else
  if (option_index == MIN_BLOCK_SIZE_INDEX) {
    if (DEBUG) {
      cerr << "option_index' == 'MIN_BLOCK_SIZE_INDEX'" << endl;
    }
    if (optarg) {
      errno = 0;
      min_block_size = strtoul(optarg, 0, 0);
      if (min_block_size == LONG_MAX || min_block_size == LONG_MIN) {
        cerr << "ERROR! In 'handle_options': 'strtoul' failed, "
        << "returning";
        if (min_block_size > 0) cerr << "'LONG_MAX':" << endl;
        else cerr << "'LONG_MIN':" << endl;
        perror("strtoul error:");
        cerr << "Setting 'min_block_size' to "
        << DEFAULT_MIN_BLOCK_SIZE <<
        ". Continuing." << endl;
        min_block_size = DEFAULT_MIN_BLOCK_SIZE;
      } /* if (min_block_size == LONG_MAX || min_block_size == LONG_MIN) */
    } /* if (optarg) */
    else min_block_size = DEFAULT_MIN_BLOCK_SIZE;
  } /* else if (option_index == MIN_BLOCK_SIZE_INDEX) */

```

33. --max-block-size.

```
<handle_options definition 20> +≡
else
  if (option_index ≡ MAX_BLOCK_SIZE_INDEX) {
    if (DEBUG) {
      cerr ≡ "option_index" ≡ "MAX_BLOCK_SIZE_INDEX" ≡ endl;
    }
    if (optarg) {
      errno = 0;
      max_block_size = strtoul(optarg, 0, 0);
      if (max_block_size ≡ LONG_MAX ∨ max_block_size ≡ LONG_MIN) {
        cerr ≡ "ERROR! In 'handle_options': 'strtoul' failed, " ≡ "returning";
        if (max_block_size > 0) cerr ≡ "'LONG_MAX':" ≡ endl;
        else cerr ≡ "'LONG_MIN':" ≡ endl;
        perror("strtoul_error:");
        cerr ≡ "Setting 'max_block_size' to " ≡ DEFAULT_MAX_BLOCK_SIZE ≡
          ". Continuing." ≡ endl;
        max_block_size = DEFAULT_MAX_BLOCK_SIZE;
      } /* if (max_block_size ≡ LONG_MAX ∨ max_block_size ≡ LONG_MIN) */
    } /* if (optarg) */
    else max_block_size = DEFAULT_MAX_BLOCK_SIZE;
  } /* else if (option_index ≡ MAX_BLOCK_SIZE_INDEX) */
```

34. --debug-level.

```
<handle_options definition 20> +≡
else
  if (option_index ≡ DEBUG_LEVEL_INDEX) {
    if (DEBUG) {
      cerr ≡ "option_index" ≡ "DEBUG_LEVEL_INDEX" ≡ endl;
    }
    if (optarg) {
      errno = 0;
      debug_level = strtol(optarg, 0, 0);
      if (debug_level ≡ LONG_MIN ∨ debug_level ≡ LONG_MAX) {
        cerr ≡ "ERROR! In 'handle_options': 'strtol' failed, " ≡ "returning";
        if (debug_level > 0) cerr ≡ "'LONG_MAX':" ≡ endl;
        else cerr ≡ "'LONG_MIN':" ≡ endl;
        perror("strtoul_error:");
        cerr ≡ "Setting 'debug_level' to 1. Continuing." ≡ endl;
        debug_level = 1;
      } /* if (debug_level ≡ LONG_MIN ∨ debug_level ≡ LONG_MAX) */
    } /* if (optarg) */
    else debug_level = 1;
    if (DEBUG) {
      cerr ≡ "debug_level" ≡ " " ≡ debug_level ≡ endl;
    } /* if (DEBUG) */
    if (debug_level > 0) DEBUG = true;
  } /* else if (option_index ≡ DEBUG_LEVEL_INDEX) */
```

35. --no-start-space.

```
<handle_options definition 20> +≡
else
  if (option_index ≡ NO_START_SPACE_INDEX) {
    if (DEBUG) {
      cerr ≡ "option_index" ≡ "NO_START_SPACE_INDEX" ≡ endl;
    }
    return_val |= NO_START_SPACE_TYPE;
  } /* else if (option_index ≡ NO_START_SPACE_INDEX) */
```

36. --no-end-space.

