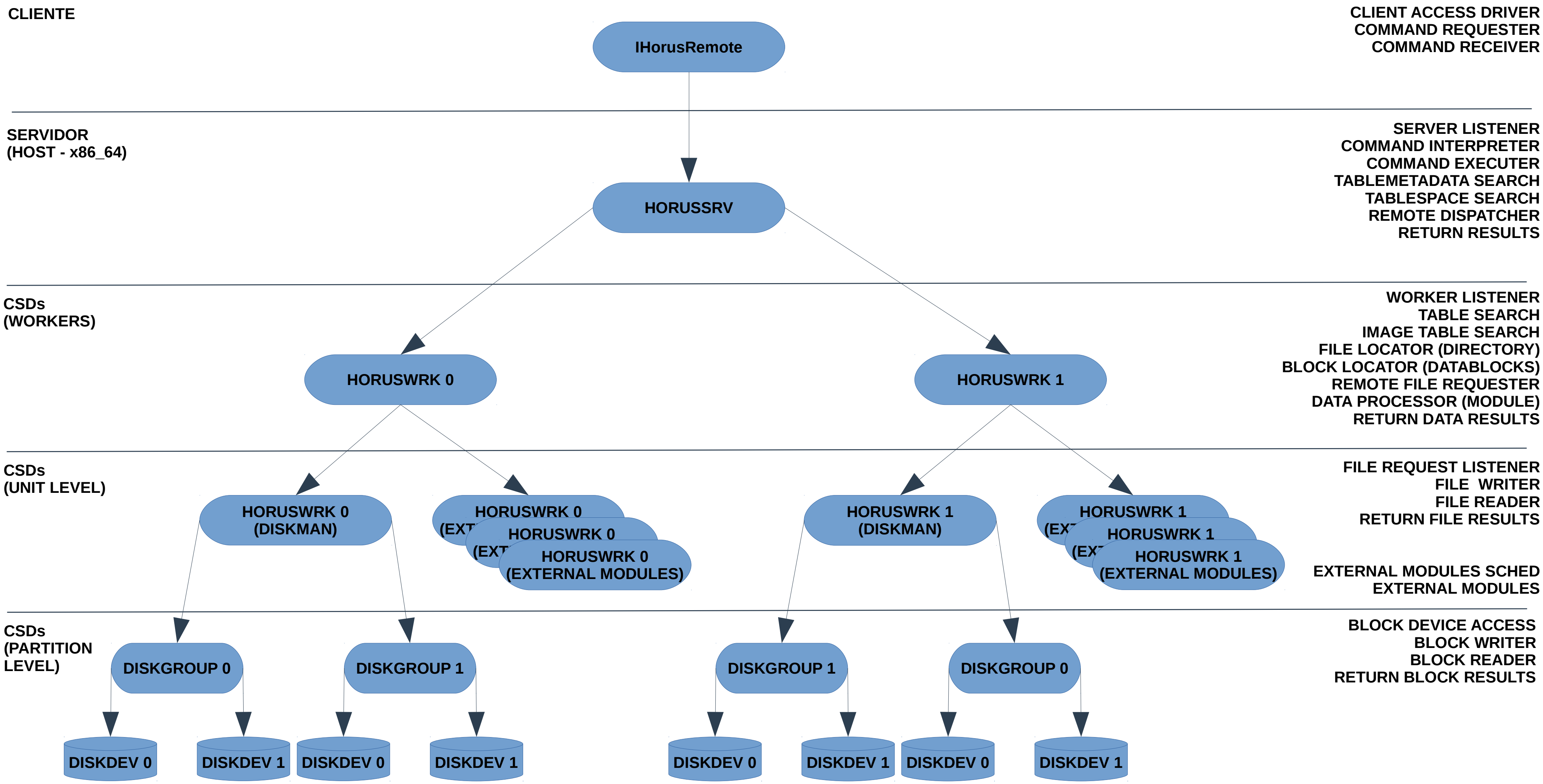
HORUS IMAGE SERVER v2.6
Análise da Hierarquia de Tarefas

Luiz Marcio Faria de Aquino Viana, M.Sc.
E-mail: lmarcio@cos.ufrj.br
lmarcio@tlmv.com.br
luiz.marcio.viana@gmail.com
Phone: +55-21-99983-7207
DRE: 120048833
CPF: 024.723.347-10
RG: 08855128-8 IFP-RJ
Registro: 2000103581 CREA-RJ

Análise dos Pontos de Desacoplamento:



CLIENTE

IHorusRemote

CLIENT ACCESS DRIVER
COMMAND REQUESTER
COMMAND RECEIVER

SERVIDOR
(HOST - x86_64)

HORUSSRV

SERVER LISTENER
COMMAND INTERPRETER
COMMAND EXECUTER
TABLEMETADATA SEARCH
TABLESPACE SEARCH
REMOTE DISPATCHER
RETURN RESULTS

CSDs
(WORKERS)

HORUSWRK 0

HORUSWRK 1

WORKER LISTENER
TABLE SEARCH
IMAGE TABLE SEARCH
FILE LOCATOR (DIRECTORY)
BLOCK LOCATOR (DATABLOCKS)
REMOTE FILE REQUESTER
DATA PROCESSOR (MODULE)
RETURN DATA RESULTS

CSDs
(UNIT LEVEL)

HORUSWRK 0
(DISKMAN)

HORUSWRK 0
(EXT

HORUSWRK 0
(EXT

HORUSWRK 0
(EXTERNAL MODULES)

HORUSWRK 1
(DISKMAN)

HORUSWRK 1
(EX

HORUSWRK 1
(EX

HORUSWRK 1
(EXTERNAL MODULES)

FILE REQUEST LISTENER
FILE WRITER
FILE READER
RETURN FILE RESULTS

EXTERNAL MODULES SCHED
EXTERNAL MODULES

CSDs
(PARTITION
LEVEL)

DISKGROUP 0

DISKGROUP 1

DISKGROUP 1

DISKGROUP 0

BLOCK DEVICE ACCESS
BLOCK WRITER
BLOCK READER
RETURN BLOCK RESULTS

DISKDEV 0

DISKDEV 1

DISKDEV 0

DISKDEV 1

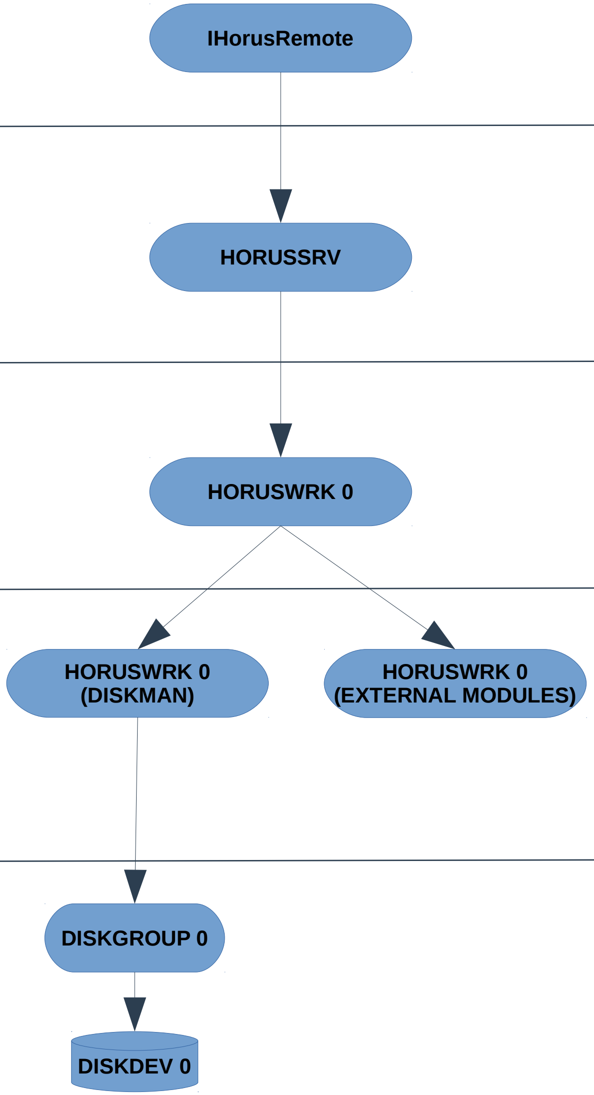
DISKDEV 0

DISKDEV 1

DISKDEV 0

DISKDEV 1

Análise da Hierarquia das Tarefas:



CLIENTE

```
public class HorusRemote implements Runnable {
    public HorusRemote() throws Exception;
    public int serverConnect(String username, String password);
    public int sendMessage(String data);
    public int startMessageLoop();
    public void run();   // Message Loop
}
```

CLIENT ACCESS DRIVER (JAVA / PYTHON)
COMMAND REQUESTER
COMMAND RECEIVER

SERVIDOR
(HOST - x86_64)

```
class CcmdListener {
    CCmdListener();
    void start();
    int sendStringMessageToCon(connection_t* pCon, char* data);
    int sendDataMessageToCon(connection_t* pCon, byte* data, int n);
};
extern void* cmd_listener_exec(void* _pArgs);
extern void* cmd_listener_run(void* _pArgs);
```

SERVER LISTENER
COMMAND INTERPRETER
COMMAND EXECUTER
TABLEMETADATA SEARCH
TABLESPACE SEARCH
REMOTE DISPATCHER
RETURN RESULTS

CSDs
(WORKERS)

```
class CCmdListener {
    CCmdListener();
    void start();
    int sendStringMessageToHost(char* data);
    int sendDataMessageToHost(byte* data, int n);
    int sendStringMessageToCurrRemoteUnit(char* data);
    int sendDataMessageToCurrRemoteUnit(byte* data, int n);
};
extern void* cmd_listener_exec(void* _pArgs);
extern void* cmd_listener_run(void* _pArgs);
```

WORKER LISTENER
TABLE SEARCH
IMAGE TABLE SEARCH
FILE LOCATOR (DIRECTORY)
BLOCK LOCATOR (DATABLOCKS)
REMOTE FILE REQUESTER
DATA PROCESSOR (MODULE)
RETURN DATA RESULTS

CSDs
(UNIT LEVEL)

```
class CDskDiskMan {
    CDskDiskMan();
    void initDiskGroup();
    int doMakeDir();
    int doListDir();
    int doCopyFile();
    int doRenameFile();
    int doWriteFile();
    int doReadFile();
};
```

```
class CExtSched {
    CExtSched();
    int initModule();
    int start();
    int execute();
    int trainModel();
    int classify();
};
extern void* ext_sched_start(void* _pArgs);
extern void* ext_sched_execute(void* _pArgs);
extern void* ext_sched_train(void* _pArgs);
extern void* ext_sched_classify(void* _pArgs);
```

FILE REQUEST LISTENER
FILE WRITER
FILE READER
RETURN FILE RESULTS

EXTERNAL MODULES SCHED
EXTERNAL MODULES

CSDs
(PARTITION LEVEL)

```
class CDskDiskGroup {
    CDskDiskGroup();
    CDskDiskDev* addDiskDev();
    CDskDiskDev* getDiskDevAtPtr(int
pos);
    CDskDiskDev* getDiskDevPtr(long
oid);
    int getCurrNumDiskDev();
};
```

```
class CDskDiskDev {
    CDskDiskDev();
    int writeBlock();
    int readBlock();
};
```

BLOCK DEVICE ACCESS
BLOCK WRITER
BLOCK READER
RETURN BLOCK RESULTS

Ponto Utilização Framework Sucuri:

CLIENTE

IHorusRemote

```
class HorusRemote(th.Thread):
    def __init__(self)
    def serverConnect(self, username, password)
    def sendMessage(self, cmd)
    def startMessageLoop(self)
    def run(self)        ## Message Loop
```

CLIENT ACCESS DRIVER (JAVA / PYTHON)
COMMAND REQUESTER
COMMAND RECEIVER

NOTAS:

#1. A UTILIZAÇÃO DO FRAMEWORK SUCURI OCORRE NA DISTRIBUIÇÃO DAS TAREFAS EXECUTADAS PELA APLICAÇÃO CLIENTE.

#2. CADA TAREFA REALIZA UM CONJUNTO DE CHAMADAS REMOTAS AO SERVIDOR HORUS IMAGE SERVER, ATRAVÉS DA INTERFACE IHORUSREMOTE.

#3. NO EXEMPLO, USAMOS O FRAMEWORK SUCURI PARA EXECUÇÃO DISTRIBUIDA DE 5 (CINCO) TAREFAS: f(), g(), h(), w(), t(). ESPALHADAS POR 6 (SEIS) NÓS DE EXECUÇÃO: a, b, c, d, e, x.

#4. O FRAMEWORK SUCURI EXIGE A DEFINIÇÃO DAS FUNÇÕES, DOS NÓS DE PROCESSAMENTO E DO FLUXO DE DADOS DE EXECUÇÃO (GRAPHO DE EXECUÇÃO).

