ss_Servicecore

1.0

Generated by Doxygen 1.7.2

Wed Dec 22 2010 12:50:37

Contents

Chapter 1

Todo List

Member awaitingPattern_mode Need to be deleted.

Member dataFile[255] Need to be deleted.

Member dbg_hex_print(BYTE ∗buffer, size_t len) use this function in new debug print system

Member deqUdp::readData(ssBuffer ∗addr, size_t ∗len, in_addr ∗ip_from=0, bool peek=false) not needed to realize all sendData check ability to use base method sendData of parent class

Member deqUdp::sendData(ssBuffer ∗buf) not needed to realize all sendData check ability to use base method sendData of parent class

Member equipListenPolling(void ∗user) Listening equipment answer - status vector:

Member equipListenPolling(void ∗) add to commonFuncsMgr class as static method

Member fileRead(char ∗fname, BYTE ∗∗buffer, size_t ∗sz) reorganize function to reading xml-files for future purposes

Member if_name[255] Need to be deleted.

Member listen_mode Need to be deleted.

Member pattern_found Need to be deleted.

Member patternFile[255] Need to be deleted.

Member process_cmdLine(int argc, char *argv[]) reorganize process to external library

Member reactionFile[255] Need to be deleted.

Member srvAppLayer::encodeBlock(rcsCmd *, BYTE **) nobody uses this method. need to be deleted.

Member srvAppLayer::encodeFuncResult(rcsCmd *in_cmd, rcsCmd *out_cmd) Need a refactoring.

Member srvAppLayer::equip_read_data(BYTE *, size_t *) too strange method.

Member srvAppLayer::equip_reading_event() too strange method.

Member srvAppLayer::execMessage(rcsCmd *ss_cmd) ServiceState vector need to change before function calling.

Member srvAppLayer::Functions[100] array not the best data structure for this purposes.

Chapter 2

Directory Hierarchy

2.1 Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

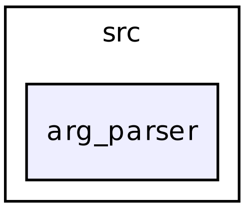
Here is a list of all files with brief descriptions:

Chapter 5

Directory Documentation

5.1 src/arg_parser/ Directory Reference

Directory dependency graph for src/arg_parser/:

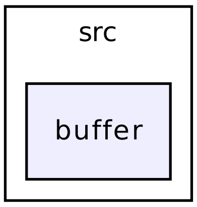


Files

- file carg_parser.cpp
- file carg_parser.h

5.2 src/buffer/ Directory Reference

Directory dependency graph for src/buffer/:



Files

- file buffer.cpp

    Class buffer implementation.

- file buffer.h

    Class buffer interface header.

- file ssBuffer.cpp

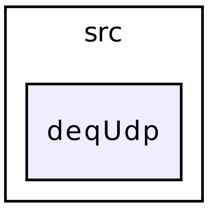
    Class ssBuffer implementation.

- file ssBuffer.h

    Class ssBuffer interface header.

5.3 src/deqUdp/ Directory Reference

Directory dependency graph for src/deqUdp/:



Files

- file deqUdp.cpp

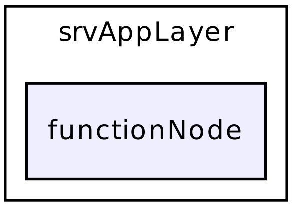
  Class deqUdp implementation.

- file deqUdp.h

  Class deqUdp interface header.

5.4 src/srvAppLayer/functionNode/ Directory Reference

Directory dependency graph for src/srvAppLayer/functionNode/:



Files

- file functionNode.cpp

  Class functionNode implementation.

- file functionNode.h

  Class functionNode interface header.

- file param_desc.cpp

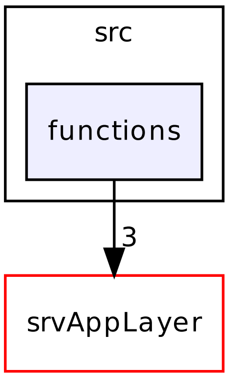
  Class param_desc implementation.

- file param_desc.h

  Class param_desc interface header.

5.5 src/functions/ Directory Reference

Directory dependency graph for src/functions/:



Files

- file commonFuncsMgr.cpp

  Class commonFuncsMgr interface header.

- file commonFuncsMgr.h

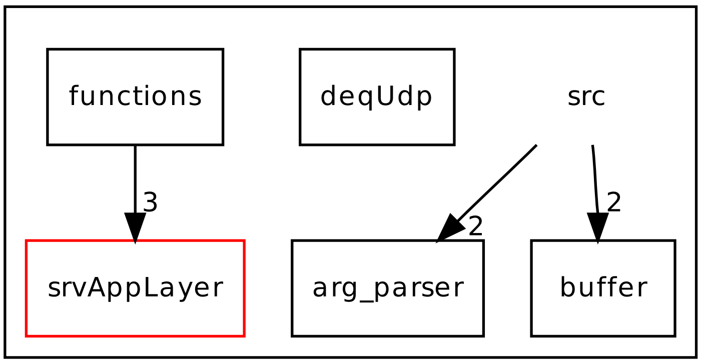
  Class commonFuncsMgr interface header.

- file specFuncsMgr.h

  Class specFuncsMgr interface header.

5.6 src/ Directory Reference

Directory dependency graph for src/:



Directories

- directory arg_parser
- directory buffer
- directory deqUdp
- directory functions
- directory srvAppLayer

Files

- file _auto_config.h
- file _global.cpp

  Global environment.

- file _global.h

  Global environment interface header.

- file auto_config.h
- file config.h
- file console_out.cpp

  aided functions to process_cmdLine

- file console_out.h

  aided functions interface header

- file main.cpp

  Programm entry point.

- file SIG_handler.cpp

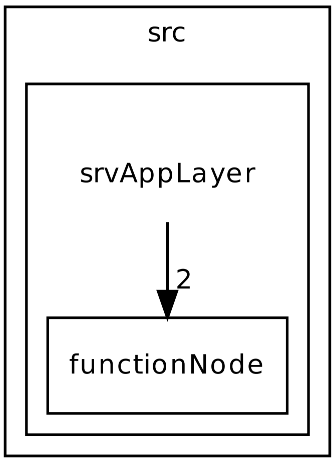
  System signals handlers manager.

- file SIG_handler.h

  System signals handlers manager interface header.

5.7 src/srvAppLayer/ Directory Reference

Directory dependency graph for src/srvAppLayer/:



Directories

- directory functionNode

Files

- file srvAppLayer.cpp

  Class srvAppLayer implementation.

- file srvAppLayer.h

  Class srvAppLayer interface header.

Chapter 6

Class Documentation

6.1 ap_Option Struct Reference

```
#include <carg_parser.h>
```

Public Attributes

- int code
- const char * name
- ap_Has_arg has_arg

6.1.1 Detailed Description

Definition at line 44 of file carg_parser.h.

6.1.2 Member Data Documentation

6.1.2.1 int ap_Option::code

Definition at line 46 of file carg_parser.h.

Referenced by optname(), parse_long_option(), and parse_short_option().

6.1.2.2 const char* ap_Option::name

Definition at line 47 of file carg_parser.h.

Referenced by optname().

6.1.2.3 ap_Has_arg ap_Option::has_arg

Definition at line 48 of file carg_parser.h.

The documentation for this struct was generated from the following file:

- src/arg_parser/carg_parser.h

6.2 ap_Record Struct Reference

```
#include <carg_parser.h>
```

Public Attributes

- int code
- char ∗ argument

6.2.1 Detailed Description

Definition at line 53 of file carg_parser.h.

6.2.2 Member Data Documentation

6.2.2.1 int ap_Record::code

Definition at line 55 of file carg_parser.h.

Referenced by ap_code(), and push_back_record().

6.2.2.2 char∗ ap_Record::argument

Definition at line 56 of file carg_parser.h.

Referenced by ap_argument(), free_data(), and push_back_record().

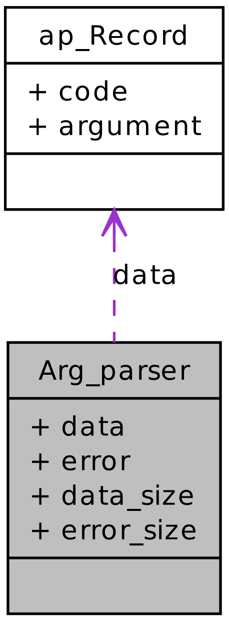
The documentation for this struct was generated from the following file:

- src/arg_parser/carg_parser.h

6.3 Arg_parser Struct Reference

```
#include <carg_parser.h>
```

Collaboration diagram for Arg_parser:



Public Attributes

- ap_Record ∗ data
- char ∗ error
- int data_size
- int error_size

6.3.1 Detailed Description

Definition at line 61 of file carg_parser.h.

6.3.2 Member Data Documentation

6.3.2.1 ap_Record∗ Arg_parser::data

Definition at line 63 of file carg_parser.h.

Referenced by ap_argument(), ap_code(), ap_init(), free_data(), and push_back_record().

6.3.2.2 char∗ Arg_parser::error

Definition at line 64 of file carg_parser.h.

Referenced by add_error(), ap_error(), ap_free(), and ap_init().

6.3.2.3 int Arg_parser::data_size

Definition at line 65 of file carg_parser.h.

Referenced by ap_arguments(), ap_init(), free_data(), and push_back_record().

6.3.2.4 int Arg_parser::error_size

Definition at line 66 of file carg_parser.h.

Referenced by add_error(), ap_free(), and ap_init().

The documentation for this struct was generated from the following file:

- src/arg_parser/carg_parser.h

6.4 buffer Class Reference

Simple queue of bytes.

```
#include <buffer.h>
```

Public Member Functions

- buffer (DWORD size)

  *Constructs buffer as bytes array of **size**.*

- ∼buffer ()
- errType write (BYTE ∗addr, DWORD len)

  *Try to write to array data from **addr** pointer with **len** bytes size.*

- DWORD read (BYTE ∗addr, DWORD len=0)

  *Try to read from array data to **addr** pointer with **len** bytes size.*

- DWORD length ()

  Returns length of stored data.

- BYTE ∗ lockBufferChunkForExternWriting ()

  Allow direct mode writing for external methods.

- errType unlockBufferChunkForExternWriting (DWORD offset)

Disable direct mode writing from external methods.

- errType removeBufferChunk (DWORD backward_offset, DWORD len)

  Remove data from buffer.

- errType copyBufferChunkTo (BYTE ∗addr, DWORD offset=0, DWORD len=0)

  Try to read from array ***offset*** *data to* ***addr*** *pointer with* ***len*** *bytes size.*

- errType dbgPrint ()

  Print for debug purposes contents of stored buffer (in a hexadecimal notation)

Private Attributes

- BYTE ∗ buff
- DWORD WrRef
- DWORD RdRef
- DWORD buffSize
- bool writingLock

6.4.1 Detailed Description

Simple queue of bytes.

Definition at line 20 of file buffer.h.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 buffer::buffer (DWORD *size*)

Constructs buffer as bytes array of **size**.

Definition at line 23 of file buffer.cpp.

References buff, buffSize, RdRef, writingLock, and WrRef.

6.4.2.2 buffer::∼buffer ( )

Definition at line 33 of file buffer.cpp.

References buff, RdRef, writingLock, and WrRef.

6.4.3 Member Function Documentation

6.4.3.1 errType buffer::write (BYTE ∗ *addr,* DWORD *len*)

Try to write to array data from **addr** pointer with **len** bytes size.

Return values

err_result_ok	- writing was succeeded
err_result_error	- not enough space. Data was written partially.

Definition at line 58 of file buffer.cpp.

References buff, buffSize, and WrRef.

6.4.3.2 DWORD buffer::read (BYTE * *addr*, DWORD *len* = 0)

Try to read from array data to **addr** pointer with **len** bytes size.

If **len** not specified (or equal to 0) method reads all array.

Readed data has been deleted in buffer.

See also

buffer::copyBufferChunkTo

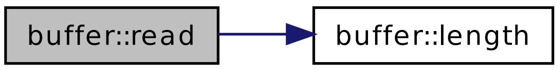
Return values

length	- size in bytes of readed data.

Definition at line 81 of file buffer.cpp.

References buff, length(), RdRef, writingLock, and WrRef.

Here is the call graph for this function:



6.4.3.3 DWORD buffer::length ( )

Returns length of stored data.

length is writing reference index minus reading reference index

Return values

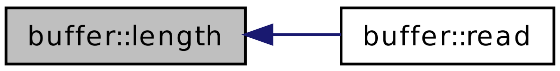
length	- stored data length in bytes

Definition at line 48 of file buffer.cpp.

References RdRef, and WrRef.

Referenced by read().

Here is the caller graph for this function:



6.4.3.4 BYTE $*$ buffer::lockBufferChunkForExternWriting ( )

Allow direct mode writing for external methods.

While direct mode writing is locking array - nobody can't to write.

See also

Unlock with buffer::unlockBufferChunkForExternWriting.

Return values

ptr	- pointer to array for writing data

Definition at line 116 of file buffer.cpp.

References buff, writingLock, and WrRef.

6.4.3.5 errType buffer::unlockBufferChunkForExternWriting (DWORD *offset*)

Disable direct mode writing from external methods.

Parameters

`in`	*offset*	- length of writing bytes by external methods

Return values

err_result_ok	- unlocking was successfull
err_result_error	- writing bytes was more than space in buffer. Data unlocked to buffer partially.

Definition at line 131 of file buffer.cpp.

References buffSize, RdRef, writingLock, and WrRef.

6.4.3.6 errType buffer::removeBufferChunk (DWORD *backward_offset,* DWORD *len*)

Remove data from buffer.

Removes data from buffer array, backward writing reference to **backward_offset** and move data from **backward_offset** + **len** to current writing referense position.

Return values

err_result_ok	- removing was successful.
err_result_error	- cannot remove data length more than data size in buffer

Definition at line 153 of file buffer.cpp.