```
<handle_options definition 20> +≡
else
  if (option_index ≡ NO_END_SPACE_INDEX) {
    if (DEBUG) {
      cerr ≡ "option_index" ≡ "NO_END_SPACE_INDEX" ≡ endl;
    }
    return_val |= NO_END_SPACE_TYPE;
  } /* else if (option_index ≡ NO_END_SPACE_INDEX) */
```

37. --delimiters.

```
<handle_options definition 20> +≡
else
  if (option_index ≡ DELIMITER_INDEX) {
    if (DEBUG) {
      cerr ≡ "option_index" ≡ "DELIMITER_INDEX" ≡ endl;
    }
    if (optarg ∧ strlen(optarg) > 0) {
      delim_start = optarg[0];
      if (strlen(optarg) > 1) delim_end = optarg[1];
      else delim_end = delim_start;
    }
    else delim_start = delim_end = ",";
  } /* else if (option_index ≡ DELIMITER_INDEX) */
```

38. --input-filename.

```
<handle_options definition 20> +≡
else
  if (option_index ≡ INPUT_FILENAME_INDEX) {
    if (DEBUG) {
      cerr ≡ "option_index" ≡ "INPUT_FILENAME_INDEX" ≡ endl;
    }
    in_filename = optarg;
  } /* else if (option_index ≡ INPUT_FILENAME_INDEX) */
```

39. --exclude-chars.

```
<handle_options definition 20> +≡
else
  if (option_index ≡ EXCLUDE_CHARS_INDEX) {
    if (DEBUG) {
      cerr ≪ "option_index" ≡ "EXCLUDE_CHARS_INDEX" ≪ endl;
    }
    for (int i = 0; i < strlen(optarg); ++i) exclude_char_vector.push_back(optarg[i]);
  } /* else if (option_index ≡ EXCLUDE_CHARS_INDEX) */
```

40. --checksum.

```
<handle_options definition 20> +≡
else
  if (option_index ≡ CHECKSUM_INDEX) {
    if (DEBUG) {
      cerr ≪ "option_index" ≡ "CHECKSUM_INDEX" ≪ endl;
    }
    if (optarg ≡ 0 ∨ strlen(optarg) ≡ 0) checksum_type = SHA1_TYPE;
    else if (!strcasecmp(optarg, "md5")) checksum_type = MD5_TYPE;
    else if (!strcasecmp(optarg, "sha1")) checksum_type = SHA1_TYPE;
    else if (!strcasecmp(optarg, "sha224")) checksum_type = SHA224_TYPE;
    else if (!strcasecmp(optarg, "sha256")) checksum_type = SHA256_TYPE;
    else if (!strcasecmp(optarg, "sha384")) checksum_type = SHA384_TYPE;
    else if (!strcasecmp(optarg, "sha512")) checksum_type = SHA512_TYPE;
    else {
      cerr ≪ "WARNING! In 'handle_options': Invalid argument to '--checksum' ≡
             optarg ≡ endl ≪ "Setting 'checksum_type' to 'SHA1_TYPE'." ≪ endl;
      checksum_type = SHA1_TYPE;
    }
  } /* else if (option_index ≡ CHECKSUM_INDEX) */
```

41. --iterations.

```
<handle_options definition 20> +≡
else
  if (option_index ≡ ITERATIONS_INDEX) {
    if (DEBUG) {
      cerr ≪ "option_index" ≡ "ITERATIONS_INDEX" ≪ endl;
    }
    errno = 0;
    iterations = strtoul(optarg, 0, 0);
    if (iterations ≡ LONG_MAX ∨ iterations ≡ LONG_MIN) {
      cerr ≪ "ERROR! In 'handle_options': 'strtoul' failed, " ≪ "returning ";
      if (iterations > 0) cerr ≪ "'LONG_MAX': " ≪ endl;
      else cerr ≪ "'LONG_MIN': " ≪ endl;
      perror("strtoul error:");
      cerr ≪ "Setting 'iterations' to 1. Continuing." ≪ endl;
      iterations = 1;
    } /* if (iterations ≡ LONG_MAX ∨ iterations ≡ LONG_MIN) */
  } /* else if (option_index ≡ ITERATIONS_INDEX) */
```

42. Invalid option_index value.