EXEMPLO FRAMEWORK SUCURI

```
def cabecalho():
    print("\nCOPPE/UFRJ 30/JAN/2021\n")
    print("Estudo para Tese D.Sc.\n")
    print("Nome: Luiz Marcio Faria de Aquino Viana")
    print("DRE: 120048833")
    print("CPF: 024.723.347-10")
    print("RG: 08855128-8 IFP-RJ\n")
    print("001-HorusDbCmdTest.py (Horus Database Command Test)\n")
def f()
def g()
def h()
def w()
def t()
cabecalho()
horus = hdb.HorusRemote()
horus.startMessageLoop()
horus.serverConnect(hdb.HORUS_DEF_DEFAULT_CLIENTUSERNAME,
hdb.HORUS_DEF_DEFAULT_CLIENTPASSWORD)
nWorkers = 2
graph = DFGraph()
sched = Scheduler(graph, nWorkers, False)
a = Node(f, 0)
b = Node(g, 0)
c = Node(h, 2)
d = Node(h, 2)
e = Node(w, 2)
x = Node(t, 2)
graph.add(a)
graph.add(b)
graph.add(c)
graph.add(d)
graph.add(e)
graph.add(x)
a.add_edge(c, 0)
b.add_edge(c, 1)
a.add_edge(d, 0)
c.add_edge(d, 1)
c.add_edge(e, 0)
b.add_edge(e, 1)
d.add_edge(x, 0)
e.add_edge(x, 1)
sched.start()
horus.serverTerminate()
horus.stopMessageLoop()
```

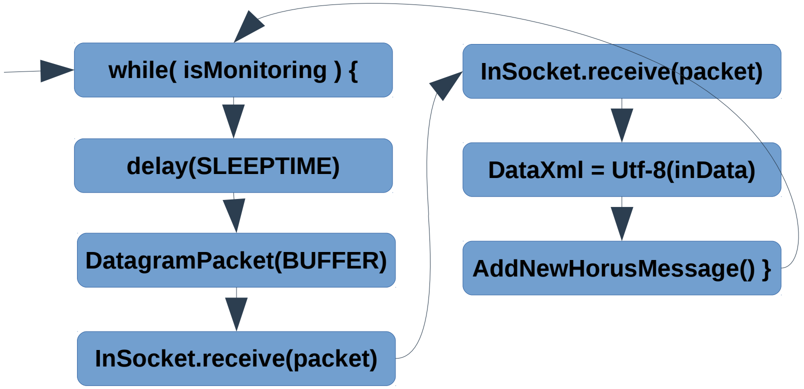
The Sucuri code is created following the steps:

Step 1: Import Sucuri Library
Step 2: Is created an empty Sucuri graph
Step 3: Is created the Sucuri scheduler, passing the graph and the maximum number of worker threads.
Step 4: The data flow nodes are inserted at the graph.
Step 5: The data flow edges are inserted at the graph.
Step 6: The Sucuri scheduler is started.

HorusRemote – Client Access Driver

```
public class HorusRemote implements Runnable {
    // input socket
    private String inAddr = HorusDefs.HORUS_DEFAULT_CLIENTRECVADDRESS;
    private int inPort = HorusDefs.HORUS_DEFAULT_CLIENTRECVPORT;
    private SocketAddress inEndpoint = null;
    private DatagramSocket inSocket = null;
    // output socket
    private String outAddr = HorusDefs.HORUS_DEFAULT_CLIENTTARGETADDRESS;
    private int outPort = HorusDefs.HORUS_DEFAULT_CLIENTTARGETPORT;
    private SocketAddress outEndpoint = null;
    private DatagramSocket outSocket = null;
    // session data
    private String uid = "-1";
    private int user_oid = -1;
    private String username = "";
    // request id
    private int reqnum = 0;
    // connection opened flag
    private int isConnected = HorusDefs.HORUS_FALSE;
    // monitor thread
    private Thread monitorThread = null;
    private int isMonitorRunning = HorusDefs.HORUS_FALSE;
    private ArrayList<HorusMessage> lsMessage = null;
    private void delay(long sleepTimeInSec);
//Public
    public HorusRemote() throws Exception;
    /* Methodes */
    public int sendMessage(String data);
    // serverConnect(): funcao que envia mensagem para iniciar uma conexao com o servidor de imagens
    // retorno: RSCODE
    public int serverConnect(String username, String password);
    // serverTerminate(): funcao que envia mensagem de termino do servidor de imagens
    // retorno: RSCODE
    public int serverTerminate();
    // listAllUsers(): funcao que obtem uma lista dos usuarios do banco de imagens
    // retorno: arquivo XML com a lista de usuarios
    public int listAllUsers();
    // listAllTables(): funcao que obtem uma lista das tabelas do banco de imagens
    // retorno: arquivo XML com a lista de tabelas
    public int listAllTables();
    // startMessageLoop(): funcao que inicia o loop de mensagens com o servidor de imagens
    // retorno: RSCODE
    public int startMessageLoop();
    // stopMessageLoop(): funcao que termina o loop de mensagens com o servidor de imagens
    // retorno: RSCODE
    public int stopMessageLoop();
    /* Getters/Setters */
    public synchronized void addNewHorusMessage(String dataXml, String inAddr);
    public synchronized HorusMessage getNextHorusMessage();
    /* Monitor */
    public void run();
}
```

1# void run() (Message Loop)



while(isMonitoring) {

delay(SLEEPTIME)

DatagramPacket(BUFFER)

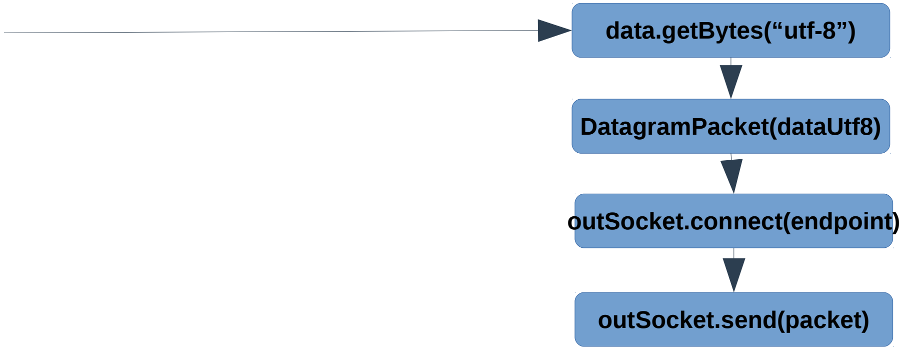
InSocket.receive(packet)

InSocket.receive(packet)

DataXml = Utf-8(inData)

AddNewHorusMessage() }

2# int sendMessage()



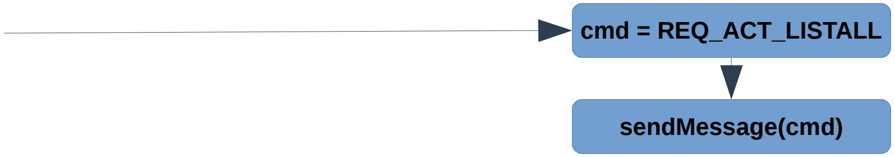
data.getBytes(“utf-8”)

DatagramPacket(dataUtf8)

outSocket.connect(endpoint)

outSocket.send(packet)

3# int listAllTables()

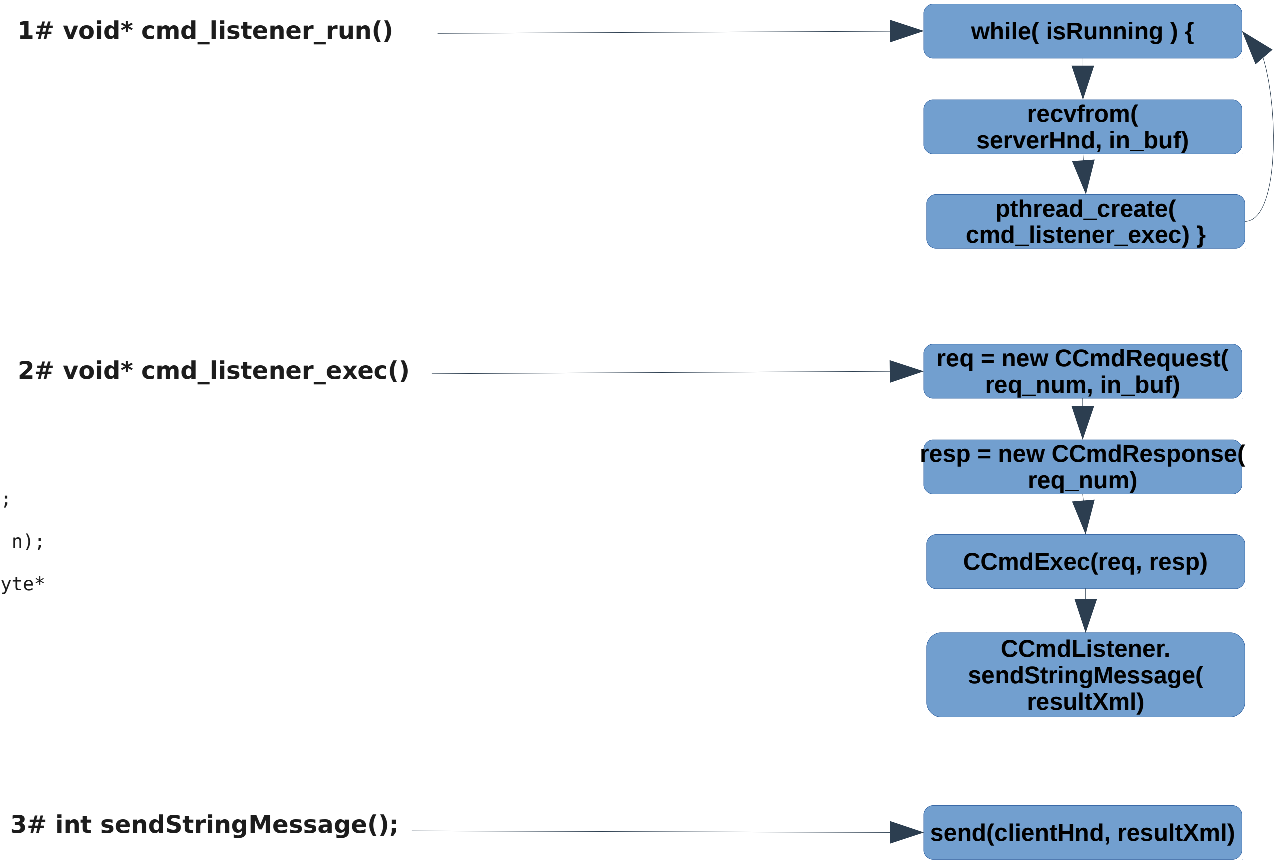


cmd = REQ_ACT_LISTALL

sendMessage(cmd)

HorusSrv – Server Listener

```
class CCmdListener {
private:
    pthread_t m_threadHnd;
    int m_isRunning;
    str_t m_ipAddr;
    long m_inPort;
    long m_outPort;
    connection_t m_arrClientCon[DEF_MAX_NUM_CONNECTION];
    int m_currNumClientCon;
public:
    CCmdListener();
    ~CCmdListener();
    void start();
    void stop();
    void test();
    void exec();
    void remoteExec1();
    void remoteExec2();
    /* Methodes */
    int sendStringMessageToCon(connection_t* pCon, char* data);
    int sendDataMessageToCon(connection_t* pCon, byte* data, int n);
    int sendStringMessageToHost(char* data);
    int sendDataMessageToHost(byte* data, int n);
    int sendStringMessageToListener(char* data);
    int sendDataMessageToListener(byte* data, int n);
    int sendStringMessageToRemoteUnit(long remoteUnitOid, char* data);
    int sendDataMessageToRemoteUnit(long remoteUnitOid, byte* data, int n);
    int sendStringMessageToAny(char* dst_addr, int dst_port, char* data);
    int sendDataMessageToAny(char* dst_addr, int dst_port, byte* data, int n);
    // MAIN_METHOD: sendDataMessageToINET()
    int sendDataMessageToINET(struct sockaddr_in* p_saddr, long saddrsz, byte*
data, int n);
    /* Getters/Setters */
    connection_t* getAvailableConnection();
    void resetConnection(connection_t* pCon);
    void resetAllConnections();
    int isRunning();
    void setRunning(int bRunning);
    char* getIpAddr();
    void setIpAddr(char* ipAddr);
    long getInPort();
    void setInPort(long inPort);
    long getOutPort();
    void setOutPort(long outPort);
};

/* NEW_TYPE: CCmdListener Pointer */
typedef CCmdListener* CCmdListenerPtr_t;

/* Command Listener Threads */
extern CCmdListener gCmdListener;
extern void* cmd_listener_exec(void* _pArgs);
extern void* cmd_listener_run(void* _pArgs);
```



1# void* cmd_listener_run()

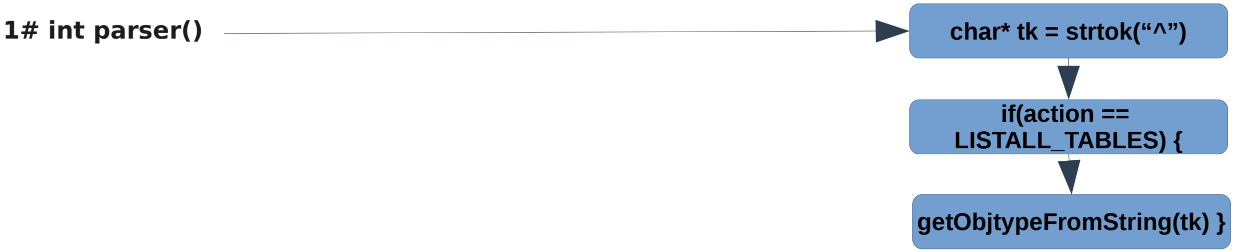
2# void* cmd_listener_exec()