References buff, RdRef, and WrRef.

6.4.3.7 errType buffer::copyBufferChunkTo (BYTE ∗ *addr,* DWORD *offset* = 0, DWORD *len* = 0)

Try to read from array **offset** data to **addr** pointer with **len** bytes size.

Extract buffer data without deleting it in buffer.

See also

buffer::read

Return values

err_result_ok	- successfully copied.
err_result_error	- start of block (**offset**) is greater than stored data length.

Definition at line 175 of file buffer.cpp.

References buff, RdRef, and WrRef.

6.4.3.8 errType buffer::dbgPrint ( )

Print for debug purposes contents of stored buffer (in a hexadecimal notation)

Definition at line 190 of file buffer.cpp.

References buff, RdRef, and WrRef.

6.4.4 Member Data Documentation

6.4.4.1 BYTE∗ buffer::buff `[private]`

Definition at line 21 of file buffer.h.

Referenced by buffer(), copyBufferChunkTo(), dbgPrint(), lockBufferChunkForExternWriting(), read(), removeBufferChunk(), write(), and ~buffer().

6.4.4.2 DWORD buffer::WrRef `[private]`

Definition at line 22 of file buffer.h.

Referenced by buffer(), copyBufferChunkTo(), dbgPrint(), length(), lockBufferChunkForExternWriting(), read(), removeBufferChunk(), unlockBufferChunkForExternWriting(), write(), and ~buffer().

6.4.4.3 DWORD buffer::RdRef `[private]`

Definition at line 22 of file buffer.h.

Referenced by buffer(), copyBufferChunkTo(), dbgPrint(), length(), read(), removeBufferChunk(), unlockBufferChunkForExternWriting(), and ~buffer().

6.4.4.4 DWORD buffer::buffSize `[private]`

Definition at line 22 of file buffer.h.

Referenced by buffer(), unlockBufferChunkForExternWriting(), and write().

6.4.4.5 bool buffer::writingLock `[private]`

Definition at line 24 of file buffer.h.

Referenced by buffer(), lockBufferChunkForExternWriting(), read(), unlockBufferChunkForExternWriting(), and ~buffer().

The documentation for this class was generated from the following files:

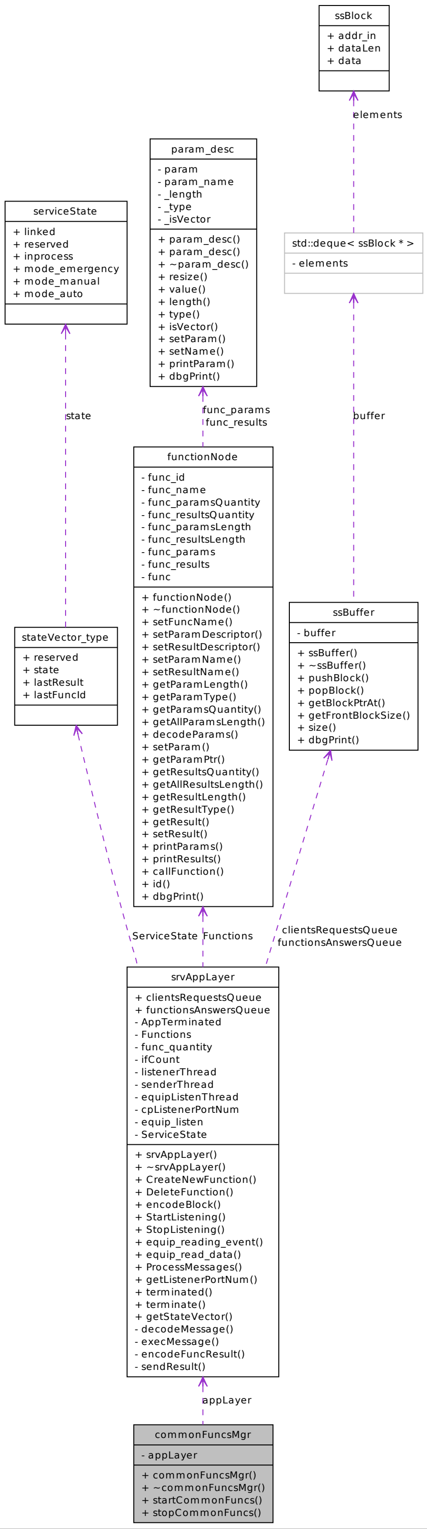
- src/buffer/buffer.h
- src/buffer/buffer.cpp

6.5 commonFuncsMgr Class Reference

common functions manager implementation (set of function independs on target)

`#include <commonFuncsMgr.h>`

Collaboration diagram for commonFuncsMgr:



Public Member Functions

- commonFuncsMgr (srvAppLayer ∗appl)
- ∼commonFuncsMgr ()
- errType startCommonFuncs ()

  Declaration of common functions.
- errType stopCommonFuncs ()

  Undeclaration of common functions.

Private Attributes

- srvAppLayer ∗ appLayer

6.5.1 Detailed Description

common functions manager implementation (set of function independs on target)

Definition at line 23 of file commonFuncsMgr.h.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 commonFuncsMgr::commonFuncsMgr (srvAppLayer ∗ *appl*)

Definition at line 60 of file commonFuncsMgr.cpp.

References appLayer.

6.5.2.2 commonFuncsMgr::∼commonFuncsMgr ( )

Definition at line 65 of file commonFuncsMgr.cpp.

6.5.3 Member Function Documentation

6.5.3.1 errType commonFuncsMgr::startCommonFuncs ( )

Declaration of common functions.

Declares functions:

- emergencyShutdown, controlModeChange, getStateVector

Return values

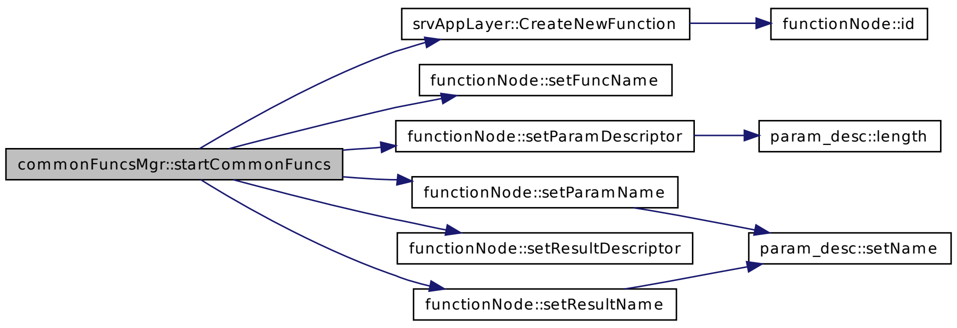
err_result_ok	- declaration was successful

Definition at line 76 of file commonFuncsMgr.cpp.

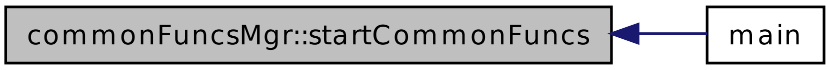
References appLayer, srvAppLayer::CreateNewFunction(), functionNode::setFuncName(), functionNode::setParamDescriptor(), functionNode::setParamName(), functionNode::setResultDescriptor(), and functionNode::setResultName().

Referenced by main().

Here is the call graph for this function:


srvAppLayer::CreateNewFunction
functionNode::id
functionNode::setFuncName
functionNode::setParamDescriptor
param_desc::length
commonFuncsMgr::startCommonFuncs
functionNode::setParamName
functionNode::setResultDescriptor
param_desc::setName
functionNode::setResultName

Here is the caller graph for this function:



6.5.3.2 errType commonFuncsMgr::stopCommonFuncs ( )

Undeclaration of common functions.

Undeclares functions:

- emergencyShutdown, controlModeChange, getStateVector

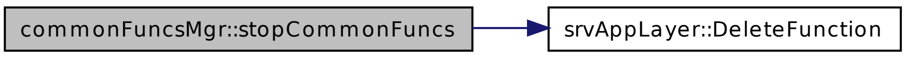
Return values

err_result_ok	- undeclaration was successful

Definition at line 151 of file commonFuncsMgr.cpp.

References appLayer, and srvAppLayer::DeleteFunction().

Here is the call graph for this function:



6.5.4 Member Data Documentation

6.5.4.1 srvAppLayer∗ commonFuncsMgr::appLayer `[private]`

Definition at line 25 of file commonFuncsMgr.h.

Referenced by commonFuncsMgr(), startCommonFuncs(), and stopCommonFuncs().

The documentation for this class was generated from the following files:

- src/functions/commonFuncsMgr.h
- src/functions/commonFuncsMgr.cpp

6.6 deqUdp Class Reference

udp communications (based on udp_port) with queues for listening and sending

`#include <deqUdp.h>`

Public Member Functions

- deqUdp (WORD portNum, const char ∗ip_str="127.0.0.1")
- ∼deqUdp ()
- errType sendData (ssBuffer ∗buf)

  Send data to udp port from queued buffer (ssBuffer)

- errType readData (ssBuffer ∗addr, size_t ∗len, in_addr ∗ip_from=0, bool peek=false)

  Read udp port data to queued buffer (ssBuffer)

6.6.1 Detailed Description

udp communications (based on udp_port) with queues for listening and sending

Definition at line 20 of file deqUdp.h.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 deqUdp::deqUdp (WORD *portNum*, const char ∗ *ip_str* = "127.0.0.1") [inline]

Definition at line 27 of file deqUdp.h.

6.6.2.2 deqUdp::∼deqUdp ( )

Definition at line 28 of file deqUdp.cpp.

6.6.3 Member Function Documentation

6.6.3.1 errType deqUdp::sendData (ssBuffer ∗ *buf*)

Send data to udp port from queued buffer (ssBuffer)

Parameters

in	*buf*	- pointer to queue for sending data

Return values

err_result_ok	- execution was successful
err_result_error	- problems with udp sendto function

Todo

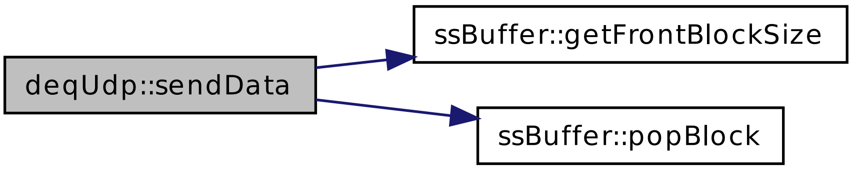
not needed to realize all sendData check ability to use base method sendData of parent class

Definition at line 39 of file deqUdp.cpp.

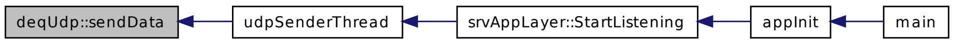
References ssBuffer::getFrontBlockSize(), and ssBuffer::popBlock().

Referenced by udpSenderThread().

Here is the call graph for this function:



Here is the caller graph for this function:



6.6.3.2 errType deqUdp::readData (ssBuffer ∗ *addr*, size_t ∗ *len*, in_addr ∗ *ipaddr* = `0`, bool *peek* = `false`)

Read udp port data to queued buffer (ssBuffer)

Parameters

`in`	*addr*	- pointer to queue for readed data
`in`	*peek*	- PEEK MODE
`out`	*len*	- length of readed data
`out`	*ipaddr*	- information about data sender

Return values

err_result_ok	- execution was successful
err_result_error	- problems with udp sendto function

Todo

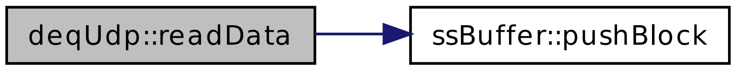
not needed to realize all sendData check ability to use base method sendData of parent class

Definition at line 77 of file deqUdp.cpp.

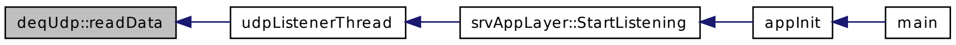
References ssBuffer::pushBlock().

Referenced by udpListenerThread().

Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this class was generated from the following files:

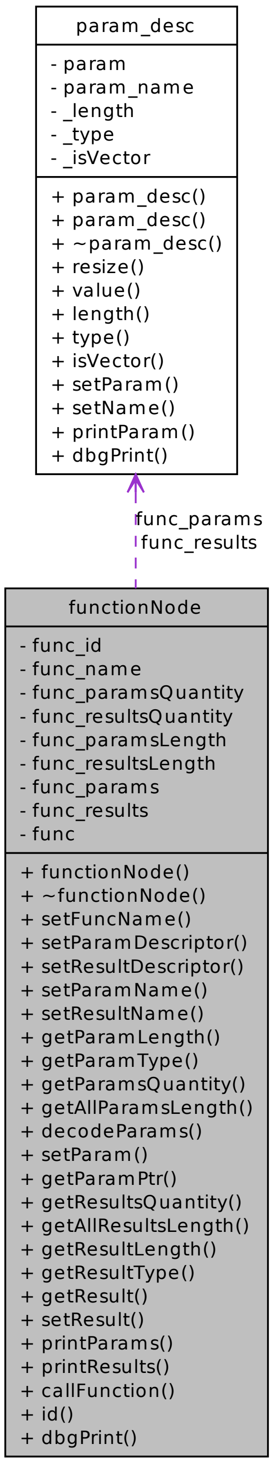
- src/deqUdp/deqUdp.h
- src/deqUdp/deqUdp.cpp

6.7 functionNode Class Reference

function node interface header

```
#include <functionNode.h>
```

Collaboration diagram for functionNode:



Public Member Functions

- functionNode (BYTE id, WORD parQnt, WORD resQnt, funcPtr ptr)

  functionNode constructor.
- ∼functionNode ()
- errType setFuncName (const char ∗name)

  Setter for function node string name.
- errType setParamDescriptor (BYTE num, OrtsType type)

  Method declares parameter for function node.
- errType setResultDescriptor (BYTE num, OrtsType type)

  Method declares results for function node.
- errType setParamName (BYTE num, const char ∗name)

  Setter for function node parameter string name.
- errType setResultName (BYTE num, const char ∗name)

  Setter for function node result string name.
- WORD getParamLength (BYTE num)

  Method evaluate function node parameter size.
- OrtsType getParamType (BYTE num)
- WORD getParamsQuantity ()

  Getter for function parameters quantity.
- WORD getAllParamsLength ()

  Method calculate sum of all function node parameters size.
- errType decodeParams (rcsCmd ∗)

  Received message parameters decode.
- errType setParam (BYTE num, void ∗param)

  Set value for declared parameter.
- void ∗ getParamPtr (BYTE num)

  Get value pointer of declared parameter.
- BYTE getResultsQuantity ()

  Getter for function results quantity.
- DWORD getAllResultsLength ()

  Method calculate sum of all function node results size.