```
< handle_options definition 20 > +≡
else {
    cerr ≪ "WARNING! In 'handle_options': " ≪ endl ≪ "'option_index' has invalid value: " ≪
        option_index ≪ endl ≪ "Will try to continue." ≪ endl;
}
/* else if (option_ctr ≡ 0) */
```

43. Ambiguous option. [LDF 2012.03.07.]

```
< handle_options definition 20 > +≡
else
if (option_ctr ≡ '?') {
    cerr ≪ "WARNING! In 'handle_options': " ≪ endl ≪
        "'getopt_long_only' returned ambiguous match. " ≪ "Breaking." ≪ endl;
    break;
} /* else if (option_ctr ≡ '?') */
```

44. Invalid option.

```
< handle_options definition 20 > +≡
else {
    if (DEBUG) {
        cerr ≪ "'getopt_long_only' returned invalid option." ≪ endl;
    }
}
```

45. End of while loop.

```
< handle_options definition 20 > +≡
if (DEBUG) {
    cerr ≪ "End of option processing" ≪ endl;
}
/* while */
if (optind < argc) {
    if (DEBUG) {
        cerr ≪ "Non-option ARGV-elements: ";
        int j = optind;
        for (int i = 0; j < argc; ++i, ++j) {
            cerr ≪ "argv[" ≪ j ≪ "] == " ≪ argv[j] ≪ endl ≪ "strlen(argv[j]) == "
                strlen(argv[j]) ≪ endl;
        }
    } /* for */
} /* if (DEBUG) */
} /* if */
```

46.

```
< handle_options definition 20 > +≡
if (DEBUG) {
    cerr ≪ "Exiting 'handle_options' successfully with return value " ≪ return_val ≪ "."
        endl;
} /* if (DEBUG) */
return return_val; } /* End of handle_options definition */
```

47. Putting `optpsgsb.web` together. [LDF 2012.03.07.]

```
⟨ Include header files 3 ⟩
⟨ Global variables 17 ⟩
⟨ handle_options declaration 19 ⟩
⟨ handle_options definition 20 ⟩
```

48.

```
⟨ optpsgsb.h 48 ⟩ ≡
  ⟨ extern declarations for global variables 18 ⟩
  ⟨ handle_options declaration 19 ⟩
```

```
ALNUM_INDEX: 20, 23.
ALPHA_INDEX: 20, 22.
ALPHA_TYPE: 7, 10, 17, 18, 22.
ALPHANUM_INDEX: 20, 23.
ALPHANUM_TYPE: 7, 10, 17, 18, 23.
arge: 4, 19, 20, 45.
argv: 4, 19, 20, 45.
begin: 5, 10.
BLANK_INDEX: 20, 26.
BLANK_TYPE: 7, 10, 17, 18, 26, 27.
block_ctr: 9, 10, 17, 18.
c: 9.
c_str: 8, 13.
cerr: 4, 5, 6, 7, 8, 10, 11, 12, 13, 14, 20, 21, 22,
     23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35,
     36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46.
char_type: 7, 10.
CHECKSUM_INDEX: 20, 40.
checksum_type: 6, 12, 17, 18, 40.
close: 14.
const_iterator: 5.
cout: 11, 30.
DEBUG: 4, 5, 6, 7, 8, 10, 11, 12, 13, 14, 20, 21, 22,
      23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
      35, 36, 37, 38, 39, 40, 41, 44, 45, 46.
debug_level: 4, 17, 18, 34.
DEBUG_LEVEL_INDEX: 20, 34.
DEFAULT_MAX_BLOCK_SIZE: 17, 18, 30, 33.
DEFAULT_MIN_BLOCK_SIZE: 17, 18, 30, 32.
DEFAULT_PASSWD_LEN: 4, 17, 18, 29.
delim_end: 7, 11, 17, 18, 37.
delim_start: 7, 11, 17, 18, 37.
DELIMITER_INDEX: 20, 37.
empty: 8.
end: 5, 10.
endl: 4, 5, 6, 7, 8, 10, 11, 12, 13, 14, 20, 21, 22,
      23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35,
      36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46.
erase: 5.
errno: 29, 32, 33, 34, 41.
exclude_char_vector: 5, 10, 17, 18, 39.
```