3# int sendStringMessage();

while(isRunning) {
recvfrom(serverHnd, in_buf)
pthread_create(cmd_listener_exec) }
req = new CCmdRequest(req_num, in_buf)
resp = new CCmdResponse(req_num)
CCmdExec(req, resp)
CCmdListener. sendStringMessage(resultXml)
send(clientHnd, resultXml)

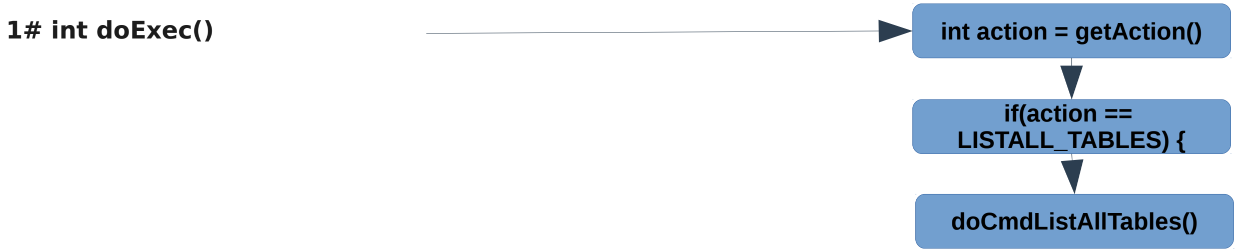
HorusSrv – Command Interpreter

```
class CCmdParser
{
private:
    long m_action;
    long m_objtype;
    str_t m_username;
    str_t m_password;
    str_t m_session;
    long m_oid;
    str_t m_table_name;
    long m_rowid;
    bigstr_t m_data;
    long m_datasz;

    long getActionFromString(char* str);
    long getObjtypeFromString(char* str);

public:

    CCmdParser();

    ~CCmdParser();

    /* Methodes */

    int parser(char* cmd);

    /* Getters/Setters */

    long getAction();
    long getObjtype();
    char* getUsername();
    char* getPassword();
    char* getSession();
    long getOid();
    char* getTableName();
    long getRowid();
    char* getData();
    long getDataSz();

};

/* NEW_TYPE: CCmdParser Pointer */

typedef CCmdParser* CCmdParserPtr_t;
```



HorusSrv – Command Executer

```
class CCmdExec
{
private:
    CCmdRequest* m_pRequest;
    CCmdResponse* m_pResponse;

    //*** COMMANDS
    //
    int doCmdConnect();
    //
    int doCmdListAllUsers();
    int doCmdListAllTables();

public:
    CCmdExec(CCmdRequest* pRequest, CCmdResponse* pResponse);
    ~CCmdExec();

    /* Methodes */

    int doExec();

    int doExecTest();

    /* Getters/Setters */

    CCmdRequest* getRequestPtr();

    CCmdResponse* getResponsePtr();

};

/* NEW_TYPE: CCmdExec Pointer */

typedef CCmdExec* CCmdExecPtr_t;
```

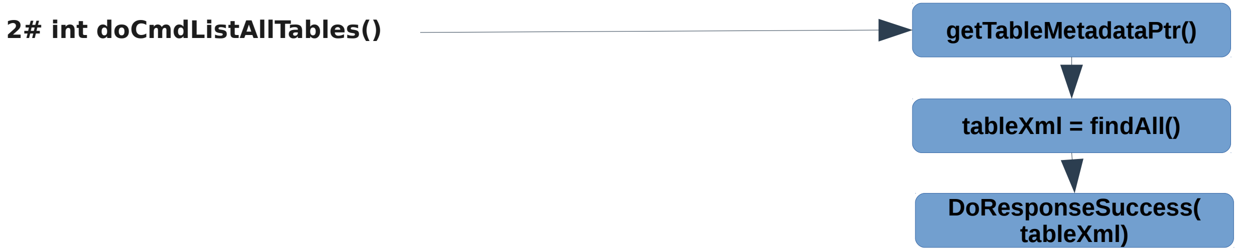


1# int doExec()

int action = getAction()

if(action ==
LISTALL_TABLES) {

doCmdListAllTables()



2# int doCmdListAllTables()

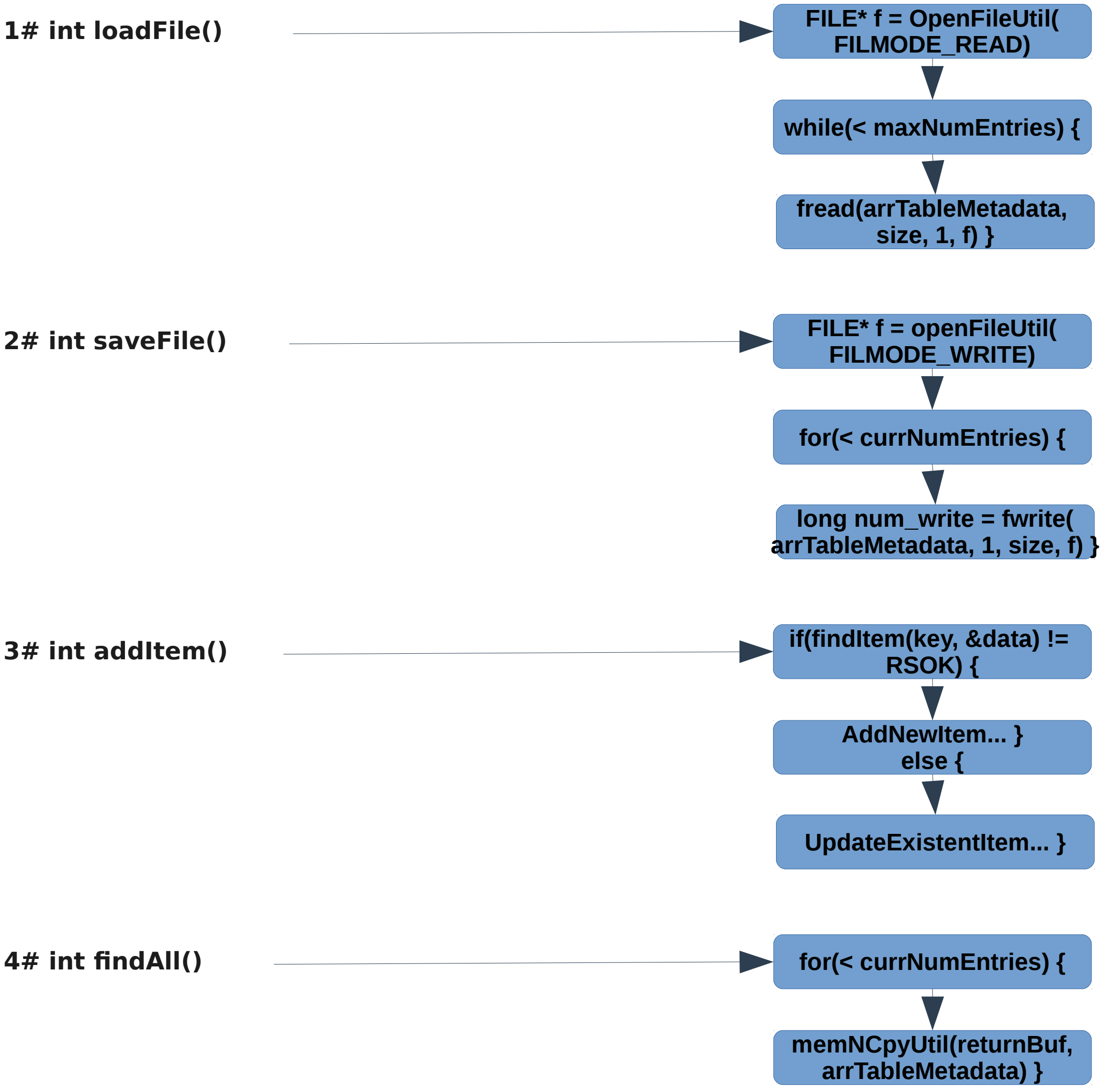
getTableMetadataPtr()

tableXml = findAll()

DoResponseSuccess(
tableXml)

HorusSrv – TableMetadata Search

```
class CTableMetadata
{
private:
    table_metadata_t* m_arrTableMetadata;
    int m_maxNumEntries;
    int m_currNumEntries;
public:
    CTableMetadata(int maxNumEntries);
    ~CTableMetadata();
    int loadFile(char* fileName);
    int saveFile(char* fileName);
    int addItem(char* key,long* tablespace_oid,long num_tablespaces,long num_entries,
        long table_type,long reference_table_oid,char* module_name,char* table_name,
        char* table_filename,char* idx1_name,char* idx1_filename,
        char* idx2_name,char* idx2_filename,char* seq_name,long uid);
    int addItem(long oid,char* key,long* tablespace_oid,long num_tablespaces,
        long num_entries,long table_type,long reference_table_oid,char* module_name,
        char* table_name,char* table_filename,char* idx1_name,char* idx1_filename,
        char* idx2_name,char* idx2_filename,char* seq_name,long uid);
    int findItem(long oid, table_metadata_t** resval);
    int findItem(char* key, table_metadata_t** resval);
    int findAll(table_metadata_t** arr, long* arrsz);
    int findItemByModuleName(char* module_name, table_metadata_t** resval);
    int deleteItem(long oid, long uid);
    int deleteItem(char* key, long uid);
    table_metadata_t* newItemData(table_metadata_t** data);
    table_metadata_t* setItemData(table_metadata_t** data,long oid,long key_hash,
        long* tablespace_oid,long num_tablespaces,long num_entries,long create_uid,
        long modify_uid,long delete_uid,long create_date,long modify_date,
        long delete_date,long is_modified,long is_deleted,long table_type,
        long reference_table_oid,char* module_name,char* table_name,
        char* table_filename,char* idx1_name,char* idx1_filename,char* idx2_name,
        char* idx2_filename,char* seq_name,char* key);
    /* Debug */
    void debug(int debugLevel, const char* className, const char* methodName);
    void debugEntry(int debugLevel,const char* className,
        const char* methodName, table_metadata_t* p);
    void showAll();
    void showEntry(table_metadata_t* p);
    char* toXml(table_metadata_t* p, char* returnXml);
    int parserXml(table_metadata_t* p, char* xmlData);
    /* Getters/Setters */
    table_metadata_t* getDataTable();
    table_metadata_t* getItemAt(int pos);
    int getMaxNumEntries();
    int getCurrNumEntries();
};

/* NEW_TYPE: CTableMetadata Pointer */

typedef CTableMetadata* CTableMetadataPtr_t;
```



1# int loadFile()

FILE* f = OpenFileUtil(FILMODE_READ)

while(< maxNumEntries) {

fread(arrTableMetadata, size, 1, f) }

2# int saveFile()

FILE* f = openFileUtil(FILMODE_WRITE)

for(< currNumEntries) {

long num_write = fwrite(arrTableMetadata, 1, size, f) }

3# int addItem()

if(findItem(key, &data) != RSOK) {

AddNewItem... } else {

UpdateExistentItem... }

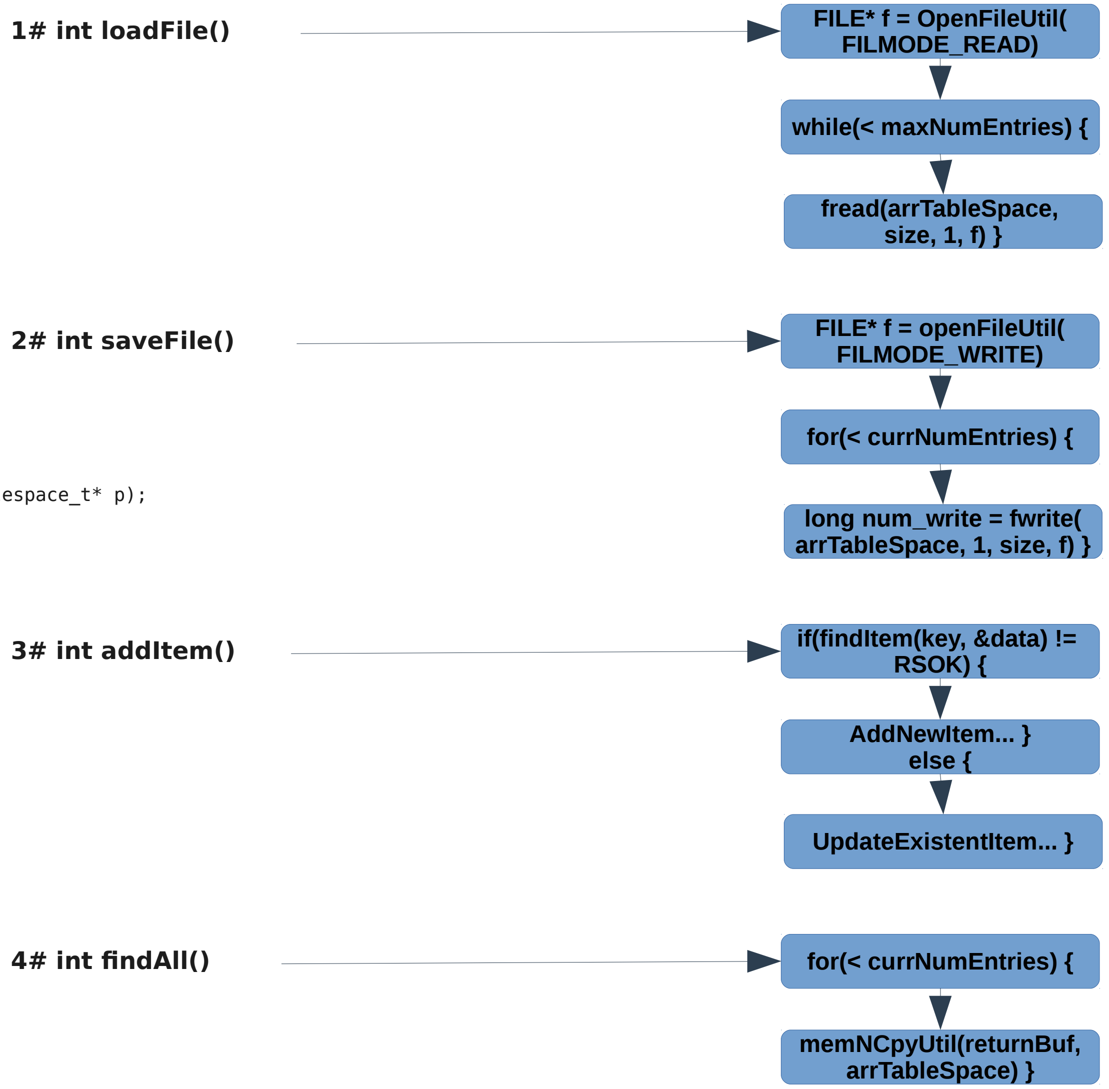
4# int findAll()