- DWORD getResultLength (BYTE i)

  Method evaluate function node result size.

- OrtsType getResultType (BYTE num)
- errType getResult (BYTE num, void ∗∗result, DWORD ∗length)

  Getter function for function node result storage.

- errType setResult (BYTE num, void ∗result)

  Setter function for function node result storage.

- errType printParams ()

  Prints params values of function node for debug purposes.

- errType printResults ()

  Prints results values of function node for debug purposes.

- errType callFunction ()

  Call function node pointer to execution code.

- BYTE id ()

  Getter for function node identifier.

- void dbgPrint ()

  Print for debug purposes contents of function node (in a hexadecimal notation)

Private Attributes

- BYTE func_id

  function idintifier.

- const char ∗ func_name

  function name. For debug printing purposes.

- WORD func_paramsQuantity

  function declaration. Quantity of params.

- WORD func_resultsQuantity

  function declaration. Quantity of results.

- WORD func_paramsLength

  Size in bytes of all function parameters.

- WORD func_resultsLength

  Size in bytes of all function reaults.

- param_desc ∗ func_params [32]

  function parameters declarations. 32 parameters at maximum.

- param_desc ∗ func_results [32]

  function results declaration. 32 results at maximum.

- funcPtr func

  pointer to a function execution code.

6.7.1 Detailed Description

function node interface header function node is a service function that will call by srvAppLayer instance in accordance with client request

Definition at line 20 of file functionNode.h.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 functionNode::functionNode (BYTE *id*, WORD *parQnt*, WORD *resQnt*, funcPtr *ptr*)

functionNode constructor.

uses to declare service function.

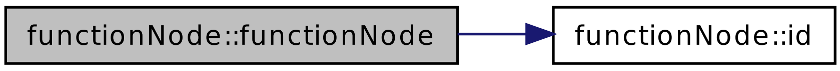
Parameters

in	*id*	- function identifier.
in	*parQnt*	- quantity of function parameters.
in	*resQnt*	- quantity of function results.
in	*ptr*	- function execution code pointer.

Definition at line 29 of file functionNode.cpp.

References func, func_id, func_params, func_paramsLength, func_paramsQuantity, func_resultsLength, func_resultsQuantity, and id().

Here is the call graph for this function:



6.7.2.2 functionNode::∼functionNode ( )

Definition at line 42 of file functionNode.cpp.

References func_params, func_paramsQuantity, func_results, and func_resultsQuantity.

6.7.3 Member Function Documentation

6.7.3.1 errType functionNode::setFuncName (const char ∗ *name*)

Setter for function node string name.

Parameters

name	- name of a function node.

Returns

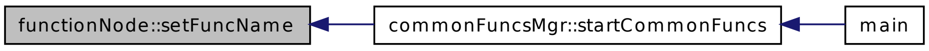
err_result_ok

Definition at line 362 of file functionNode.cpp.

References func_name.

Referenced by commonFuncsMgr::startCommonFuncs().

Here is the caller graph for this function:



6.7.3.2 errType functionNode::setParamDescriptor (BYTE *num*, OrtsType *type*)

Method declares parameter for function node.

Parameters

num	- parameter index (zero based index)
type	- parameter type with respect to OrtsType enumeration

Returns

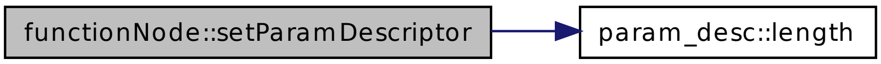
err_result_ok - declaring successful

Definition at line 61 of file functionNode.cpp.

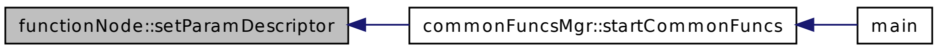
References func_params, func_paramsLength, func_paramsQuantity, and param_desc::length().

Referenced by commonFuncsMgr::startCommonFuncs().

Here is the call graph for this function:



Here is the caller graph for this function:


functionNode::setParamDescriptor
commonFuncsMgr::startCommonFuncs
main

6.7.3.3 errType functionNode::setResultDescriptor (BYTE *num*, OrtsType *type*)

Method declares results for function node.

Parameters

num	- result index (zero based index)
type	- result type with respect to OrtsType enumeration

Returns

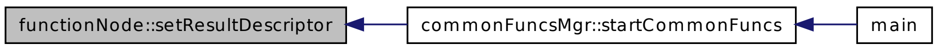
err_result_ok - declaring successful err_not_init - declaring unsuccessful

Definition at line 83 of file functionNode.cpp.

References func_results.

Referenced by commonFuncsMgr::startCommonFuncs().

Here is the caller graph for this function:


functionNode::setResultDescriptor
commonFuncsMgr::startCommonFuncs
main

6.7.3.4 errType functionNode::setParamName (BYTE *num*, const char ∗ *name*)

Setter for function node parameter string name.

Parameters

num	- number of a parameter
name	- name of a parameter

Returns

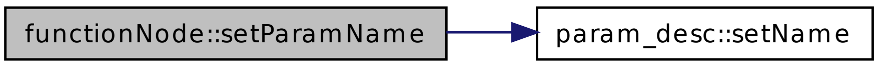
err_result_ok

Definition at line 377 of file functionNode.cpp.

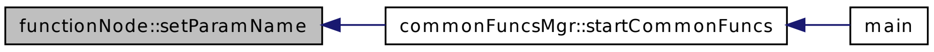
References func_params, and param_desc::setName().

Referenced by commonFuncsMgr::startCommonFuncs().

Here is the call graph for this function:



Here is the caller graph for this function:



6.7.3.5 errType functionNode::setResultName (BYTE *num*, const char ∗ *name*)

Setter for function node result string name.

Parameters

num	- number of a result
name	- name of a result

Returns

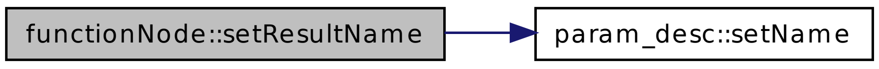
err_result_ok

Definition at line 392 of file functionNode.cpp.

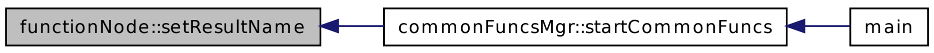
References func_results, and param_desc::setName().

Referenced by commonFuncsMgr::startCommonFuncs().

Here is the call graph for this function:



Here is the caller graph for this function:



6.7.3.6 WORD functionNode::getParamLength (BYTE *num*)

Method evaluate function node parameter size.

Parameters

num	- parameter index (zero based index)

Returns

size of parameter in bytes

Definition at line 105 of file functionNode.cpp.

References func_params, and func_paramsQuantity.

6.7.3.7 OrtsType functionNode::getParamType (BYTE *num*)

6.7.3.8 WORD functionNode::getParamsQuantity ( )

Getter for function parameters quantity.

Definition at line 51 of file functionNode.cpp.

References func_paramsQuantity.

6.7.3.9 WORD functionNode::getAllParamsLength ( )

Method calculate sum of all function node parameters size.

Returns

size of all parameters in bytes

Definition at line 117 of file functionNode.cpp.

References func_params, func_paramsLength, func_paramsQuantity, and param_desc::length().

Here is the call graph for this function:

6.7.3.10 errType functionNode::decodeParams (rcsCmd $*$ *packet*)

Received message parameters decode.

check parameters quantity and fills parameters pointers with received values

Parameters

packet	- received rcsCmd message

Returns

err_result_error - received parameters does not match to declaration
err_result_ok - received parameters successfully decodes

1. Define length of parametric part by function declaration
2. Count length of parametric part of received request message

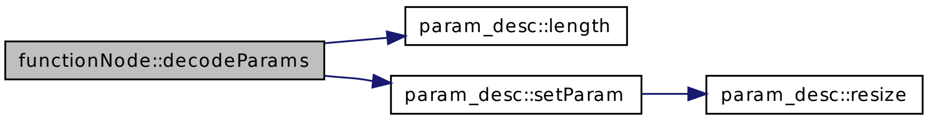
3. Compare received params quantity with declared params quantity

4. If received params quantity is equal to declared params quantity - fills parameters values. otherwise - return value is err_result_error.

Definition at line 191 of file functionNode.cpp.

References func_params, func_paramsLength, func_paramsQuantity, param_desc::length(), and param_desc::setParam().

Here is the call graph for this function:



6.7.3.11 errType functionNode::setParam (BYTE *num*, void ∗ *param*)

Set value for declared parameter.

Parameters

num	- parameter index (zero based index)
param	- pointer to value

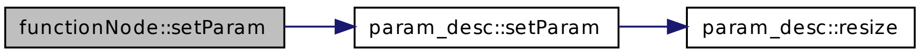
Returns

err_result_ok - value setting was successful

Definition at line 165 of file functionNode.cpp.

References func_params, and param_desc::setParam().

Here is the call graph for this function:



6.7.3.12 void ∗ functionNode::getParamPtr (BYTE *num*)

Get value pointer of declared parameter.

Parameters

num	- parameter index (zero based index)

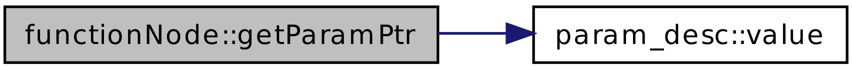
Returns

nonzero - value pointer
0 - parameter not exist

Definition at line 178 of file functionNode.cpp.

References func_params, and param_desc::value().

Here is the call graph for this function:



6.7.3.13 BYTE functionNode::getResultsQuantity ( )

Getter for function results quantity.

Definition at line 153 of file functionNode.cpp.

References func_resultsQuantity.

6.7.3.14 DWORD functionNode::getAllResultsLength ( )

Method calculate sum of all function node results size.

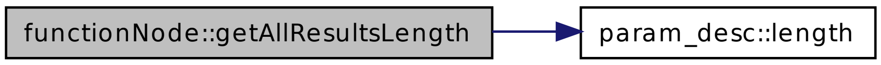
Returns

size of all results in bytes

Definition at line 131 of file functionNode.cpp.

References func_results, func_resultsLength, func_resultsQuantity, and param_desc::length().

Here is the call graph for this function:



6.7.3.15 DWORD functionNode::getResultLength (BYTE *res_no*)

Method evaluate function node result size.

Parameters

num	- result index (zero based index)

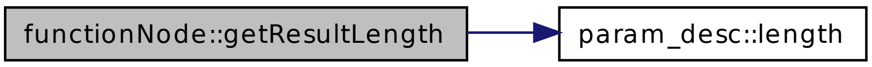
Returns

size of result in bytes

Definition at line 145 of file functionNode.cpp.

References func_results, and param_desc::length().

Here is the call graph for this function:



6.7.3.16 OrtsType functionNode::getResultType (BYTE *num*)

6.7.3.17 errType functionNode::getResult (BYTE *num,* void ∗∗ *out_res,* DWORD ∗ *length*)

Getter function for function node result storage.

Parameters

num[in]	- number of result in declaration of function node

out_res[out]	- pointer to result value
length[out]	- length in bytes of stored value

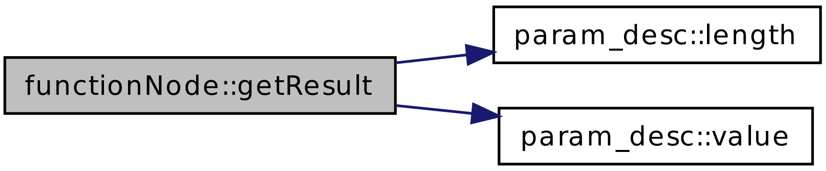
Returns

err_result_ok - value gets successfully

Definition at line 280 of file functionNode.cpp.

References func_results, param_desc::length(), and param_desc::value().

Here is the call graph for this function:



6.7.3.18 errType functionNode::setResult (BYTE *num,* void ∗ *res*)

Setter function for function node result storage.

Parameters

num[in]	- number of result in declaration of function node
res[in]	- pointer to result value

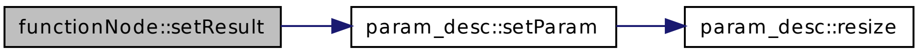
Returns

err_result_ok - value successfully sets

Definition at line 261 of file functionNode.cpp.

References func_id, func_results, and param_desc::setParam().

Here is the call graph for this function:



6.7.3.19 errType functionNode::printParams ( )

Prints params values of function node for debug purposes.

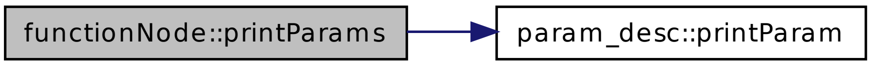
Returns

err_result_ok - values prints successfully

Definition at line 293 of file functionNode.cpp.

References func_params, func_paramsQuantity, and param_desc::printParam().

Here is the call graph for this function:



6.7.3.20 errType functionNode::printResults ( )

Prints results values of function node for debug purposes.

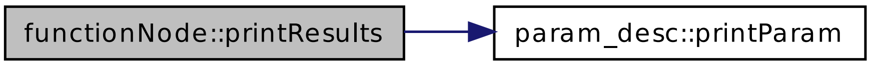
Returns

err_result_ok - values prints successfully

Definition at line 316 of file functionNode.cpp.