```
EXCLUDE_CHARS_INDEX: 20, 39.
exclude_iter: 5, 10.
exit: 4, 8, 14, 30.
extra_random: 8, 17, 18, 31.
EXTRA_RANDOM_INDEX: 20, 31.
false: 4, 17, 20.
find: 10.
get: 10.
getopt_long_only: 20.
GRAPH_INDEX: 20, 24.
GRAPH_TYPE: 7, 10, 17, 18, 24.
handle_options: 4, 19, 20, 46.
HELP_INDEX: 20, 30.
i: 10, 39, 45.
ifstream: 8.
in_file: 8, 10, 14.
in_filename: 8, 17, 18, 38.
INPUT_FILENAME_INDEX: 20, 38.
isalnum: 10.
isalpha: 10.
isblank: 10.
isgraph: 10.
isprint: 10.
isspace: 10.
iter: 5.
iterations: 7, 9, 13, 17, 18, 41.
ITERATIONS_INDEX: 20, 41.
iterator: 5.
j: 9, 45.
LENGTH_INDEX: 20, 29.
LONG_MAX: 32, 33, 34, 41.
LONG_MIN: 32, 33, 34, 41.
long_options: 20, 21.
main: 4, 14, 17.
max: 10.
max_block_size: 7, 10, 17, 18, 33.
MAX_BLOCK_SIZE_INDEX: 20, 33.
MD5_TYPE: 6, 12, 17, 18, 40.
min_block_size: 7, 10, 17, 18, 32.
MIN_BLOCK_SIZE_INDEX: 20, 32.
name: 21.
```

NO_END_SPACE_INDEX: 20, 36.
NO_END_SPACE_TYPE: 7, 10, 17, 18, 36.
NO_START_SPACE_INDEX: 20, 35.
NO_START_SPACE_TYPE: 7, 10, 17, 18, 35.
NO_TABS_INDEX: 20, 28.
NO_TABS_TYPE: 7, 10, 17, 18, 28.
NOTE: 8.
open: 8.
optarg: 20, 21, 29, 32, 33, 34, 37, 38, 39, 40, 41.
optind: 20, 45.
option: 20.
option_ctr: 20, 21, 42, 43.
option_index: 20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42.
passwd: 9, 10, 11, 12.
passwd_len: 4, 10, 17, 18, 29.
perror: 13, 29, 32, 33, 34, 41.
prev_char: 9, 10.
PRINTABLE_INDEX: 20, 25.
PRINTABLE_TYPE: 7, 10, 17, 18, 25.
push_back: 39.
return_val: 20, 22, 23, 24, 25, 26, 27, 28, 35, 36, 46.
SHA1_TYPE: 6, 12, 17, 18, 40.
SHA224_TYPE: 6, 12, 17, 18, 40.
SHA256_TYPE: 6, 12, 17, 18, 40.
SHA384_TYPE: 6, 12, 17, 18, 40.
SHA512_TYPE: 6, 12, 17, 18, 40.
sort: 5.
SPACE_INDEX: 20, 27.
SPACE_TYPE: 7, 10, 17, 18, 27.
status: 4, 7, 13.
std: 3, 16.
str: 12, 13.
strcasecmp: 40.
string: 9, 17, 18.
stringstream: 9.
strlen: 37, 39, 40, 45.
strtol: 34.
strtoul: 29, 32, 33, 41.
system: 13.
system_strm: 9, 12, 13.
this_option_optind: 20.
true: 4, 20, 31, 34.
ULONG_MAX: 29.
unique: 5.
vector: 5, 17, 18.
WEXITSTATUS: 13.
WIFEXITED: 13.

⟨ Global variables 17 ⟩ Used in section 47.
⟨ Include header files 3, 16 ⟩ Used in sections 15 and 47.
⟨ Main 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 ⟩ Used in section 15.
⟨ extern declarations for global variables 18 ⟩ Used in section 48.
⟨ `optpsgsb.h` 48 ⟩
⟨ *handle_options* declaration 19 ⟩ Used in sections 47 and 48.
⟨ *handle_options* definition 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46 ⟩ Used in section 47.

OptiNum Grid Installer
Version 1.0

Part V: Generate Passwords/Passphrases
by Laurence D. Finston

September 2013

	Section	Page
Generate Passwords/Passphrases	1	1