for(< currNumEntries) {

memNCpyUtil(returnBuf, arrTableMetadata) }

HorusSrv – TableSpace Search

```
class CTableSpace
{
private:
    tablespace_t* m_arrTableSpace;
    int m_maxNumEntries;
    int m_currNumEntries;
public:
    CTableSpace(int maxNumEntries);
    ~CTableSpace();
    int loadFile(char* fileName);
    int saveFile(char* fileName);
    int addItem(char* key, char* name, char* path, long uid);
    int addItemEx(long oid, char* key, char* name, char* path, long uid);
    int findItem(long oid, tablespace_t** resval);
    int findItem(char* key, tablespace_t** resval);
    int findAll(tablespace_t** arr, long* arrsz);
    int deleteItem(long oid, long uid);
    int deleteItem(char* key, long uid);
    tablespace_t* newItemData(tablespace_t** data);
    tablespace_t* setItemData(
        tablespace_t** data,long oid,long key_hash,long create_uid,
        long delete_uid,long create_date,long delete_date,long is_deleted,
        char* tablespace_name,char* tablespace_path,char* key);
    void debug(int debugLevel, const char* className, const char* methodName);
    void debugEntry(int debugLevel, const char* className, const char* methodName, tablespace_t* p);
    void showAll();
    void showEntry(tablespace_t* p);
    char* toXml(tablespace_t* p, char* returnXml);
    int parserXml(tablespace_t* p, char* xmlData);
};

/* NEW_TYPE: CTableSpace Pointer */

typedef CTableSpace* CTableSpacePtr_t;
```



1# int loadFile()

FILE* f = OpenFileUtil(FILMODE_READ)

while(< maxNumEntries) {

fread(arrTableSpace, size, 1, f) }

2# int saveFile()

FILE* f = openFileUtil(FILMODE_WRITE)

for(< currNumEntries) {

long num_write = fwrite(arrTableSpace, 1, size, f) }

3# int addItem()

if(findItem(key, &data) != RSOK) {

AddNewItem... } else {

UpdateExistentItem... }

4# int findAll()

for(< currNumEntries) {

memNCpyUtil(returnBuf, arrTableSpace) }

HorusSrv – Remote Dispatcher

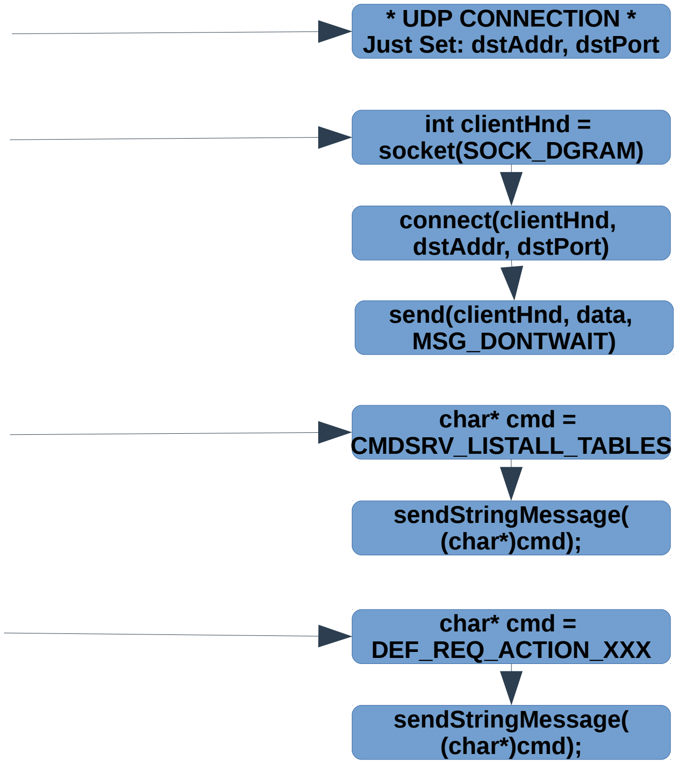
```
class CCmdDispatch
{
private:
    str_t m_chname;
    int m_reqnum;
public:
    CCmdDispatch(char* chname);
    ~CCmdDispatch();
    /* Local Methodes */
    int serverConnect(char* username, char* passwd);
    int serverTerminate();
    int listAllUsers(users_t** ret_datarows, long* ret_numrows, char* sessionId);
    int listAllTables(table_metadata_t** ret_datarows, long* ret_numrows, char* sessionId);
    /* Remote Unit Methodes */
    int serverTerminate(CCfgRemoteUnit* pRemoteUnit);
    //MKDIR
    int doActionTS1MkDir(CCfgRemoteUnit* pRemoteUnit, long currpart, long numparts);
    //UPLOAD
    int doActionTS1Upload(CCfgRemoteUnit* pRemoteUnit, long currpart, long numparts);
    //DOWNLOAD
    int doActionTS1Download(CCfgRemoteUnit* pRemoteUnit, long currpart, long numparts);
    //DOWNLOAD_AND_PROCESS
    int doActionTS1DownloadAndProcess(CCfgRemoteUnit* pRemoteUnit, long currpart, long numparts, long mode);
    //CONVERT_MOD
    int doActionTS1Convert(CCfgRemoteUnit* pRemoteUnit, long currpart, long numparts);
    //REPROJ_MOD
    int doActionTS1Reproj(CCfgRemoteUnit* pRemoteUnit, long currpart, long numparts);
    //SIMPL_MOD
    int doActionTS1Simpl(CCfgRemoteUnit* pRemoteUnit, long currpart, long numparts);
    /* Getters/Setters */
    char* getChName();
    int getReqNum();
};

/* NEW_TYPE: CCmdDispatch Pointer */

typedef CCmdDispatch* CCmdDispatchPtr_t;
```

1# int serverConnect()

2# int sendStringMessage()

3# int listAllTables()

4# int doActionXXX()



* UDP CONNECTION *
Just Set: dstAddr, dstPort

int clientHnd =
socket(SOCK_DGRAM)

connect(clientHnd,
dstAddr, dstPort)

send(clientHnd, data,
MSG_DONTWAIT)

char* cmd =
CMDSRV_LISTALL_TABLES

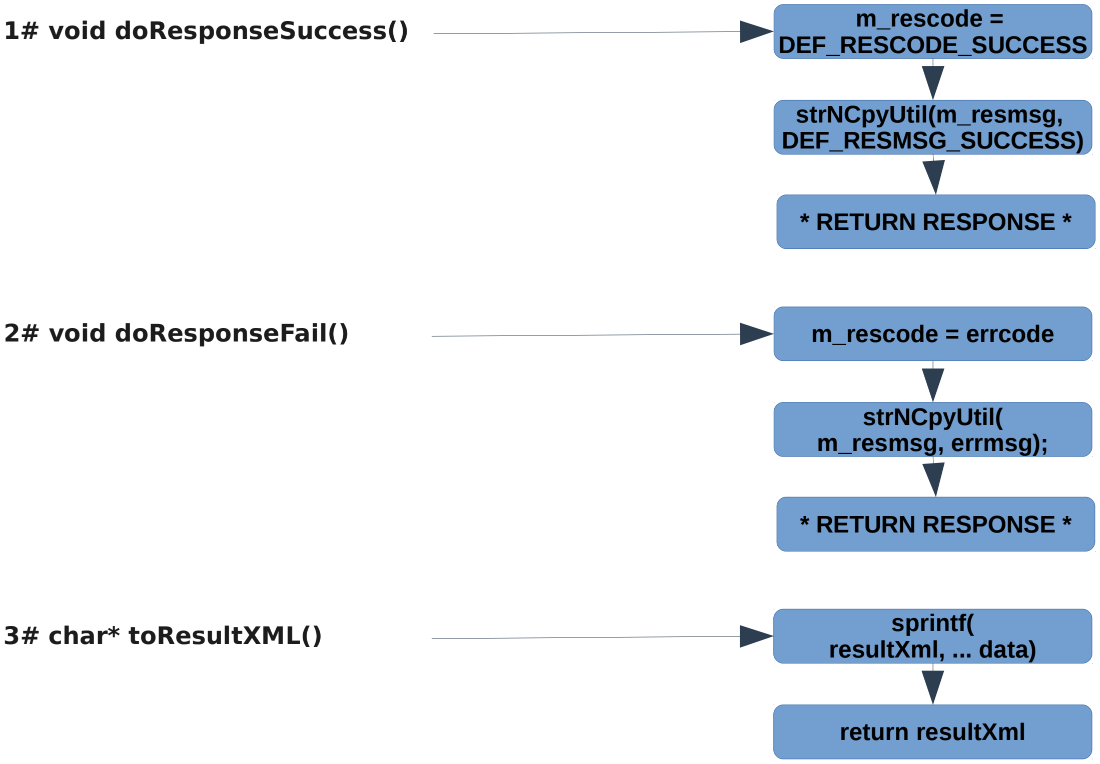
sendStringMessage(
(char*)cmd);

char* cmd =
DEF_REQ_ACTION_XXX

sendStringMessage(
(char*)cmd);

HorusSrv – Return Results

```
class CCmdResponse
{
private:
    long m_reqnum;
    long m_reqtimestamp;
    long m_rescode;
    bigstr_t m_resmsg;
    long m_restype;
    long m_numval;
    bigstr_t m_strval;
    char* m_data;
    long m_datasz;
public:
    CCmdResponse(long reqnum);
    ~CCmdResponse();
    /* Methodes */
    void doResponseSuccess(long numval);
    void doResponseSuccess(char* strval);
    void doResponseSuccess(long restype, long numval);
    void doResponseSuccess(long restype, char* strval);
    void doResponseSuccess(long restype, char* data, long datasz);
    void doResponseFail(long errcode, char* errmsg);
    char* toResultXml(char* resultXml);
    void debug();
    /* Getters/Setters */
    long getReqNum();
    long getReqTimestamp();
    long getResCode();
    char* getResMsg();
    long getResType();
    long getNumVal();
    char* getStrVal();
    char* getData();
    long getDataSz();
};

/* NEW_TYPE: CCmdResponse Pointer */

typedef CCmdResponse* CCmdResponsePtr_t;
```



1# void doResponseSuccess()

m_rescode = DEF_RESCODE_SUCCESS

strNCpyUtil(m_resmsg, DEF_RESMSG_SUCCESS)

* RETURN RESPONSE *

2# void doResponseFail()

m_rescode = errcode

strNCpyUtil(m_resmsg, errmsg);