References func_results, func_resultsQuantity, and param_desc::printParam().

Here is the call graph for this function:



6.7.3.21 errType functionNode::callFunction ( )

Call function node pointer to execution code.

Returns

result - result code returns from calling code as return value

Definition at line 338 of file functionNode.cpp.

References func, and func_name.

6.7.3.22 BYTE functionNode::id ( )

Getter for function node identifier.

Returns

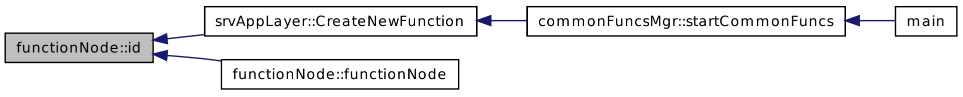
func_id - function identifier

Definition at line 352 of file functionNode.cpp.

References func_id.

Referenced by srvAppLayer::CreateNewFunction(), and functionNode().

Here is the caller graph for this function:


functionNode::id
srvAppLayer::CreateNewFunction
commonFuncsMgr::startCommonFuncs
main
functionNode::functionNode

6.7.3.23 void functionNode::dbgPrint ( )

Print for debug purposes contents of function node (in a hexadecimal notation)

Definition at line 404 of file functionNode.cpp.

References func_id, func_params, func_paramsLength, func_paramsQuantity, and func_-resultsQuantity.

6.7.4 Member Data Documentation

6.7.4.1 BYTE functionNode::func_id [private]

function idintifier.

Definition at line 22 of file functionNode.h.

Referenced by dbgPrint(), functionNode(), id(), and setResult().

6.7.4.2 const char∗ functionNode::func_name [private]

function name. For debug printing purposes.

Definition at line 23 of file functionNode.h.

Referenced by callFunction(), and setFuncName().

6.7.4.3 WORD functionNode::func_paramsQuantity [private]

function declaration. Quantity of params.

Definition at line 24 of file functionNode.h.

Referenced by dbgPrint(), decodeParams(), functionNode(), getAllParamsLength(), get-ParamLength(), getParamsQuantity(), printParams(), setParamDescriptor(), and ∼functionNode().

6.7.4.4 WORD functionNode::func_resultsQuantity [private]

function declaration. Quantity of results.

Definition at line 25 of file functionNode.h.

Referenced by dbgPrint(), functionNode(), getAllResultsLength(), getResultsQuantity(), printResults(), and ∼functionNode().

6.7.4.5 WORD functionNode::func_paramsLength [private]

Size in bytes of all function parameters.

Definition at line 27 of file functionNode.h.

Referenced by dbgPrint(), decodeParams(), functionNode(), getAllParamsLength(), and setParamDescriptor().

6.7.4.6 WORD functionNode::func_resultsLength `[private]`

Size in bytes of all function reaults.

Definition at line 28 of file functionNode.h.

Referenced by functionNode(), and getAllResultsLength().

6.7.4.7 param_desc∗ functionNode::func_params[32] `[private]`

function parameters declarations. 32 parameters at maximum.

Definition at line 30 of file functionNode.h.

Referenced by dbgPrint(), decodeParams(), functionNode(), getAllParamsLength(), getParamLength(), getParamPtr(), printParams(), setParam(), setParamDescriptor(), setParamName(), and ∼functionNode().

6.7.4.8 param_desc∗ functionNode::func_results[32] `[private]`

function results declaration. 32 results at maximum.

Definition at line 31 of file functionNode.h.

Referenced by getAllResultsLength(), getResult(), getResultLength(), printResults(), setResult(), setResultDescriptor(), setResultName(), and ∼functionNode().

6.7.4.9 funcPtr functionNode::func `[private]`

pointer to a function execution code.

Definition at line 33 of file functionNode.h.

Referenced by callFunction(), and functionNode().

The documentation for this class was generated from the following files:

- src/srvAppLayer/functionNode/functionNode.h
- src/srvAppLayer/functionNode/functionNode.cpp

6.8 param_desc Class Reference

parameter description

`#include <param_desc.h>`

Public Member Functions

- param_desc (OrtsType type, WORD len)

  Constructs parameter description with some type and information about storing value length.

- param_desc (OrtsType type)

  Overloaded constructor that constructs parameter description of a vector type.

- ∼param_desc ()
- errType resize (WORD new_size)

  Function calling if need to resize of storage for new value or something else...

- void ∗ value ()

  Getter for pointer of storing value.

- WORD length ()

  Getter for length of storing value.

- OrtsType type ()

  Getter for type of storing value.

- bool isVector ()

  Function checks if type of storing value is a vector.

- errType setParam (void ∗param_val)

  Setter function for setting param value.

- errType setName (const char ∗name)

  Setter for parameter string name.

- errType printParam ()

  Print parameter value for debug purposes.

- void dbgPrint ()

  Print for debug purposes contents of function node (in a hexadecimal notation)

Private Attributes

- void ∗ param

  pointer to parameter value

- char ∗ param_name

  parameter string name

- WORD _length

  length in bytes of storing parameter value

- OrtsType _type
- bool _isVector

  storage value is a vector (one-dimensional array) of simplified values types

6.8.1 Detailed Description

parameter description parameter description uses in function node parameter declaration

Definition at line 20 of file param_desc.h.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 param_desc::param_desc (OrtsType *type*, WORD *param_len*)

Constructs parameter description with some type and information about storing value length.

Parameters

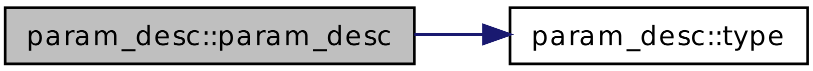
type	- type (or simplified reference for vector) of storing value
param_len	- length in bytes of storing value

Returns

Definition at line 26 of file param_desc.cpp.

References _isVector, _length, _type, param, and type().

Here is the call graph for this function:



6.8.2.2 param_desc::param_desc (OrtsType *type*)

Overloaded constructor that constructs parameter description of a vector type.

Parameters

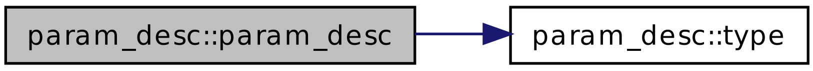
type	- type of storing value

Returns

Definition at line 40 of file param_desc.cpp.

References _isVector, _length, _type, and type().

Here is the call graph for this function:



6.8.2.3 param_desc::∼param_desc ( )

Definition at line 49 of file param_desc.cpp.

References param, and param_name.

6.8.3 Member Function Documentation

6.8.3.1 errType param_desc::resize (WORD *new_size*)

Function calling if need to resize of storage for new value or something else...

Parameters

new_size	- size in bytes of new storage

Returns

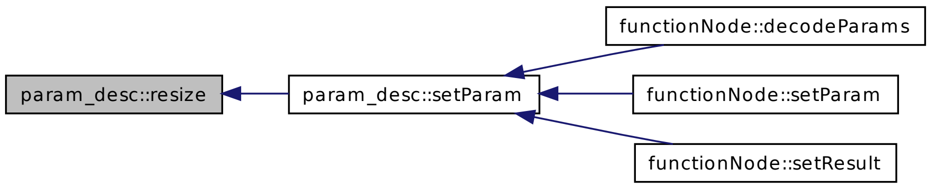
err_result_ok - memory successfully reallocated
err_mem_alloc - error of allocating memory for new size
err_not_init - new size does not differ from old size

Definition at line 63 of file param_desc.cpp.

References _length, and param.

Referenced by setParam().

Here is the caller graph for this function:



6.8.3.2 void ∗ param_desc::value ( )

Getter for pointer of storing value.

Returns

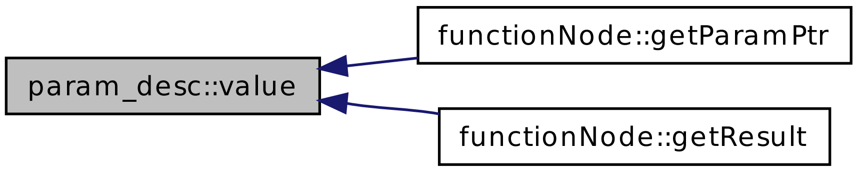
param - pointer to storing value

Definition at line 92 of file param_desc.cpp.

References param.

Referenced by functionNode::getParamPtr(), and functionNode::getResult().

Here is the caller graph for this function:



6.8.3.3 WORD param_desc::length ( )

Getter for length of storing value.

Returns

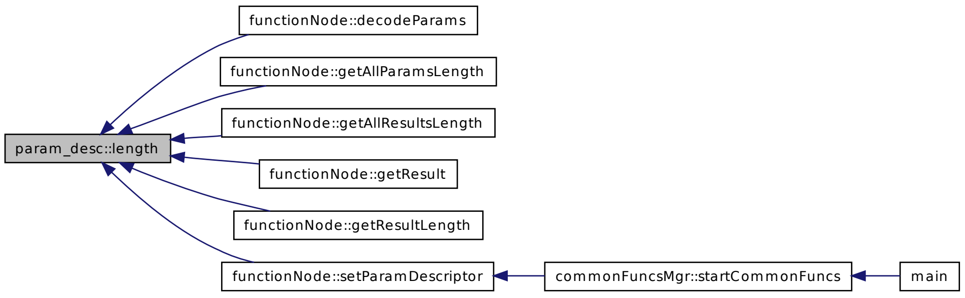
length - storage size in bytes

Definition at line 83 of file param_desc.cpp.

References _length.

Referenced by functionNode::decodeParams(), functionNode::getAllParamsLength(), functionNode::getAllResultsLength(), functionNode::getResult(), functionNode::getResultLength(), and functionNode::setParamDescriptor().

Here is the caller graph for this function:


functionNode::decodeParams
functionNode::getAllParamsLength
functionNode::getAllResultsLength
param_desc::length
functionNode::getResult
functionNode::getResultLength
functionNode::setParamDescriptor
commonFuncsMgr::startCommonFuncs
main

6.8.3.4 OrtsType param_desc::type ( )

Getter for type of storing value.

Returns

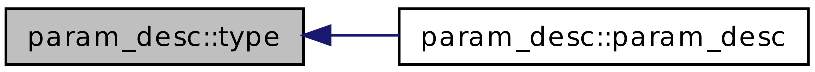
type - pointer to storing value type

Definition at line 101 of file param_desc.cpp.

References _type.

Referenced by param_desc().

Here is the caller graph for this function:



6.8.3.5 bool param_desc::isVector ( )

Function checks if type of storing value is a vector.

Returns

true - storing value is a vector
false - storing value is a scalar

Definition at line 111 of file param_desc.cpp.

References _type.

6.8.3.6 errType param_desc::setParam (void ∗ *param_val*)

Setter function for setting param value.

Parameters

param_-val[in]	- pointer to parameter value

Returns

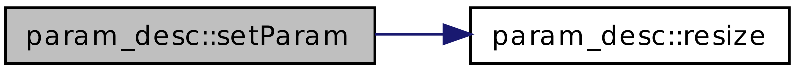
err_result_ok - value was successfully set
err_params_value - null-pointer error

Definition at line 122 of file param_desc.cpp.

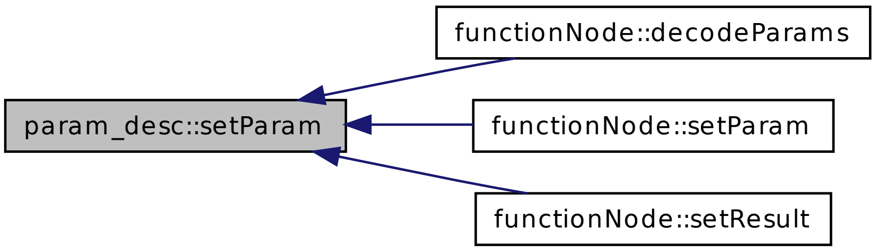
References _isVector, _length, param, and resize().

Referenced by functionNode::decodeParams(), functionNode::setParam(), and functionN-ode::setResult().

Here is the call graph for this function:



Here is the caller graph for this function:



6.8.3.7 errType param_desc::setName (const char $*$ *name*)

Setter for parameter string name.

Parameters

name	- name of a parameter.

Returns

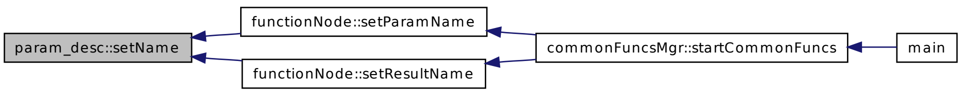
err_result_ok

Definition at line 139 of file param_desc.cpp.

References param_name.

Referenced by functionNode::setParamName(), and functionNode::setResultName().

Here is the caller graph for this function:


param_desc::setName
functionNode::setParamName
functionNode::setResultName
commonFuncsMgr::startCommonFuncs
main

6.8.3.8 errType param_desc::printParam ( )

Print parameter value for debug purposes.

Returns

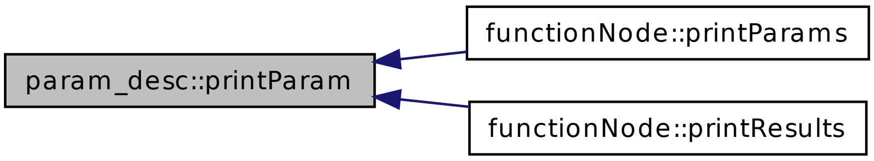
err_result_ok - value was printed successfully

Definition at line 151 of file param_desc.cpp.

References _type, param, and param_name.

Referenced by functionNode::printParams(), and functionNode::printResults().

Here is the caller graph for this function:



6.8.3.9 void param_desc::dbgPrint ( )

Print for debug purposes contents of function node (in a hexadecimal notation)

Definition at line 207 of file param_desc.cpp.

References param.