* RETURN RESPONSE *

3# char* toResultXML()

sprintf(resultXml, ... data)

return resultXml

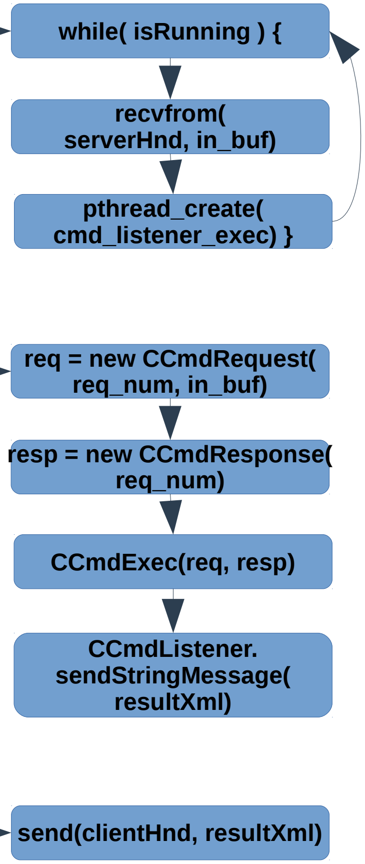
HorusWrk – Worker Listener

```
class CCmdListener
{
private:
    pthread_t m_threadHnd;
    int m_isRunning;
    str_t m_ipAddr;
    long m_inPort;
    long m_outPort;
    connection_t m_arrClientCon[DEF_MAX_NUM_CONNECTION];
    int m_currNumClientCon;
public:
    CCmdListener();
    ~CCmdListener();
    void start();
    void stop();
    void test();
    /* Methodes */
    int sendStringMessageToHost(char* data);
    int sendDataMessageToHost(byte* data, int n);
    int sendStringMessageToCurrRemoteUnit(char* data);
    int sendDataMessageToCurrRemoteUnit(byte* data, int n);
    int sendStringMessageToRemoteUnit(long remoteUnitOid, char* data);
    int sendDataMessageToRemoteUnit(long remoteUnitOid, byte* data, int n);
    int sendStringMessageToAny(char* dst_addr, int dst_port, char* data);
    int sendDataMessageToAny(char* dst_addr, int dst_port, byte* data, int n);
    // AUXILIARY_METHODS: sendStringMessageToCon() / sendDataMessageToCon()
    int sendStringMessageToCon(connection_t* pCon, char* data);
    int sendDataMessageToCon(connection_t* pCon, byte* data, int n);
    // MAIN_METHOD: sendDataMessageToINET()
    int sendDataMessageToINET(struct sockaddr_in* p_saddr, long saddrsz, byte* data, int n);
    /* Getters/Setters */
    connection_t* getAvailableConnection();
    void resetConnection(connection_t* pCon);
    void resetAllConnections();
    int isRunning();
    void setRunning(int bRunning);
    char* getIpAddr();
    void setIpAddr(char* ipAddr);
    long getInPort();
    void setInPort(long inPort);
    long getOutPort();
    void setOutPort(long outPort);
};

/* NEW_TYPE: CCmdListener Pointer */
typedef CCmdListener* CCmdListenerPtr_t;

/* Command Listener Threads */
extern CCmdListener gCmdListener;
extern void* cmd_listener_exec(void* _pArgs);
extern void* cmd_listener_run(void* _pArgs);
```

1# void* cmd_listener_run()

2# void* cmd_listener_exec()

3# int sendStringMessage();



while(isRunning) {

recvfrom(serverHnd, in_buf)

pthread_create(cmd_listener_exec) }

req = new CCmdRequest(req_num, in_buf)

resp = new CCmdResponse(req_num)

CCmdExec(req, resp)

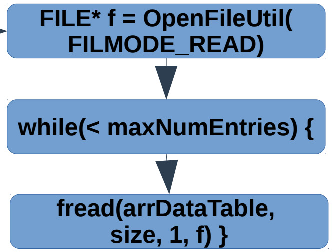
CCmdListener. sendStringMessage(resultXml)

send(clientHnd, resultXml)

HorusWrk – Data Table Search

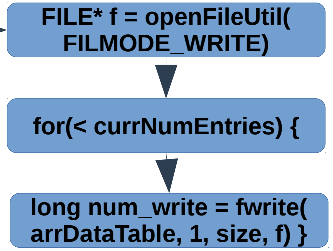
```
class CDataTable
{
private:
    str_t m_dataSeqName;
    str_t m_dataTableName;
    str_t m_dataTableFileName;
    pathname_t m_dataTableFullPath;
    data_table_t* m_arrDataTable;
    int m_maxNumEntries;
    int m_currNumEntries;
    long calculateDataPart(long curr_part, long num_parts, long* part_start);
    long checkScoreList(double* score_ls, long scoresz, double score_val);
public:
    CDataTable(char* seqName, char* tableName, char* tableFileName, char* tableFullPath, int maxNumEntries);
    ~CDataTable();
    int loadFile(char* fileName);
    int saveFile(char* fileName);
    int addItem(
        char* key,long image_oid,long is_train_data,double train_score,
        long is_test_data,double test_score,long is_classified_data,double classification_score,
        long xmin,long ymin,long xmax,long ymax);
    int addItem(
        long oid,char* key,long image_oid,long is_train_data,double train_score,
        long is_test_data,double test_score,long is_classified_data,double classification_score,
        long xmin,long ymin,long xmax,long ymax);
    int findAll(data_table_t** arr, long* arrsz);
    long findAllByClassifScoreList(data_table_t** arr, long* arrsz, double* score_ls, long scoresz, long curr_part, long num_parts);
    int findItem(long oid, data_table_t** resval);
    int findItem(char* key, data_table_t** resval);
    int deleteItem(long oid, long uid);
    int deleteItem(char* key, long uid);
    data_table_t* newItemData(data_table_t** data);
    data_table_t* setItemData(
        data_table_t** data,long oid,long key_hash,long image_oid,long create_date,long modify_date,
        long is_modified,long update_date,long is_updated,long delete_date,long is_deleted,long is_train_data,
        double train_score,long is_test_data,double test_score,long is_classified_data,double classification_score,
        long xmin,long ymin,long xmax,long ymax,char* key);
    /* Debug */
    void debug(int debugLevel, const char* className, const char* methodName);
    void debugEntry(int debugLevel, const char* className, const char* methodName, data_table_t* p);
    char* toXml(data_table_t* p, char* returnXml);
    int parserXml(data_table_t* p, char* xmlData);
    /* Getters/Setters */
    char* getDataSeqName();
    char* getDataTableName();
    char* getDataTableFileName();
    char* getDataTableFullPath();
    data_table_t* getDataTable();
    int getMaxNumEntries();
    int getCurrNumEntries();
};

/* NEW_TYPE: CDataTable Pointer */
typedef CDataTable* CDataTablePtr_t;
```

1# int loadFile()



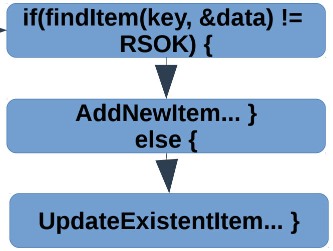
```
FILE* f = OpenFileUtil(
FILMODE_READ)
while(< maxNumEntries) {
fread(arrDataTable,
size, 1, f) }
```

2# int saveFile()



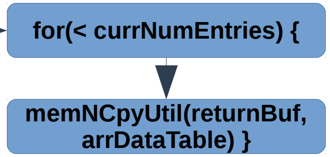
```
FILE* f = openFileUtil(
FILMODE_WRITE)
for(< currNumEntries) {
long num_write = fwrite(
arrDataTable, 1, size, f) }
```

3# int addItem()



```
if(findItem(key, &data) !=
RSOK) {
AddNewItem... }
else {
UpdateExistentItem... }
```

4# int findAll()

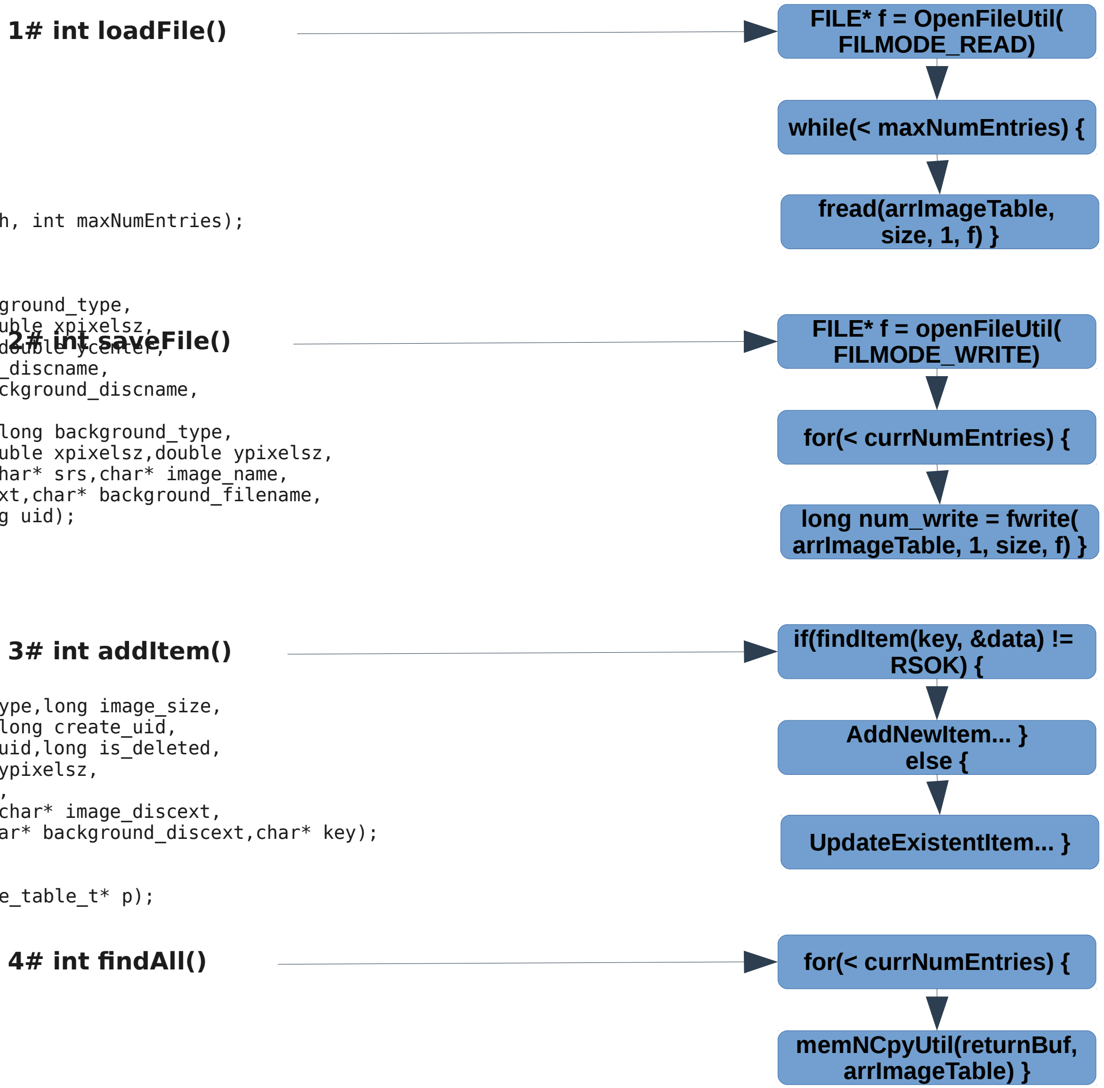


```
for(< currNumEntries) {
memNCpyUtil(returnBuf,
arrDataTable) }
```

```
// scoresz - size of score array
// curr_part - number of the current part
// num_parts - number of parts
// RETURN: number of selected rows
long findAllByClassifScoreList(data_table_t** arr, long* arrsz, double* score_ls, long scoresz, long curr_part, long num_parts);

// findItem(): find item in data table
// oid - data table oid
// resval - result data
int findItem(long oid, data_table_t** resval);

// findItem(): find item in data table
// key - data table key
// resval - result data
int findItem(char* key, data_table_t** resval);

// deleteItem(): mark item as deleted in data table
// oid - data table oid
// uid - user id
int deleteItem(long oid, long uid);

// deleteItem(): mark item as deleted in data table
// key - data table key
// uid - user id
int deleteItem(char* key, long uid);

// newItemData(): new item data
// data - image table reference
data_table_t* newItemData(data_table_t** data);

// setItemData(): set item values in data table
// data - data reference
// oid - object id
// key_hash - record key hash
// image_oid - referenced image identifier
// create_date - create timestamp
// modify_date - modify timestamp
// is_modified - record is modified
// update_date - update timestamp
// is_updated - record is updated
// delete_date - delete timestamp
// is_deleted - record is deleted
// is_train_data  - identify if is train data
// train_score - score of train data
// is_test_data - identify if is test data
// test_score - score of test data
// is_classified_data - identify if is classified data
// classification_score - score of classified data
// xmin - x of lower left corner of image area
// ymin - y of lower left corner of image area
// xmax - x of upper right corner of image area
// ymax - y of upper right corner of image area
// key - record key
data_table_t* setItemData(
    data_table_t** data,
    long oid,
    long key_hash,
    long image_oid,
    long create_date,
    long modify_date,
    long is_modified,
    long update_date,
    long is_updated,
    long delete_date,
    long is_deleted,
    long is_train_data,
    double train_score,
    long is_test_data,
    double test_score,
    long is_classified_data,
    double classification_score,
    long xmin,
    long ymin,
    long xmax,
    long ymax,
    char* key);

/* Debug */

// debug(): show all tablespace data
// debugLevel - debug level
// className - class name
// methodName - method name
void debug(int debugLevel, const char* className, const char* methodName);

// debugEntry(): show tablespace data
// debugLevel - debug level
// className - class name
// methodName - method name
// p - pointer to a tablespace entry
void debugEntry(int debugLevel, const char* className, const char* methodName, data_table_t* p);

// toXml(): show tablespace data
// p - pointer to a tablespace entry
// returnXml - XML return data
char* toXml(data_table_t* p, char* returnXml);

// parserXml(): xml data entry parser
// p - pointer reference
// xmlData - xml data
int parserXml(data_table_t* p, char* xmlData);

/* Getters/Setters */

char* getDataSeqName();

char* getDataTableName();

char* getDataTableFileName();

char* getDataTableFullPath();

data_table_t* getDataTable();

int getMaxNumEntries();

int getCurrNumEntries();

};

/* NEW_TYPE: CDataTable Pointer */

typedef CDataTable* CDataTablePtr_t;
```

HorusWrk – Image Table Search

```
class CImageTable
{
private:
    str_t m_imageSeqName;
    str_t m_imageTableName;
    str_t m_imageTableFileName;
    pathname_t m_imageTableFullPath;
    image_table_t* m_arrImageTable;
    int m_maxNumEntries;
    int m_currNumEntries;
public:
    CImageTable(char* seqName, char* tableName, char* tableFileName, char* tableFullPath, int maxNumEntries);
    ~CImageTable();
    int loadFile(char* fileName);
    int saveFile(char* fileName);
    int addItem(char* key,long image_type,long image_size,long has_background,long background_type,
        long background_size,long xpixels,long ypixels,double xorigin,double yorigin,double xpixelsz,
        double ypixelsz,double xmin,double ymin,double xmax,double ymax,double xcenter,double ycenter,
        char* srs,char* image_name,char* image_filename,char* image_fileext,char* image_discname,
        char* image_discext,char* background_filename,char* background_fileext,char* background_discname,
        char* background_discext,long uid);
    int addItem(long oid,char* key,long image_type,long image_size,long has_background,long background_type,
        long background_size,long xpixels,long ypixels,double xorigin,double yorigin,double xpixelsz,double ypixelsz,
        double xmin,double ymin,double xmax,double ymax,double xcenter,double ycenter,char* srs,char* image_name,
        char* image_filename,char* image_fileext,char* image_discname,char* image_discext,char* background_filename,
        char* background_fileext,char* background_discname,char* background_discext,long uid);
    int findItem(long oid, image_table_t* resval);
    int findItem(char* key, image_table_t* resval);
    image_table_t* findItemPtr(long oid);
    image_table_t* findItemPtr(char* key);
    int findAll(image_table_t** arr, long* arrsz);
    int deleteItem(long oid, long uid);
    int deleteItem(char* key, long uid);
    image_table_t* newItemData(image_table_t** data);
    image_table_t* setItemData(image_table_t** data,long oid,long key_hash,long image_type,long image_size,
        long has_background,long background_type,long create_date,long create_uid,
        long modify_date,long modify_uid,long is_modified,long delete_date,long delete_uid,long is_deleted,
        long xpixels,long ypixels,double xorigin,double yorigin,double xpixelsz,double ypixelsz,
double xmin,double ymin,double xmax,double ymax,double xcenter,double ycenter,char* srs,
        char* image_name,char* image_filename,char* image_fileext,char* image_discname,char* image_discext,
        char* background_filename,char* background_fileext,char* background_discname,char* background_discext,char* key);
    /* Debug */
    void debug(int debugLevel, const char* className, const char* methodName);
    void debugEntry(int debugLevel, const char* className, const char* methodName, image_table_t* p);
    void toXml(image_table_t* p, char* xmlData);
    int parserXml(image_table_t* p, char* xmlData);
    /* Getters/Setters */
    char* getImageSeqName();
    char* getImageTableName();
    char* getImageTableFileName();
    char* getImageTableFullPath();
    image_table_t* getDataTable();
    int getMaxNumEntries();
    int getCurrNumEntries();
};

/* NEW_TYPE: CImageTeble Pointer */
typedef CImageTable* CImageTablePtr_t;
```



1# int loadFile()

FILE* f = OpenFileUtil(FILMODE_READ)

while(< maxNumEntries) {

fread(arrImageTable, size, 1, f) }

2# int saveFile()

FILE* f = openFileUtil(FILMODE_WRITE)

for(< currNumEntries) {

long num_write = fwrite(arrImageTable, 1, size, f) }

3# int addItem()

if(findItem(key, &data) != RSOK) {

AddNewItem... } else {

UpdateExistentItem... }

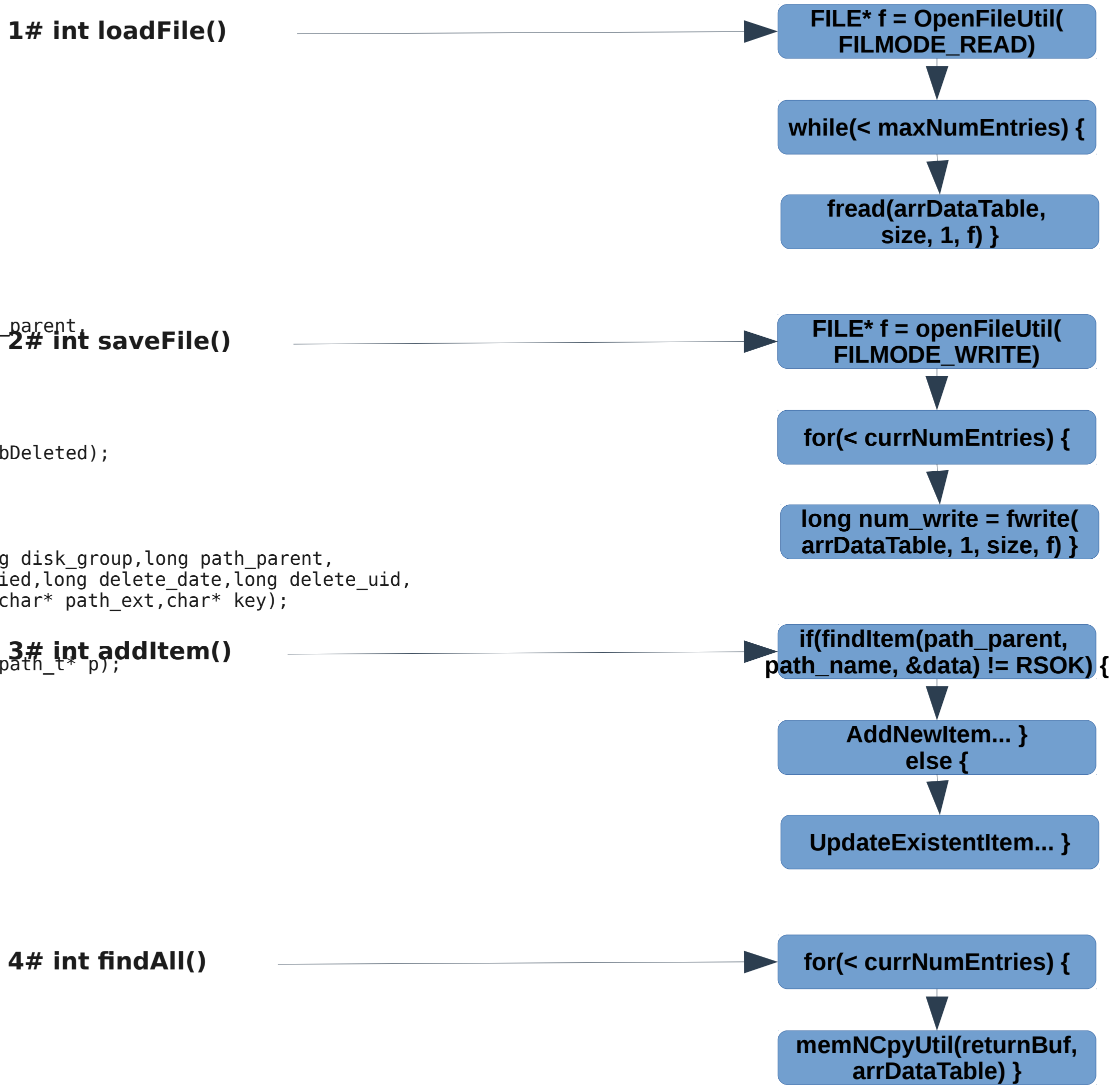
4# int findAll()

for(< currNumEntries) {

memNCpyUtil(returnBuf, arrImageTable) }

HorusWrk – File Locator (Directory)

```
class CDskPathMan
{
private:
    bigstr_t m_dirTableFile;
    pathname_t m_dirTableFilePath;
    dsk_path_t* m_arrDskPath;
    int m_maxNumDskPath;
    int m_currNumDskPath;
public:
    CDskPathMan(int maxNumDskPath, char* dirTableFile, char* dirTableFilePath);
    ~CDskPathMan();
    void init(int maxNumDskPath, char* dirTableFile, char* dirTableFilePath);
    void terminate();
    /* Methodes */
    int loadFile(char* fileName);
    int saveFile(char* fileName);
    int addItem(char* path_name,char* path_ext,long path_type,long disk_group,long path_parent,
        long block_start,long block_qty,long data_size,long uid,long* oid);
    int findItem(long oid, dsk_path_t** resval);
    int findItem(char* key, dsk_path_t** resval);
    int findItem(long path_parent, char* path_name, dsk_path_t** resval);
    int findAll(dsk_path_t** arr, long* arrsz);
    int findAllChildByPathParent(dsk_path_t** arr, long* arrsz, long path_parent, long bDeleted);
    int getNumEntriesByPathParent(long* num_entries, long path_parent, long bDeleted);
    int deleteItem(long oid, long uid);
    int deleteItem(char* key, long uid);
    dsk_path_t* newItemData(dsk_path_t** data);
    dsk_path_t* setItemData(dsk_path_t** data,long oid,long key_hash,long path_type,long disk_group,long path_parent,
        long create_date,long create_uid,long modify_date,long modify_uid,long is_modified,long delete_date,long delete_uid,
        long is_deleted,long block_start,long block_qty,long data_size,char* path_name,char* path_ext,char* key);
    /* Debug */
    void debug(int debugLevel, const char* className, const char* methodName);
    void debugEntry(int debugLevel, const char* className, const char* methodName, dsk_path_t* p);
    /* Getters/Setters */
    char* getDirTableFile();
    char* getDirTableFilePath();
    int getMaxNumDskPath();
    int getCurrNumDskPath();
};

/* NEW_TYPE: CDskPathMan Pointer */
typedef CDskPathMan* CDskPathManPtr_t;
```



1# int loadFile()

FILE* f = OpenFileUtil(FILMODE_READ)

while(< maxNumEntries) {

fread(arrDataTable, size, 1, f) }

2# int saveFile()

FILE* f = openFileUtil(FILMODE_WRITE)

for(< currNumEntries) {

long num_write = fwrite(arrDataTable, 1, size, f) }

3# int addItem()

if(findItem(path_parent, path_name, &data) != RSOK) {

AddNewItem... } else {

UpdateExistentItem... }

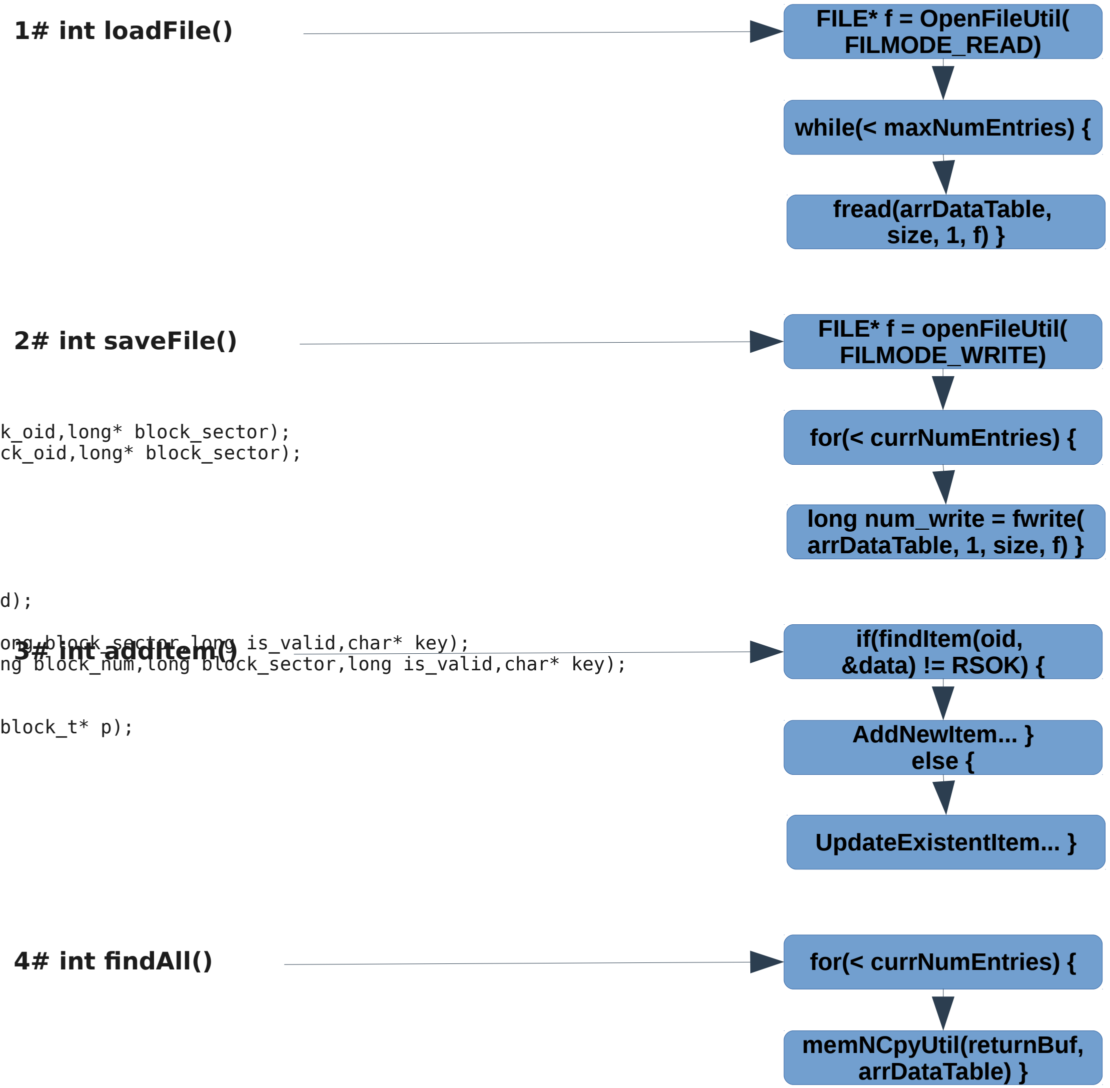
4# int findAll()