6.8.4 Member Data Documentation

6.8.4.1 void∗ param_desc::param `[private]`

pointer to parameter value

Definition at line 22 of file param_desc.h.

Referenced by dbgPrint(), param_desc(), printParam(), resize(), setParam(), value(), and ∼param_desc().

6.8.4.2 char∗ param_desc::param_name `[private]`

parameter string name

Definition at line 23 of file param_desc.h.

Referenced by printParam(), setName(), and ∼param_desc().

6.8.4.3 WORD param_desc::_length `[private]`

length in bytes of storing parameter value

Definition at line 24 of file param_desc.h.

Referenced by length(), param_desc(), resize(), and setParam().

6.8.4.4 OrtsType param_desc::_type `[private]`

Definition at line 25 of file param_desc.h.

Referenced by isVector(), param_desc(), printParam(), and type().

6.8.4.5 bool param_desc::_isVector `[private]`

storage value is a vector (one-dimensional array) of simplified values types

Definition at line 27 of file param_desc.h.

Referenced by param_desc(), and setParam().

The documentation for this class was generated from the following files:

- src/srvAppLayer/functionNode/param_desc.h
- src/srvAppLayer/functionNode/param_desc.cpp

6.9 serviceState Struct Reference

stateVector_type structural field.

`#include <srvAppLayer.h>`

Public Attributes

- BYTE linked:1
- BYTE reserved:3
- BYTE inprocess:1
- BYTE mode_emergency:1
- BYTE mode_manual:1
- BYTE mode_auto:1

6.9.1 Detailed Description

stateVector_type structural field.

Definition at line 21 of file srvAppLayer.h.

6.9.2 Member Data Documentation

6.9.2.1 BYTE serviceState::linked

Definition at line 22 of file srvAppLayer.h.

6.9.2.2 BYTE serviceState::reserved

Definition at line 23 of file srvAppLayer.h.

6.9.2.3 BYTE serviceState::inprocess

Definition at line 24 of file srvAppLayer.h.

6.9.2.4 BYTE serviceState::mode_emergency

Definition at line 25 of file srvAppLayer.h.

6.9.2.5 BYTE serviceState::mode_manual

Definition at line 26 of file srvAppLayer.h.

6.9.2.6 BYTE serviceState::mode_auto

Definition at line 27 of file srvAppLayer.h.

The documentation for this struct was generated from the following file:

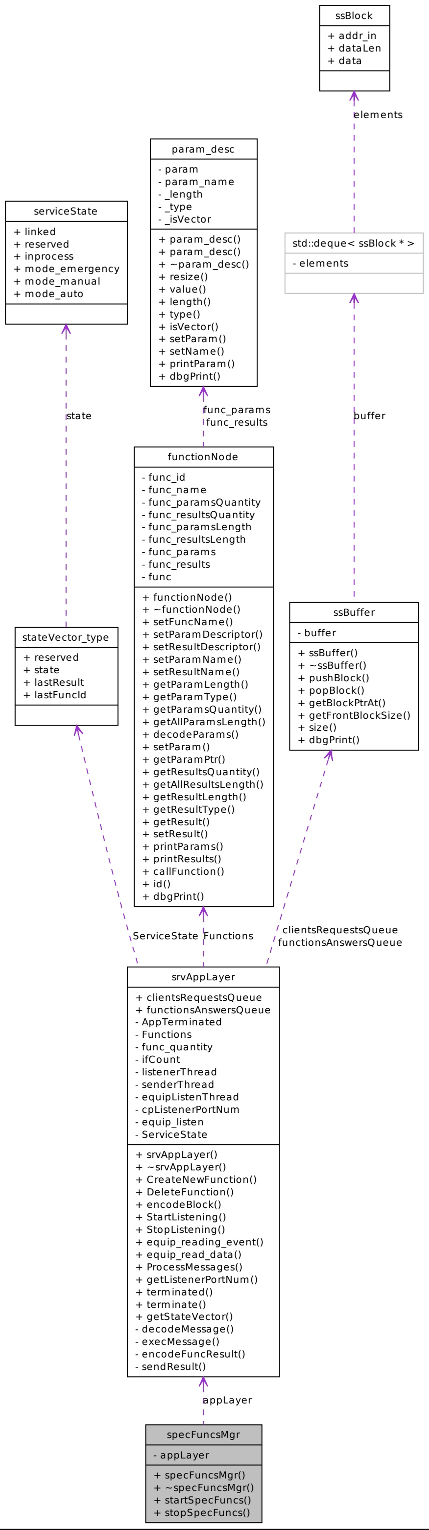
- src/srvAppLayer/srvAppLayer.h

6.10 specFuncsMgr Class Reference

special functions set manager.

```
#include <specFuncsMgr.h>
```

Collaboration diagram for specFuncsMgr:



Public Member Functions

- specFuncsMgr (srvAppLayer ∗appl)

  Links special functions set with srvAppLayer.

- ∼specFuncsMgr ()
- errType startSpecFuncs ()

  Declare special functions set.

- errType stopSpecFuncs ()

Private Attributes

- srvAppLayer ∗ appLayer

6.10.1 Detailed Description

special functions set manager. implementation in /funcs/0/0_SpecFuncs.cpp ... /funcs/9/9_-CommonFuncs.cpp (set of function depends on target)

Definition at line 20 of file specFuncsMgr.h.

6.10.2 Constructor & Destructor Documentation

6.10.2.1 specFuncsMgr::specFuncsMgr (srvAppLayer ∗ *appl*)

Links special functions set with srvAppLayer.

6.10.2.2 specFuncsMgr::∼specFuncsMgr ( )

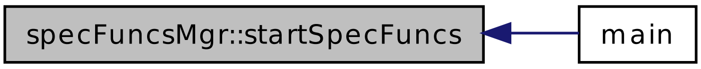
6.10.3 Member Function Documentation

6.10.3.1 errType specFuncsMgr::startSpecFuncs ( )

Declare special functions set.

Referenced by main().

Here is the caller graph for this function:



6.10.3.2 errType specFuncsMgr::stopSpecFuncs ( )

6.10.4 Member Data Documentation

6.10.4.1 srvAppLayer∗ specFuncsMgr::appLayer `[private]`

Definition at line 22 of file specFuncsMgr.h.

The documentation for this class was generated from the following file:

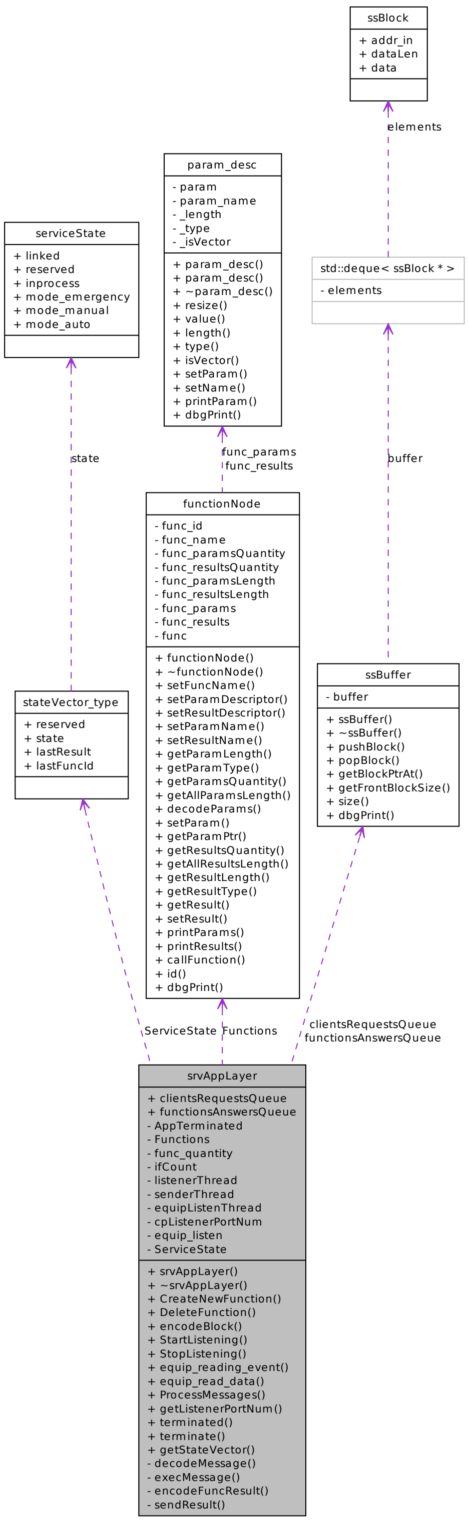
- src/functions/specFuncsMgr.h

6.11 srvAppLayer Class Reference

Application core layer implementaion.

`#include <srvAppLayer.h>`

Collaboration diagram for srvAppLayer:



Public Member Functions

- srvAppLayer (WORD portNum)

  Constructs instance of srvAppLayer and links it to udp port number for listening clients requests.

- ∼srvAppLayer ()
- errType CreateNewFunction (functionNode ∗func)

  Method to add new declared function to service layer.

- errType DeleteFunction (BYTE id)

  Method to delete function from service layer.

- errType encodeBlock (rcsCmd ∗, BYTE ∗∗)

  Method to encode data from rcsCmd message.

- errType StartListening ()

  Method to prepare and start base communication engine.

- errType StopListening ()

  Method to stop base communication engine.

- errType equip_reading_event ()

  Method to asynchonous polling of equip_listen socket.

- errType equip_read_data (BYTE ∗, size_t ∗)

  Method to read data from equip_listen socket.

- errType ProcessMessages ()

  Method to make one step of srvAppLayer step.

- WORD getListenerPortNum ()

  getter for udp port number that listens all clients requests

- BYTE terminated ()

  Method to check application internal termination signal.

- void terminate (BYTE mode=1)

  getter for AppTerminated signal

- stateVector_type getStateVector ()

  getter for ServiceState vector

Public Attributes

- ssBuffer ∗ clientsRequestsQueue

  Queue that stores received requests from client.

- ssBuffer ∗ functionsAnswersQueue

  Queue that stores service functions answers to clients.

Private Member Functions

- errType decodeMessage (BYTE ∗dataBlock, DWORD length, rcsCmd ∗ss_cmd)

  step 1. decode recieved message from client

- errType execMessage (rcsCmd ∗ss_cmd)

  step 2. send data to requested service function

- errType encodeFuncResult (rcsCmd ∗in_cmd, rcsCmd ∗out_cmd)

  step 3. encode function execution results for sending back to client

- errType sendResult (sockaddr_in ∗sin, rcsCmd ∗ss_cmd)

  step 4. send function answer to client

Private Attributes

- BYTE AppTerminated

  Application termination process indicator
  - *0 - Application run normally*
  - *1 - Applcation need to exit only*
  - *2 - Application need exit with reboot.*

- functionNode ∗ Functions [100]

  Service functions array.

- BYTE func_quantity

  Counter that stores really declared functions quantity.

- BYTE ifCount

  Counter of ethernet interfaces. No have an idea how to use it.

- pthread_t listenerThread

  Handle to client requests listening thread.

- pthread_t senderThread

  Handle to client answers sending thread.

- pthread_t equipListenThread

  Handle to equipmnent data listening thread.

- WORD cpListenerPortNum

  settings: Udp port number to listen requests from network clients

- udp_port ∗ equip_listen

  udp_port instance that associates with listening data from equipment

- stateVector_type ServiceState

  Service state vector.

6.11.1 Detailed Description

Application core layer implementaion. This layer delegate network calls to service functions and return back functions results.

Definition at line 46 of file srvAppLayer.h.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 srvAppLayer::srvAppLayer (WORD *portNum*)

Constructs instance of srvAppLayer and links it to udp port number for listening clients requests.

Parameters

`in`	*portNum*	- udp port number that will use for clients requests listening

Definition at line 159 of file srvAppLayer.cpp.

References AppTerminated, cpListenerPortNum, func_quantity, and ifCount.

6.11.2.2 srvAppLayer::∼srvAppLayer ( )

Definition at line 168 of file srvAppLayer.cpp.

References equip_listen, func_quantity, and Functions.

6.11.3 Member Function Documentation

6.11.3.1 errType srvAppLayer::decodeMessage (BYTE ∗ *dataBlock*, DWORD *length*, rcsCmd ∗ *ss_cmd*) [private]

step 1. decode recieved message from client

Method to decode message data to function call and parameters set.

Method using method rcsCmd::encode to transform data array to rcsCmd message.

Method checks for correct sign, calling function existing and compares real received data length with header information about data length.

Parameters

in	*dataBlock*	- pointer to received bytes array
in	*length*	- size of received bytes array.
out	*ss_cmd*	- decoded message.

Return values

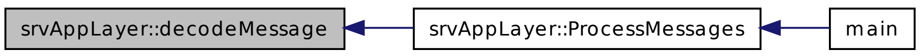
err_result_ok	- decoded message is correct.
err_params_decode	- params in decoded message is incorrect.
err_not_found	- decoded message is calling for non existing function
err_crc_error	- decoded message signature is incorrect.

Definition at line 333 of file srvAppLayer.cpp.

References Functions.

Referenced by ProcessMessages().

Here is the caller graph for this function:



6.11.3.2 errType srvAppLayer::execMessage (rcsCmd ∗ *ss_cmd*) [private]

step 2. send data to requested service function

Method to execute function by id in rcsCmd message.

Method checks for correct params by comparing params with description in commonFuncsMgr or specFuncsMgr.

After requested function calling method changes ServiceState vector and set answer ticket for return value from requested function.

Todo

ServiceState vector need to change before function calling.

Parameters

in	*ss_cmd*	- message with request for function calling.

Return values

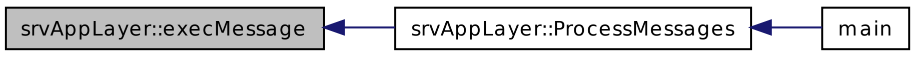
err_result_ok	- message executed successfully.
err_params_decode	- params in message differs with description.

Definition at line 383 of file srvAppLayer.cpp.

References Functions, stateVector_type::lastFuncId, stateVector_type::lastResult, and ServiceState.

Referenced by ProcessMessages().

Here is the caller graph for this function:



6.11.3.3 errType srvAppLayer::encodeFuncResult (rcsCmd $*$ *in_cmd*, rcsCmd $*$ *out_cmd*) [private]

step 3. encode function execution results for sending back to client

Method encoding all executed function results to rcsCmd message.

Method preparing rcsCmd message with making of message signing.

Todo

Need a refactoring.

Parameters

in	*in_cmd*	- message with request for function calling.
out	*out_cmd*	- message with results from requested function.

Return values

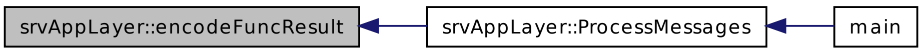
err_result_ok	- message executed successfully.
err_not_found	- function to execute was not found.

Definition at line 413 of file srvAppLayer.cpp.

References Functions.

Referenced by ProcessMessages().

Here is the caller graph for this function:



6.11.3.4 errType srvAppLayer::sendResult (sockaddr_in * *sfrom*, rcsCmd * *ss_cmd*) `[private]`

step 4. send function answer to client

Method sending rcsCmd message to needed recepient.

Method preparing data array from rcsCmd message and push data array with address of recepient in to functionsAnswersQueue

Parameters

`in`	*sfrom*	- recepient address.
`in`	*ss_cmd*	- message needed to send.

Return values

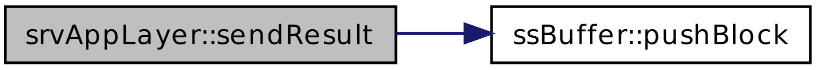
err_result_ok	- message added to sending queue successfully.

Definition at line 462 of file srvAppLayer.cpp.

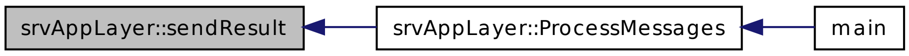
References functionsAnswersQueue, ssBuffer::pushBlock(), and wUdp.

Referenced by ProcessMessages().

Here is the call graph for this function:



Here is the caller graph for this function:



6.11.3.5 errType srvAppLayer::CreateNewFunction (functionNode ∗ *func*)

Method to add new declared function to service layer.

Parameters

`in`	*func*	- functionNode instance

Return values

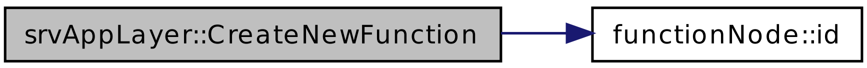
err_result_ok	- function added successfully

Definition at line 182 of file srvAppLayer.cpp.

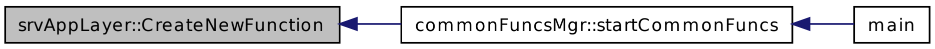
References func_quantity, Functions, and functionNode::id().

Referenced by commonFuncsMgr::startCommonFuncs().

Here is the call graph for this function:



Here is the caller graph for this function:


srvAppLayer::CreateNewFunction
commonFuncsMgr::startCommonFuncs
main

6.11.3.6 errType srvAppLayer::DeleteFunction (BYTE *id*)

Method to delete function from service layer.

Parameters

in	*id*	- functionNode identifier

Return values

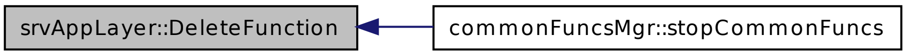
err_result_ok	- function deleted successfully

Definition at line 197 of file srvAppLayer.cpp.

References func_quantity, and Functions.

Referenced by commonFuncsMgr::stopCommonFuncs().

Here is the caller graph for this function:



6.11.3.7 errType srvAppLayer::encodeBlock (rcsCmd $*$ *ss_cmd,* BYTE $**$ *data*)

Method to encode data from rcsCmd message.

Parameters

in	*ss_cmd*	- rcsCmd message
out	*data*	- pointer to pointer that will include result of method

Return values

err_result_ok	- Block encoded successfully

Todo

nobody uses this method. need to be deleted.

Definition at line 214 of file srvAppLayer.cpp.

6.11.3.8 errType srvAppLayer::StartListening ( )

Method to prepare and start base communication engine.

if errors has been occured,method initiate appTerminate signal.

Return values

err_result_ok	- communication engine starts successfully
err_sock_error	- udp sockets prerparing error

1. Prepare clientsRequestsQueue.

Queue stores all received messages from clients and clients adresses.

2. Prepare functionsAnswersQueue.

Queue stores all functions answers to clients requests.

3. Start udpListenerThread

Thread listen for udp messages from clients and stores messages in clientsRequest-sQueue.

4. Prepare equip_listen.

This is an udp port instance that using to listen data from equipment

5. Start udpSenderThread

Thread sending udp messageds to clients from functionsAnswersQueue

6. Start equipListenPolling

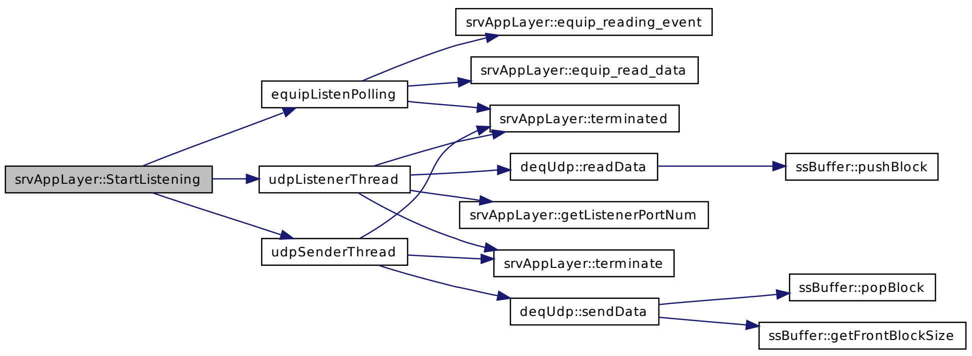
Thread listen for udp messages from equipment

Definition at line 229 of file srvAppLayer.cpp.

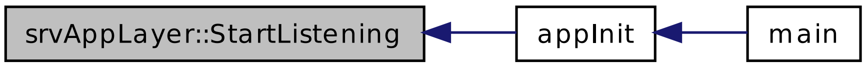
References AppTerminated, clientsRequestsQueue, eq_udp_listen_port, equip_listen, equipListenPolling(), equipListenThread, functionsAnswersQueue, ifCount, listenerThread, senderThread, udpListenerThread(), and udpSenderThread().

Referenced by appInit().

Here is the call graph for this function:



Here is the caller graph for this function:



6.11.3.9 errType srvAppLayer::StopListening ( )

Method to stop base communication engine.

method initiate appTerminate signal, deletes queues.

Return values

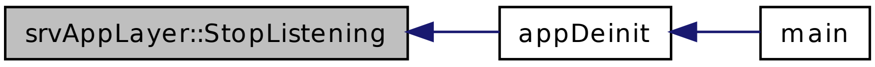
err_result_ok	- engine stopped successfully

Definition at line 274 of file srvAppLayer.cpp.

References AppTerminated, clientsRequestsQueue, functionsAnswersQueue, and verbose_-level.

Referenced by appDeinit().

Here is the caller graph for this function:



6.11.3.10 errType srvAppLayer::equip_reading_event ( )

Method to asynchonous polling of equip_listen socket.

Todo

too strange method.

May be it need be deleted.

Return values

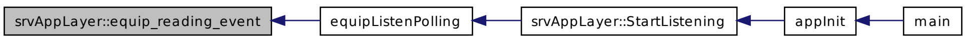
err_result_ok	- udp socket received data

Definition at line 301 of file srvAppLayer.cpp.

References equip_listen.

Referenced by equipListenPolling().

Here is the caller graph for this function:



6.11.3.11 errType srvAppLayer::equip_read_data (BYTE ∗ *buffer*, size_t ∗ *sz*)

Method to read data from equip_listen socket.

Todo

too strange method.

May be it need be deleted.

Parameters

out	*buffer*	- uses to store recevied data
out	*sz*	- size in bytes of received data

Return values

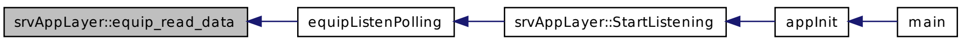
err_result_ok	- udp socket received data has been readed

Definition at line 316 of file srvAppLayer.cpp.

References equip_listen.

Referenced by equipListenPolling().

Here is the caller graph for this function:



6.11.3.12 errType srvAppLayer::ProcessMessages ( )

Method to make one step of srvAppLayer step.

Method checks at start AppTerminated signal, size of clientsRequestsQueue and rcvComplete_-flag before it continuous execution.

Return values

err_result_ok	- one step of engine execution was successfully.
err_not_init	- method break execution.

1) Read from clientsRequestsQueue one new request

2) Decode readed request by decodeMessage

3) Execute requested function if decoding was successfully by execMessage

4) Encoding function result ticket if execution was not successfully

5) Encode remains function results be encodeFuncResult

6) Write results to sending queue by sendResult

7) Release allocated memory

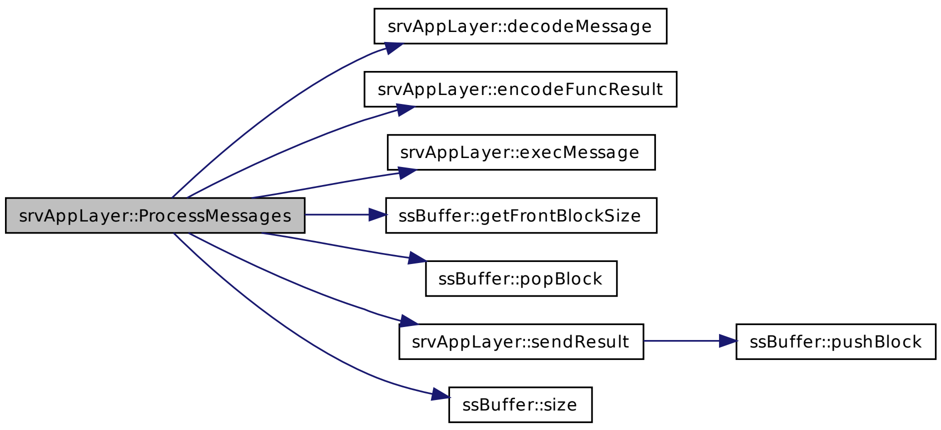
8) Sync listening and sending threads by rcvComplete_flag and sndAllow_flag

Definition at line 488 of file srvAppLayer.cpp.

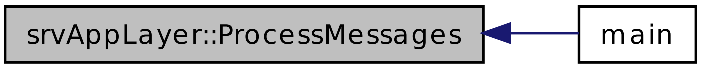
References AppTerminated, clientsRequestsQueue, decodeMessage(), encodeFuncResult(), execMessage(), Functions, ssBuffer::getFrontBlockSize(), ssBuffer::popBlock(), rcvComplete_flag, sendResult(), ssBuffer::size(), and sndAllow_flag.

Referenced by main().

Here is the call graph for this function:


srvAppLayer::decodeMessage
srvAppLayer::encodeFuncResult
srvAppLayer::execMessage
srvAppLayer::ProcessMessages
ssBuffer::getFrontBlockSize
ssBuffer::popBlock
srvAppLayer::sendResult
ssBuffer::pushBlock
ssBuffer::size

Here is the caller graph for this function:



6.11.3.13 WORD srvAppLayer::getListenerPortNum ( )

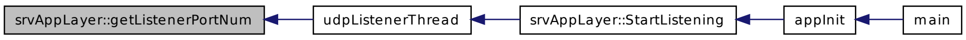
getter for udp port number that listens all clients requests

Definition at line 565 of file srvAppLayer.cpp.

References cpListenerPortNum.

Referenced by udpListenerThread().

Here is the caller graph for this function:



6.11.3.14 BYTE srvAppLayer::terminated ( )

Method to check application internal termination signal.

Return values

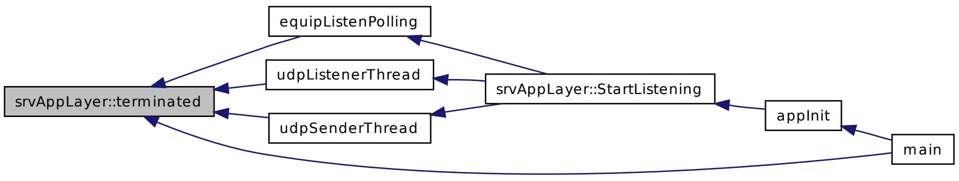
0	- Application runs normally
1	- Application terminating and preparing to exit to the OS
2	- Application terminating and preparing to reboot the OS

Definition at line 148 of file srvAppLayer.cpp.

References AppTerminated.

Referenced by equipListenPolling(), main(), udpListenerThread(), and udpSenderThread().

Here is the caller graph for this function:



6.11.3.15 void srvAppLayer::terminate (BYTE *mode* = 1)

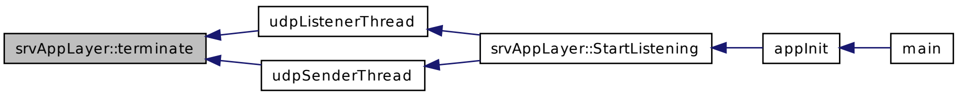
getter for AppTerminated signal

Definition at line 571 of file srvAppLayer.cpp.

References AppTerminated.

Referenced by udpListenerThread(), and udpSenderThread().

Here is the caller graph for this function:



6.11.3.16 stateVector_type srvAppLayer::getStateVector ( )

getter for ServiceState vector

Definition at line 577 of file srvAppLayer.cpp.

References ServiceState.