for(< currNumEntries) {

memNCpyUtil(returnBuf, arrDataTable) }

HorusWrk – Block Locator

```
class CDskBlockMan
{
private:
    bigstr_t m_blockTableFile;
    pathname_t m_blockTableFilePath;
    dsk_block_t* m_arrDskBlock;
    int m_maxNumDskBlock;
    int m_currNumDskBlock;
    // BLOCK MUTEX
    //pthread_mutex_t m_block_mutex;
public:
    CDskBlockMan(int maxNumDskBlock, char* blockTableFile, char* blockTableFilePath);
    ~CDskBlockMan();
    void init(int maxNumDskBlock, char* blockTableFile, char* blockTableFilePath);
    void terminate();
    /* Methodes */
    int loadFile(char* fileName);
    int saveFile(char* fileName);
    int addItem(long path_oid,long block_num,long block_sector,long* block_oid);
    int findItem(long oid, dsk_block_t** resval);
    int findItem(char* key, dsk_block_t** resval);
    int findFirstFreeBlock(long path_oid,long block_num,dsk_block_t** resval,long* block_oid,long* block_sector);
    int findBlockByBlockNum(long path_oid,long block_num,dsk_block_t** resval,long* block_oid,long* block_sector);
    int findAll(dsk_block_t** arr, long* arrsz);
    int findAllByPathOid(long path_oid, dsk_block_t** arr, long* arrsz);
    int countItemsByPathOid(long path_oid, long* num_items);
    int deleteItem(long oid, long uid);
    int deleteItem(char* key, long uid);
    int deleteAllItemByPathOid(long path_oid, long uid);
    int copyAllItemByPathOid(long spath_oid, long dpath_parent, long dpath_oid, long uid);
    dsk_block_t* newItemData(dsk_block_t** data);
    dsk_block_t* setItemData(dsk_block_t** data,long oid,long path_oid,long block_num,long block_sector,long is_valid,char* key);
    dsk_block_t* setItemData_NotThreadSafe(dsk_block_t** data,long oid,long path_oid,long block_num,long block_sector,long is_valid,char* key);
    /* Debug */
    void debug(int debugLevel, const char* className, const char* methodName);
    void debugEntry(int debugLevel, const char* className, const char* methodName, dsk_block_t* p);
    /* Getters/Setters */
    char* getBlockTableFile();
    int getMaxNumDskBlock();
    int getCurrNumDskBlock();
};

/* NEW_TYPE: CDskBlockMan Pointer */
typedef CDskBlockMan* CDskBlockManPtr_t;
```



1# int loadFile()

FILE* f = OpenFileUtil(FILMODE_READ)

while(< maxNumEntries) {

fread(arrDataTable, size, 1, f) }

2# int saveFile()

FILE* f = openFileUtil(FILMODE_WRITE)

for(< currNumEntries) {

long num_write = fwrite(arrDataTable, 1, size, f) }

3# int addItem()

if(findItem(oid, &data) != RSOK) {

AddNewItem... } else {

UpdateExistentItem... }

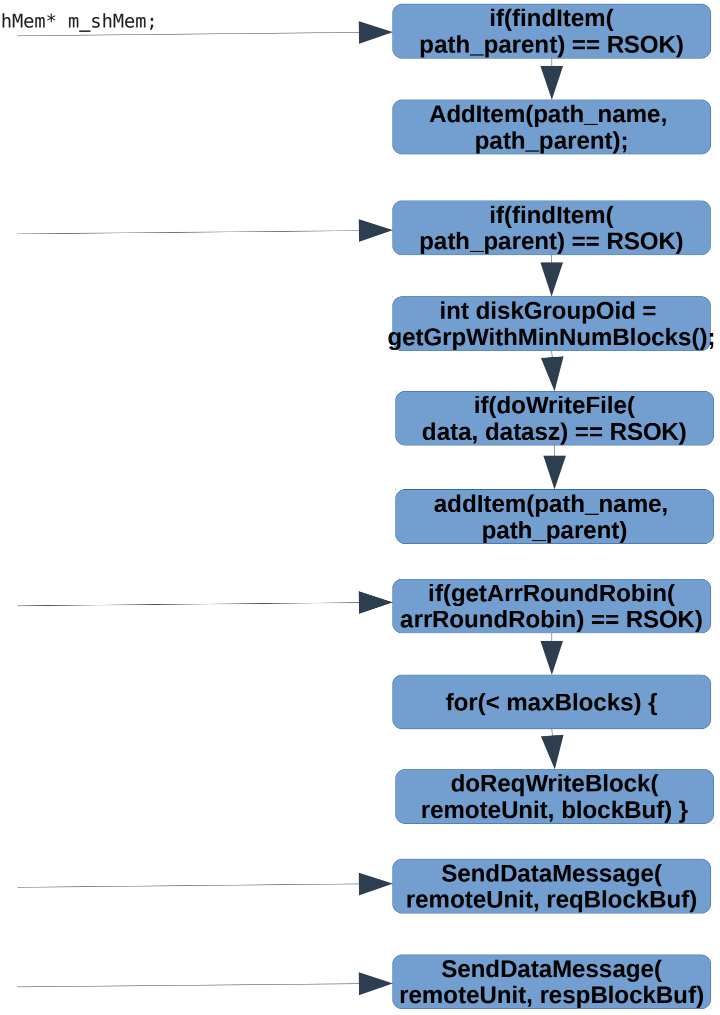
4# int findAll()

for(< currNumEntries) {

memNCpyUtil(returnBuf, arrDataTable) }

HorusWrk – Remote File Requester

```
class CDskDiskMan
{
private:
    void* m_pRemoteUnit; void* m_pDispatch; str_t m_session; bigstr_t m_dirTableFile; pathname_t m_dirTableFilePath; CDskShMem* m_shMem;
    :
public:
    CDskDiskMan(void* pRemoteUnit,char* dirTableFile,...);
    ~CDskDiskMan();
    void init(char* dirTableFile,char* dirTableFilePath,...);
    void initDiskGroup();
    void terminate();
    CDskDiskGroup* addDiskGroup(long oid,long numOfDisks,...);
    int testDiskMan_WriteLocalFile(long path_parent, long bWaitKey);
    int testDiskMan_ReadLocalFile(long path_parent, long bWaitKey);
    int testDiskMan_DeleteLocalFile(long path_parent, long bWaitKey);
    int testDiskMan_UploadRemoteFile(long path_parent, long bWaitKey);
    int testDiskMan_DownloadRemoteFile(long path_parent, long bWaitKey);
    int testDiskMan_DeleteRemoteFile(long path_parent, long bWaitKey);
    int testDiskMan_RemoveDir(long bWaitKey);
    int testDiskMan();
    int doChDir(long path_parent);
    int doMakeDir(long uid, long path_parent, char* path_name, long* path_oid);
    int doRemoveDir(long uid, long path_oid);
    int doRemoveDir(long uid, long path_parent, char* path_name);
    int doListDir(dsk_path_t** data, long* datasz, long path_parent, long bDeleted);
    int doListDir(dsk_path_t** data, long* datasz, long path_parent, char* path_name, long bDeleted);
    int showListDir(long path_parent, long bDeleted, long bWaitKey);
    int doFileExist(long path_oid, dsk_path_t* pCurrPath);
    int doFileExist(long path_parent, char* path_name, dsk_path_t* pCurrPath);
    int doDeleteFile(long uid, long path_oid);
    int doDeleteFile(long uid, long path_parent, char* path_name);
    int doCopyFile(long uid, long spath_parent, char* spath_name, long dpath_parent, char* dpath_name, long* dpath_oid);
    int doMoveFile(long uid, long spath_parent, char* spath_name, long dpath_parent, char* dpath_name);
    int doRenameFile(long uid, long spath_parent, char* spath_name, char* dpath_name);
    int doWriteFile(byte* data, long datasz, long path_parent, char* path_name, long uid, long* path_oid);
    int doWriteFileMT(byte* data, long datasz, long path_parent, char* path_name, long uid, long* path_oid);
    int doReadFile(byte** data, long* datasz, long path_parent, char* path_name, long uid, long* path_oid);
    int doReadFileMT(byte** data, long* datasz, long path_parent, char* path_name, long uid, long* path_oid);
    int doReqUploadFile(CCfgRemoteUnit* pCurrRemoteUnit,...);
    int doReqDownloadFile(CCfgRemoteUnit* pCurrRemoteUnit,...);
    int doReqDownloadFile(CCfgRemoteUnit* pCurrRemoteUnit,...);
    int doReqWriteFile(CCfgRemoteUnit* pCurrRemoteUnit,...);
    int doReqReadFile(CCfgRemoteUnit* pCurrRemoteUnit,...);
    int doReqWriteBlock(CCfgRemoteUnit* pCurrRemoteUnit,...);
    int doReqReadBlock(CCfgRemoteUnit* pRemoteUnit,...);
    int doRespUploadFile(CCfgRemoteUnit* pCurrRemoteUnit,...);
    int doRespDownloadFile(CCfgRemoteUnit* pCurrRemoteUnit,...);
    int doRespDownloadFile(CCfgRemoteUnit* pCurrRemoteUnit,...);
    int doRespWriteFile(CCfgRemoteUnit* pCurrRemoteUnit,...);
    int doRespReadFile(CCfgRemoteUnit* pCurrRemoteUnit,...);
    int doRespWriteBlock(CCfgRemoteUnit* pCurrRemoteUnit,...);
    int doRespReadBlock(CCfgRemoteUnit* pRemoteUnit,...);
};
```



1# int doMakeDir()

if(findItem(path_parent) == RSOK)

AddItem(path_name, path_parent);

2# int doUploadFile()

if(findItem(path_parent) == RSOK)

int diskGroupOid = getGrpWithMinNumBlocks();

if(doWriteFile(data, datasz) == RSOK)

addItem(path_name, path_parent)

3# int doWriteFile()

if(getArrRoundRobin(arrRoundRobin) == RSOK)

for(< maxBlocks) {

doReqWriteBlock(remoteUnit, blockBuf) }

4# int doReqWriteBlock()

SendDataMessage(remoteUnit, reqBlockBuf)

5# int doRespWriteBlock()

SendDataMessage(remoteUnit, respBlockBuf)

HorusWrk – Data Processor (Module)