6.11.4 Member Data Documentation

6.11.4.1 BYTE srvAppLayer::AppTerminated [private]

Application termination process indicator

- 0 - Application run normally

- 1 - Applcation need to exit only
- 2 - Application need exit with reboot.

Definition at line 48 of file srvAppLayer.h.

Referenced by ProcessMessages(), srvAppLayer(), StartListening(), StopListening(), terminate(), and terminated().

6.11.4.2 functionNode∗ srvAppLayer::Functions[100] `[private]`

Service functions array.

100 functions at maximum.

Todo

array not the best data structure for this purposes.

Definition at line 53 of file srvAppLayer.h.

Referenced by CreateNewFunction(), decodeMessage(), DeleteFunction(), encodeFuncResult(), execMessage(), ProcessMessages(), and ∼srvAppLayer().

6.11.4.3 BYTE srvAppLayer::func_quantity `[private]`

Counter that stores really declared functions quantity.

Definition at line 56 of file srvAppLayer.h.

Referenced by CreateNewFunction(), DeleteFunction(), srvAppLayer(), and ∼srvAppLayer().

6.11.4.4 BYTE srvAppLayer::ifCount `[private]`

Counter of ethernet interfaces. No have an idea how to use it.

Definition at line 57 of file srvAppLayer.h.

Referenced by srvAppLayer(), and StartListening().

6.11.4.5 pthread_t srvAppLayer::listenerThread `[private]`

Handle to client requests listening thread.

Definition at line 59 of file srvAppLayer.h.

Referenced by StartListening().

6.11.4.6 pthread_t srvAppLayer::senderThread `[private]`

Handle to client answers sending thread.

Definition at line 60 of file srvAppLayer.h.

Referenced by StartListening().

6.11.4.7 pthread_t srvAppLayer::equipListenThread [private]

Handle to equipmnent data listening thread.

Note

programm not have equipment data sending thread

Definition at line 61 of file srvAppLayer.h.

Referenced by StartListening().

6.11.4.8 WORD srvAppLayer::cpListenerPortNum [private]

settings: Udp port number to listen requests from network clients

Definition at line 64 of file srvAppLayer.h.

Referenced by getListenerPortNum(), and srvAppLayer().

6.11.4.9 udp_port∗ srvAppLayer::equip_listen [private]

udp_port instance that associates with listening data from equipment

Definition at line 65 of file srvAppLayer.h.

Referenced by equip_read_data(), equip_reading_event(), StartListening(), and ∼srvAppLayer().

6.11.4.10 stateVector_type srvAppLayer::ServiceState [private]

Service state vector.

Definition at line 72 of file srvAppLayer.h.

Referenced by execMessage(), and getStateVector().

6.11.4.11 ssBuffer∗ srvAppLayer::clientsRequestsQueue

Queue that stores received requests from client.

Definition at line 76 of file srvAppLayer.h.

Referenced by ProcessMessages(), StartListening(), StopListening(), and udpListenerThread().

6.11.4.12 ssBuffer∗ srvAppLayer::functionsAnswersQueue

Queue that stores service functions answers to clients.

Definition at line 77 of file srvAppLayer.h.

Referenced by sendResult(), StartListening(), StopListening(), and udpSenderThread().

The documentation for this class was generated from the following files:

- src/srvAppLayer/srvAppLayer.h
- src/srvAppLayer/srvAppLayer.cpp

6.12 ssBlock Struct Reference

ssBuffer list entry.

`#include <ssBuffer.h>`

Public Attributes

- sockaddr_in addr_in

  recepient of datablock address,

- DWORD dataLen

  size in bytes of data,

- BYTE ∗ data

  data array pointer.

6.12.1 Detailed Description

ssBuffer list entry.

Definition at line 20 of file ssBuffer.h.

6.12.2 Member Data Documentation

6.12.2.1 sockaddr_in ssBlock::addr_in

recepient of datablock address,

Definition at line 22 of file ssBuffer.h.

Referenced by ssBuffer::dbgPrint(), ssBuffer::popBlock(), and ssBuffer::pushBlock().

6.12.2.2 DWORD ssBlock::dataLen

size in bytes of data,

Definition at line 23 of file ssBuffer.h.

Referenced by ssBuffer::dbgPrint(), ssBuffer::getFrontBlockSize(), ssBuffer::popBlock(), and ssBuffer::pushBlock().

6.12.2.3 BYTE∗ ssBlock::data

data array pointer.

Definition at line 24 of file ssBuffer.h.

Referenced by ssBuffer::dbgPrint(), ssBuffer::popBlock(), and ssBuffer::pushBlock().

The documentation for this struct was generated from the following file:

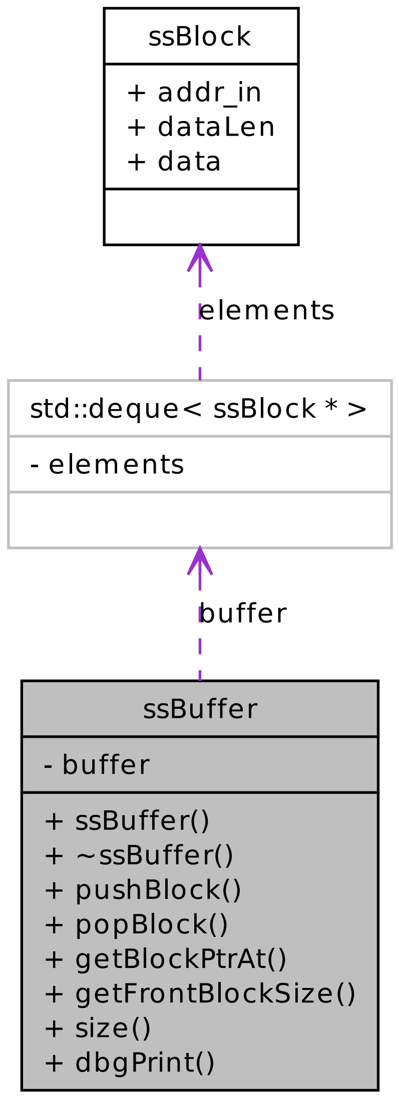
- src/buffer/ssBuffer.h

6.13 ssBuffer Class Reference

list (deque) implementaion for storing ssBlock elements.

```
#include <ssBuffer.h>
```

Collaboration diagram for ssBuffer:



Public Member Functions

- ssBuffer ()
- ∼ssBuffer ()

- errType pushBlock (sockaddr_in ∗, BYTE ∗, DWORD len)
  push datablock to queue
- DWORD popBlock (sockaddr_in ∗, BYTE ∗)
  pop datablock from queue
- errType getBlockPtrAt (int index, ssBlock ∗block)
  read item from queue by ***index***
- DWORD getFrontBlockSize ()
  get first in queue (front) block size in bytes
- DWORD size ()
  return length (items quantity) of queue
- void dbgPrint ()
  Print for debug purposes contents of stored queue (in a hexadecimal notation)

Private Attributes

- deque< ssBlock ∗ > buffer

6.13.1 Detailed Description

list (deque) implementaion for storing ssBlock elements. deque uses to organize sending or receiving queue

Definition at line 33 of file ssBuffer.h.

6.13.2 Constructor & Destructor Documentation

6.13.2.1 ssBuffer::ssBuffer ( )

Definition at line 23 of file ssBuffer.cpp.

6.13.2.2 ssBuffer::∼ssBuffer ( )

Definition at line 28 of file ssBuffer.cpp.

6.13.3 Member Function Documentation

6.13.3.1 errType ssBuffer::pushBlock (sockaddr_in ∗ *addr,* BYTE ∗ *block,* DWORD *len*)

push datablock to queue

copies data from **block** and **addr** pointers to new memory locations

Parameters

`in`	*addr*	- recepient address (owner of datablock)
`in`	*block*	- datablock
`in`	*len*	- size in bytes of datablock

Return values

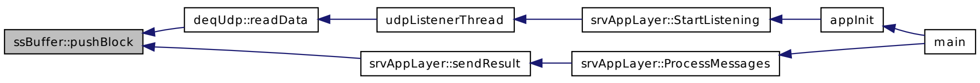
err_result_ok	- pushing was successfully

Definition at line 41 of file ssBuffer.cpp.

References ssBlock::addr_in, ssBlock::data, and ssBlock::dataLen.

Referenced by deqUdp::readData(), and srvAppLayer::sendResult().

Here is the caller graph for this function:



6.13.3.2 DWORD ssBuffer::popBlock (sockaddr_in ∗ *addr,* BYTE ∗ *block*)

pop datablock from queue

copies data from queue to **block** and **addr** pointers

block and **addr** need to be allocated before calling this method.

queue element will be removed from queue

Parameters

`in`	*addr*	- recepient address (owner of datablock)
`in`	*block*	- datablock

Return values

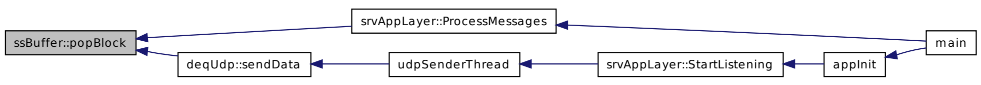
lenght	- size in bytes of readed datablock

Definition at line 77 of file ssBuffer.cpp.

References ssBlock::addr_in, ssBlock::data, and ssBlock::dataLen.

Referenced by srvAppLayer::ProcessMessages(), and deqUdp::sendData().

Here is the caller graph for this function:



6.13.3.3 errType ssBuffer::getBlockPtrAt (int *index*, ssBlock ∗ *block*)

read item from queue by **index**

Parameters

in	*index*	- queue item index
out	*block*	- readed queue item

Definition at line 99 of file ssBuffer.cpp.

6.13.3.4 DWORD ssBuffer::getFrontBlockSize ( )

get first in queue (front) block size in bytes

need to be prepared for reading next queue element

Return values

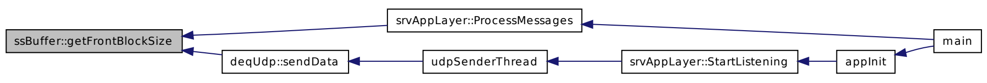
dataLen	- size in bytes of data block

Definition at line 61 of file ssBuffer.cpp.

References ssBlock::dataLen.

Referenced by srvAppLayer::ProcessMessages(), and deqUdp::sendData().

Here is the caller graph for this function:


ssBuffer::getFrontBlockSize
srvAppLayer::ProcessMessages
deqUdp::sendData
udpSenderThread
srvAppLayer::StartListening
appInit
main

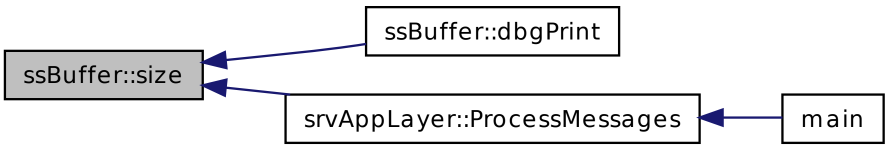
6.13.3.5 DWORD ssBuffer::size ( )

return length (items quantity) of queue

Definition at line 112 of file ssBuffer.cpp.

Referenced by dbgPrint(), and srvAppLayer::ProcessMessages().

Here is the caller graph for this function:



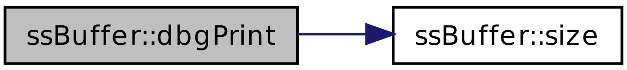
6.13.3.6 void ssBuffer::dbgPrint ( )

Print for debug purposes contents of stored queue (in a hexadecimal notation)

Definition at line 123 of file ssBuffer.cpp.

References ssBlock::addr_in, ssBlock::data, ssBlock::dataLen, and size().

Here is the call graph for this function:



6.13.4 Member Data Documentation

6.13.4.1 deque$<$ssBlock$*$$>$ ssBuffer::buffer [private]

Definition at line 35 of file ssBuffer.h.

The documentation for this class was generated from the following files:

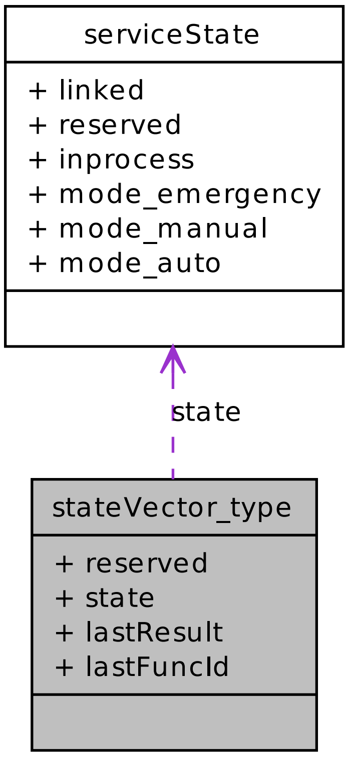
- src/buffer/ssBuffer.h
- src/buffer/ssBuffer.cpp

6.14 stateVector_type Struct Reference

Main vector of service base states.

```
#include <srvAppLayer.h>
```

Collaboration diagram for stateVector_type:



Public Attributes

- BYTE reserved
- serviceState state
- errType lastResult
- BYTE lastFuncId

6.14.1 Detailed Description

Main vector of service base states.

Definition at line 34 of file srvAppLayer.h.

6.14.2 Member Data Documentation

6.14.2.1 BYTE stateVector_type::reserved

Definition at line 35 of file srvAppLayer.h.

6.14.2.2 serviceState stateVector_type::state

Definition at line 36 of file srvAppLayer.h.

6.14.2.3 errType stateVector_type::lastResult

Definition at line 37 of file srvAppLayer.h.

Referenced by srvAppLayer::execMessage().

6.14.2.4 BYTE stateVector_type::lastFuncId

Definition at line 38 of file srvAppLayer.h.

Referenced by srvAppLayer::execMessage().