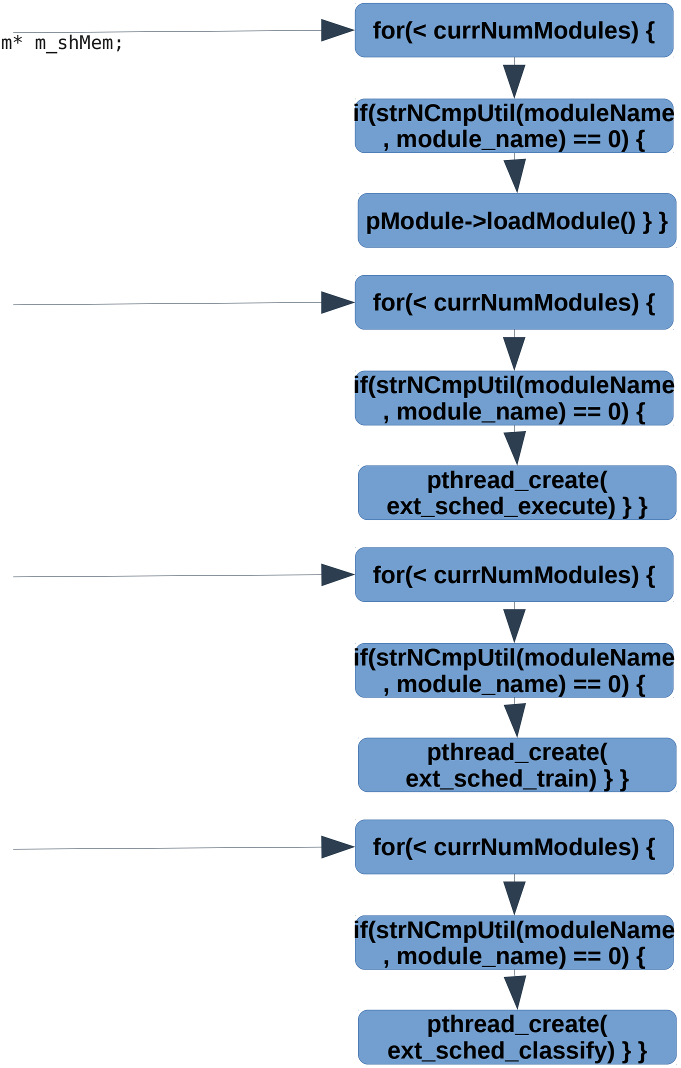
```
class CDskDiskMan
{
private:
    void* m_pRemoteUnit; void* m_pDispatch; str_t m_session; bigstr_t m_dirTableFile; pathname_t m_dirTableFilePath; CDskShMem* m_shMem;
    :
public:
    CDskDiskMan(void* pRemoteUnit,char* dirTableFile,...);
    ~CDskDiskMan();
    void init(char* dirTableFile,char* dirTableFilePath,...);
    void initDiskGroup();
    void terminate();
    CDskDiskGroup* addDiskGroup(long oid,long numOfDisks,...);
    int testDiskMan_WriteLocalFile(long path_parent, long bWaitKey);
    int testDiskMan_ReadLocalFile(long path_parent, long bWaitKey);
    int testDiskMan_DeleteLocalFile(long path_parent, long bWaitKey);
    int testDiskMan_UploadRemoteFile(long path_parent, long bWaitKey);
    int testDiskMan_DownloadRemoteFile(long path_parent, long bWaitKey);
    int testDiskMan_DeleteRemoteFile(long path_parent, long bWaitKey);
    int testDiskMan_RemoveDir(long bWaitKey);
    int testDiskMan();
    int doChDir(long path_parent);
    int doMakeDir(long uid, long path_parent, char* path_name, long* path_oid);
    int doRemoveDir(long uid, long path_oid);
    int doRemoveDir(long uid, long path_parent, char* path_name);
    int doListDir(dsk_path_t** data, long* datasz, long path_parent, long bDeleted);
    int doListDir(dsk_path_t** data, long* datasz, long path_parent, char* path_name, long bDeleted);
    int showListDir(long path_parent, long bDeleted, long bWaitKey);
    int doFileExist(long path_oid, dsk_path_t* pCurrPath);
    int doFileExist(long path_parent, char* path_name, dsk_path_t* pCurrPath);
    int doDeleteFile(long uid, long path_oid);
    int doDeleteFile(long uid, long path_parent, char* path_name);
    int doCopyFile(long uid, long spath_parent, char* spath_name, long dpath_parent, char* dpath_name, long* dpath_oid);
    int doMoveFile(long uid, long spath_parent, char* spath_name, long dpath_parent, char* dpath_name);
    int doRenameFile(long uid, long spath_parent, char* spath_name, char* dpath_name);
    int doWriteFile(byte* data, long datasz, long path_parent, char* path_name, long uid, long* path_oid);
    int doWriteFileMT(byte* data, long datasz, long path_parent, char* path_name, long uid, long* path_oid);
    int doReadFile(byte** data, long* datasz, long path_parent, char* path_name, long uid, long* path_oid);
    int doReadFileMT(byte** data, long* datasz, long path_parent, char* path_name, long uid, long* path_oid);
    int doReqUploadFile(CCfgRemoteUnit* pCurrRemoteUnit,...);
    int doReqDownloadFile(CCfgRemoteUnit* pCurrRemoteUnit,...);
    int doReqDownloadFile(CCfgRemoteUnit* pCurrRemoteUnit,...);
    int doReqWriteFile(CCfgRemoteUnit* pCurrRemoteUnit,...);
    int doReqReadFile(CCfgRemoteUnit* pCurrRemoteUnit,...);
    int doReqWriteBlock(CCfgRemoteUnit* pCurrRemoteUnit,...);
    int doReqReadBlock(CCfgRemoteUnit* pRemoteUnit,...);
    int doRespUploadFile(CCfgRemoteUnit* pCurrRemoteUnit,...);
    int doRespDownloadFile(CCfgRemoteUnit* pCurrRemoteUnit,...);
    int doRespDownloadFile(CCfgRemoteUnit* pCurrRemoteUnit,...);
    int doRespWriteFile(CCfgRemoteUnit* pCurrRemoteUnit,...);
    int doRespReadFile(CCfgRemoteUnit* pCurrRemoteUnit,...);
    int doRespWriteBlock(CCfgRemoteUnit* pCurrRemoteUnit,...);
    int doRespReadBlock(CCfgRemoteUnit* pRemoteUnit,...);
};
```

1# void loadModule()

2# void execute()

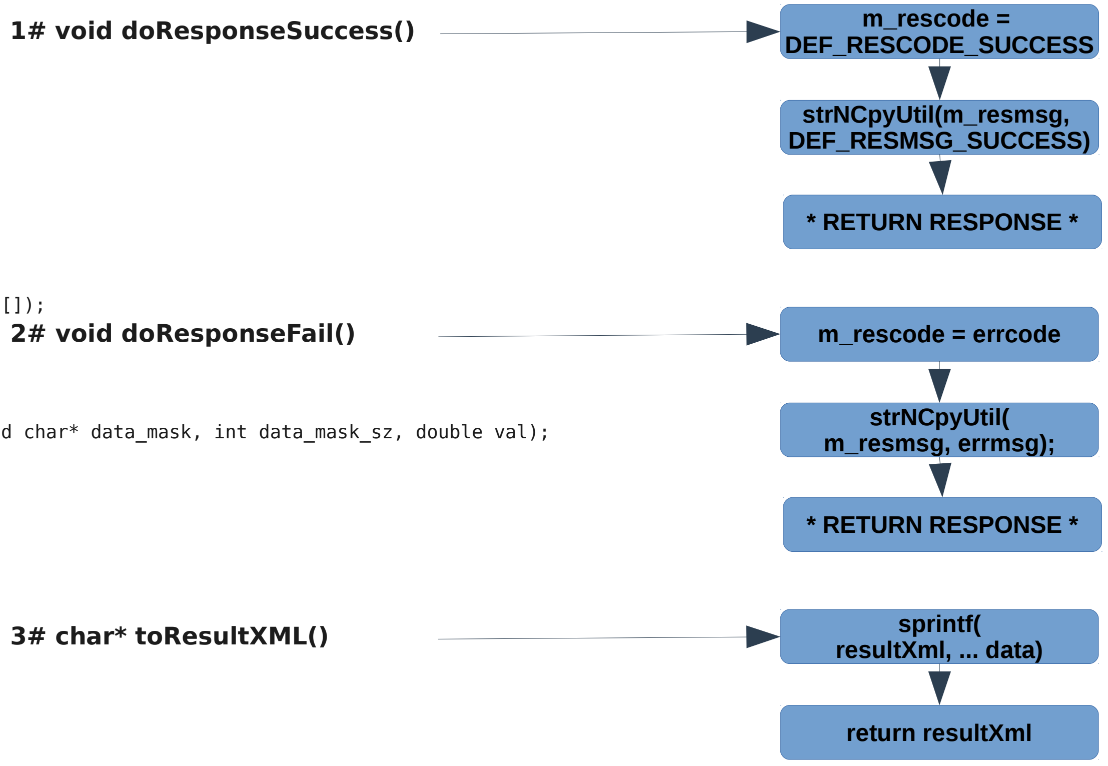
4# int trainModel()

5# int classify()


for(< currNumModules) {
if(strNCmpUtil(moduleName, module_name) == 0) {
pModule->loadModule() } }
for(< currNumModules) {
if(strNCmpUtil(moduleName, module_name) == 0) {
pthread_create(ext_sched_execute) } }
for(< currNumModules) {
if(strNCmpUtil(moduleName, module_name) == 0) {
pthread_create(ext_sched_train) } }
for(< currNumModules) {
if(strNCmpUtil(moduleName, module_name) == 0) {
pthread_create(ext_sched_classify) } }

HorusWrk – Return Data Results

```
class CExtSched
{
private:
    sched_t* m_pLsModules;
    int m_numMaxModules;
    int m_currNumModules;
    int m_numLoadedModules;
    int m_numRunningModules;
public:
    CExtSched(int numMaxModules);
    ~CExtSched();
    /* Methodes */
    int init();
    /* Module Operations */
    int loadModule(char* module_name);
    int initModule(char* module_name, int numParms, char* parmNames[], char* parmValues[]);
    int terminateModule(char* module_name);
    int start(char* module_name);
    int execute(char* module_name);
    int sleep(char* module_name);
    int trainModel(char* module_name, unsigned char* data, int data_sz, double val);
    int trainModelWithMask(char* module_name, unsigned char* data, int data_sz, unsigned char* data_mask, int data_mask_sz, double val);
    int classify(char* module_name, unsigned char* data, int data_sz, double* resval);
    int saveModel(char* module_name, char* model_name, char* model_file_name);
    int loadModel(char* module_name, char* model_name, char* model_file_name);
    /* Getters/Setters */
    sched_t* getSchedByModuleOid(long oid);
    sched_t* getSchedByModuleName(char* moduleName);
    // Management Of Sched List
    int getCurrNumModules();
    int getNumLoadedModules();
    int getNumRunningModules();
};

/* NEW_TYPE: CExtSched Pointer */
typedef CExtSched* CExtSchedPtr_t;

/* Extension Sched Global Declaration */
extern CExtSched gExtSched;

/* Extension Sched Threads */
extern void* ext_sched_start(void* _pArgs);
extern void* ext_sched_execute(void* _pArgs);
extern void* ext_sched_train(void* _pArgs);
extern void* ext_sched_train_with_mask(void* _pArgs);
extern void* ext_sched_classify(void* _pArgs);
```



1# void doResponseSuccess()

m_rescode = DEF_RESCODE_SUCCESS

strNCpyUtil(m_resmsg, DEF_RESMSG_SUCCESS)

* RETURN RESPONSE *

2# void doResponseFail()

m_rescode = errcode

strNCpyUtil(m_resmsg, errmsg);

* RETURN RESPONSE *

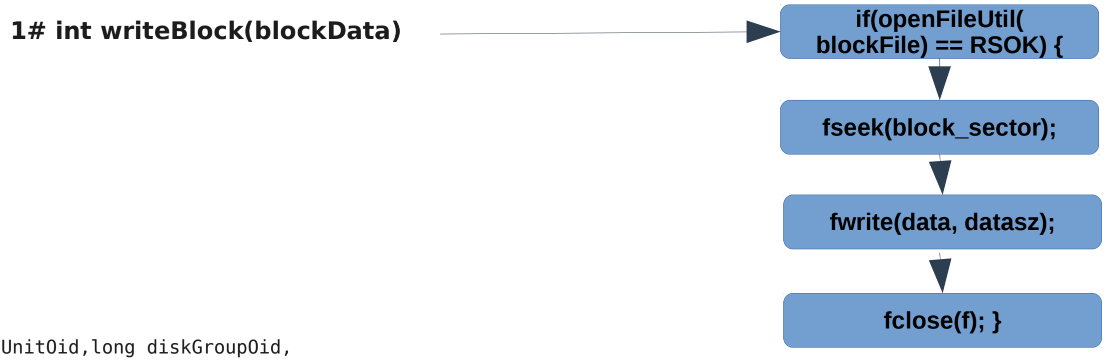
3# char* toResultXML()

sprintf(resultXml, ... data)

return resultXml

HorusWrk – File Writer

```
class CDskDiskDev
{
private:
    void* m_pRemoteUnit;                // point to remote disk device unit
    void* m_pDiskGroup;                 // point to disk group
    long m_oid;                         // disk device unit id
    long m_posDiskDev;                  // disk device pos
    long m_remoteUnitOid;               // remote disk device unit id
    long m_diskGroupOid;                // disk group oid
    long m_lastUpdate;                  // last disk device initialization
    str_t m_name;                       // disk device name
    str_t m_blockTableFile;             // superblock table file
    str_t m_dataFile;                   // data file
    pathname_t m_blockTableFilePath;    // superblock table file path
    pathname_t m_dataFilePath;          // data file path
    CDskBlockMan* m_blockMan;           // point to disk blocks manager
public:
    CDskDiskDev(void* pRemoteUnit,void* pDiskGroup,long oid,long posDiskDev,long remoteUnitOid,long diskGroupOid,
        long lastUpdate,char* name,char* blockTableFile,char* dataFile,char* blockTableFilePath,char* dataFilePath);
    ~CDskDiskDev();
    void init(void* pRemoteUnit,void* pDiskGroup,long oid,long posDiskDev,long remoteUnitOid,long diskGroupOid,
        long lastUpdate,char* name,char* blockTableFile,char* dataFile,char* blockTableFilePath,char* dataFilePath);
    void terminate();
    /* Methodes */
    int writeBlock(long path_oid,long block_num,long block_size,byte* blockbuf,dsk_block_t** p_block,long* block_oid,long* block_sector);
    int readBlock(long path_oid,long block_num,long block_size,byte* blockbuf,dsk_block_t** p_block,long* block_oid,long* block_sector);
    /* Getters/Setters */
    void* getRemoteUnitPtr();
    void* getDiskGroupPtr();
    long getOid();
    long getPosDiskDev();
    long getRemoteUnitOid();
    long getDiskGroupOid();
    long getLastUpdate();
    char* getName();
    char* getBlockTableFile();
    char* getDataFile();
    char* getBlockTableFilePath();
    char* getDataFilePath();
    CDskBlockMan* getDskBlockMan();
};
/* NEW_TYPE: CDskDiskDev Pointer */
typedef CDskDiskDev* CDskDiskDevPtr_t;
```



1# int writeBlock(blockData)

```
if(openFileUtil(
blockFile) == RSOK) {
fseek(block_sector);
fwrite(data, datasz);
fclose(f); }
```

Dúvidas

?