The documentation for this struct was generated from the following file:

- src/srvAppLayer/srvAppLayer.h

Chapter 7

File Documentation

7.1 src/_auto_config.h File Reference

Defines

- #define HAVE_ARPA_INET_H 1
- #define HAVE_INTTYPES_H 1
- #define HAVE_MALLOC 1
- #define HAVE_MEMORY_H 1
- #define HAVE_NETINET_IN_H 1
- #define HAVE_REALLOC 1
- #define HAVE_SOCKET 1
- #define HAVE_STDBOOL_H 1
- #define HAVE_STDDEF_H 1
- #define HAVE_STDINT_H 1
- #define HAVE_STDLIB_H 1
- #define HAVE_STRINGS_H 1
- #define HAVE_STRING_H 1
- #define HAVE_SYS_SOCKET_H 1
- #define HAVE_SYS_STAT_H 1
- #define HAVE_SYS_TYPES_H 1
- #define HAVE_UNISTD_H 1
- #define HAVE__BOOL 1
- #define PACKAGE "r168"
- #define PACKAGE_BUGREPORT "nosenko@rec-etu.com"
- #define PACKAGE_NAME "src/main.cpp"
- #define PACKAGE_STRING "src/main.cpp 0.1"
- #define PACKAGE_TARNAME "src-main-cpp"
- #define PACKAGE_URL ""
- #define PACKAGE_VERSION "0.1"
- #define STDC_HEADERS 1
- #define VERSION "0.1"

7.1.1 Define Documentation

7.1.1.1 #define HAVE_ARPA_INET_H 1

Definition at line 5 of file _auto_config.h.

7.1.1.2 #define HAVE_INTTYPES_H 1

Definition at line 8 of file _auto_config.h.

7.1.1.3 #define HAVE_MALLOC 1

Definition at line 12 of file _auto_config.h.

7.1.1.4 #define HAVE_MEMORY_H 1

Definition at line 15 of file _auto_config.h.

7.1.1.5 #define HAVE_NETINET_IN_H 1

Definition at line 18 of file _auto_config.h.

7.1.1.6 #define HAVE_REALLOC 1

Definition at line 22 of file _auto_config.h.

7.1.1.7 #define HAVE_SOCKET 1

Definition at line 25 of file _auto_config.h.

7.1.1.8 #define HAVE_STDBOOL_H 1

Definition at line 28 of file _auto_config.h.

7.1.1.9 #define HAVE_STDDEF_H 1

Definition at line 31 of file _auto_config.h.

7.1.1.10 #define HAVE_STDINT_H 1

Definition at line 34 of file _auto_config.h.

7.1.1.11 #define HAVE_STDLIB_H 1

Definition at line 37 of file _auto_config.h.

7.1.1.12 #define HAVE_STRINGS_H 1

Definition at line 40 of file _auto_config.h.

7.1.1.13 #define HAVE_STRING_H 1

Definition at line 43 of file _auto_config.h.

7.1.1.14 #define HAVE_SYS_SOCKET_H 1

Definition at line 46 of file _auto_config.h.

7.1.1.15 #define HAVE_SYS_STAT_H 1

Definition at line 49 of file _auto_config.h.

7.1.1.16 #define HAVE_SYS_TYPES_H 1

Definition at line 52 of file _auto_config.h.

7.1.1.17 #define HAVE_UNISTD_H 1

Definition at line 55 of file _auto_config.h.

7.1.1.18 #define HAVE__BOOL 1

Definition at line 58 of file _auto_config.h.

7.1.1.19 #define PACKAGE "r168"

Definition at line 61 of file _auto_config.h.

7.1.1.20 #define PACKAGE_BUGREPORT "nosenko@rec-etu.com"

Definition at line 64 of file _auto_config.h.

7.1.1.21 #define PACKAGE_NAME "src/main.cpp"

Definition at line 67 of file _auto_config.h.

7.1.1.22 #define PACKAGE_STRING "src/main.cpp 0.1"

Definition at line 70 of file _auto_config.h.

7.1.1.23 #define PACKAGE_TARNAME "src-main-cpp"

Definition at line 73 of file _auto_config.h.

7.1.1.24 #define PACKAGE_URL ""

Definition at line 76 of file _auto_config.h.

7.1.1.25 #define PACKAGE_VERSION "0.1"

Definition at line 79 of file _auto_config.h.

7.1.1.26 #define STDC_HEADERS 1

Definition at line 82 of file _auto_config.h.

7.1.1.27 #define VERSION "0.1"

Definition at line 85 of file _auto_config.h.

7.2 src/_global.cpp File Reference

Global environment.

```
#include <netinet/in.h>
#include <queue>
#include "../../rcsLib/ortsTypes/ortsTypes.h"
#include "buffer/ssBuffer.h"
#include "../../udp_port/udp_port.h"
#include "../../rcsLib/rcsCmd/rcsCmd.h"
#include "ICAppLayer/FunctionNode/param_desc.h"
#include "ICAppLayer/FunctionNode/FunctionNode.h"
```

```
#include "ICAppLayer/ICAppLayer.h"
```

Variables

- int verbose_level = 0

  Debug detail level printing.

- bool listen_mode = false

  needless variable.

- bool awaitingPattern_mode = false

  needless variable.

- bool pattern_found = false

  needless variable.

- char patternFile [255] = {0}

  needless variable.

- char reactionFile [255] = {0}

  needless variable.

- char dataFile [255] = {0}

  needless variable.

- char if_name [255] = {0}

  needless variable.

- bool AppTerminated = false

  Programm termination signal.

- char eq_ip_addr [255] = {0}

  Equipment IP address.

- WORD wUdp = 0

  Server udp port number for communicate with client.

- WORD eq_udp_listen_port = 0

  Server udp port number for listen an equipment.

- WORD eq_udp_sending_port = 0

  Server udp port number for sending into equipment.

- in_addr equipAddr

  Storage for in_addr of equipment.

7.2.1 Detailed Description

Global environment.

Author

Vladimir A. Nosenko (nosenko@ieee.org)

Date

December, 2010

Copyright (c) 2010 Vladimir A.Nosenko.

The license and distribution terms for this file may be found in the file LICENSE in this distribution

Definition in file _global.cpp.

7.2.2 Variable Documentation

7.2.2.1 int verbose_level = 0

Debug detail level printing.

Definition at line 26 of file _global.cpp.

Referenced by appInit(), process_cmdLine(), srvAppLayer::StopListening(), and term_-handler().

7.2.2.2 bool listen_mode = false

needless variable.

Todo

Need to be deleted.

Definition at line 29 of file _global.cpp.

7.2.2.3 bool awaitingPattern_mode = false

needless variable.

Todo

Need to be deleted.

Definition at line 30 of file _global.cpp.

7.2.2.4 bool pattern_found = false

needless variable.

Todo

Need to be deleted.

Definition at line 31 of file _global.cpp.

7.2.2.5 char patternFile[255] = {0}

needless variable.

Todo

Need to be deleted.

Definition at line 32 of file _global.cpp.

7.2.2.6 char reactionFile[255] = {0}

needless variable.

Todo

Need to be deleted.

Definition at line 33 of file _global.cpp.

7.2.2.7 char dataFile[255] = {0}

needless variable.

Todo

Need to be deleted.

Definition at line 34 of file _global.cpp.

7.2.2.8 char if_name[255] = {0}

needless variable.

Todo

Need to be deleted.

Definition at line 35 of file _global.cpp.

7.2.2.9 bool AppTerminated = false

Programm termination signal.

All processes need to finish own job.

Definition at line 38 of file _global.cpp.

Referenced by main().

7.2.2.10 char eq_ip_addr[255] = {0}

Equipment IP address.

Definition at line 40 of file _global.cpp.

Referenced by process_cmdLine().

7.2.2.11 WORD wUdp = 0

Server udp port number for communicate with client.

Definition at line 43 of file _global.cpp.

Referenced by main(), process_cmdLine(), srvAppLayer::sendResult(), udpListenerThread(), and udpSenderThread().

7.2.2.12 WORD eq_udp_listen_port = 0

Server udp port number for listen an equipment.

Definition at line 44 of file _global.cpp.

Referenced by main(), process_cmdLine(), and srvAppLayer::StartListening().

7.2.2.13 WORD eq_udp_sending_port = 0

Server udp port number for sending into equipment.

Definition at line 45 of file _global.cpp.

Referenced by main(), and process_cmdLine().

7.2.2.14 in_addr equipAddr

Storage for in_addr of equipment.

Definition at line 47 of file _global.cpp.

Referenced by main(), and process_cmdLine().

7.3 src/_global.h File Reference

Global environment interface header.

Variables

- bool AppTerminated

  Programm termination signal.

- bool awaitingPattern_mode

  needless variable.

- bool pattern_found

  needless variable.

- int verbose_level

  Debug detail level printing.

- bool listen_mode

  needless variable.

- char eq_ip_addr [255]

  Equipment IP address.

- char dataFile [255]

  needless variable.

- char if_name [255]

  needless variable.

- char patternFile [255]

  needless variable.

- char reactionFile [255]

  needless variable.

- WORD eq_udp_listen_port

  Server udp port number for listen an equipment.

- WORD eq_udp_sending_port

  Server udp port number for sending into equipment.

- in_addr equipAddr

  Storage for in_addr of equipment.

- WORD wUdp

  Server udp port number for communicate with client.

7.3.1 Detailed Description

Global environment interface header.

Author

Vladimir A. Nosenko (nosenko@ieee.org)

Date

December, 2010

Copyright (c) 2010 Vladimir A.Nosenko.

The license and distribution terms for this file may be found in the file LICENSE in this distribution

Definition in file _global.h.

7.3.2 Variable Documentation

7.3.2.1 bool AppTerminated

Programm termination signal.

All processes need to finish own job.

Definition at line 38 of file _global.cpp.

Referenced by main().

7.3.2.2 bool awaitingPattern_mode

needless variable.

Todo

Need to be deleted.

Definition at line 30 of file _global.cpp.

7.3.2.3 bool pattern_found

needless variable.

Todo

Need to be deleted.

Definition at line 31 of file _global.cpp.

7.3.2.4 int verbose_level

Debug detail level printing.

Definition at line 26 of file _global.cpp.

Referenced by appInit(), process_cmdLine(), srvAppLayer::StopListening(), and term_-handler().

7.3.2.5 bool listen_mode

needless variable.

Todo

Need to be deleted.

Definition at line 29 of file _global.cpp.

7.3.2.6 char eq_ip_addr[255]

Equipment IP address.

Definition at line 40 of file _global.cpp.

Referenced by process_cmdLine().

7.3.2.7 char dataFile[255]

needless variable.

Todo

Need to be deleted.

Definition at line 34 of file _global.cpp.

7.3.2.8 char if_name[255]

needless variable.

Todo

Need to be deleted.

Definition at line 35 of file _global.cpp.

7.3.2.9 char patternFile[255]

needless variable.

Todo

Need to be deleted.

Definition at line 32 of file _global.cpp.

7.3.2.10 char reactionFile[255]

needless variable.

Todo

Need to be deleted.

Definition at line 33 of file _global.cpp.

7.3.2.11 WORD eq_udp_listen_port

Server udp port number for listen an equipment.

Definition at line 44 of file _global.cpp.

Referenced by main(), process_cmdLine(), and srvAppLayer::StartListening().

7.3.2.12 WORD eq_udp_sending_port

Server udp port number for sending into equipment.

Definition at line 45 of file _global.cpp.

Referenced by main(), and process_cmdLine().

7.3.2.13 in_addr equipAddr

Storage for in_addr of equipment.

Definition at line 47 of file _global.cpp.

Referenced by main(), and process_cmdLine().

7.3.2.14 WORD wUdp

Server udp port number for communicate with client.

Definition at line 43 of file _global.cpp.

Referenced by main(), process_cmdLine(), srvAppLayer::sendResult(), udpListenerThread(), and udpSenderThread().

7.4 src/arg_parser/carg_parser.cpp File Reference

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "carg_parser.h"
```

Functions

- char ap_resize_buffer (void ∗buf, const int min_size)
- char push_back_record (Arg_parser ∗ap, const int code, const char ∗argument)
- char add_error (Arg_parser ∗ap, const char ∗msg)
- void free_data (Arg_parser ∗ap)
- char parse_long_option (Arg_parser ∗ap, const char ∗const opt, const char ∗const arg, const ap_Option options[ ], int ∗argindp)
- char parse_short_option (Arg_parser ∗ap, const char ∗const opt, const char ∗const arg, const ap_Option options[ ], int ∗argindp)
- char ap_init (Arg_parser ∗ap, const int argc, const char ∗const argv[ ], const ap_-Option options[ ], const char in_order)
- void ap_free (Arg_parser ∗ap)
- const char ∗ ap_error (const Arg_parser ∗ap)
- int ap_arguments (const Arg_parser ∗ap)
- int ap_code (const Arg_parser ∗ap, const int i)
- const char ∗ ap_argument (const Arg_parser ∗ap, const int i)

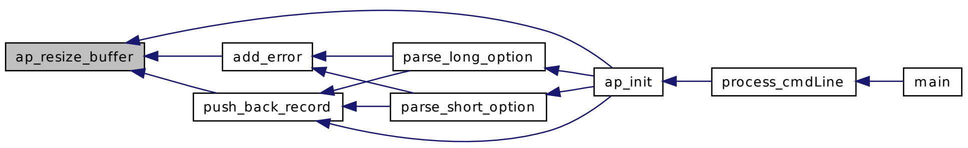
7.4.1 Function Documentation

7.4.1.1 char ap_resize_buffer (void ∗ *buf,* const int *min_size*)

Definition at line 26 of file carg_parser.cpp.

Referenced by add_error(), ap_init(), and push_back_record().

Here is the caller graph for this function:



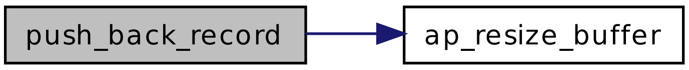
7.4.1.2 char push_back_record (Arg_parser ∗ *ap*, const int *code*, const char ∗ *argument*)

Definition at line 37 of file carg_parser.cpp.

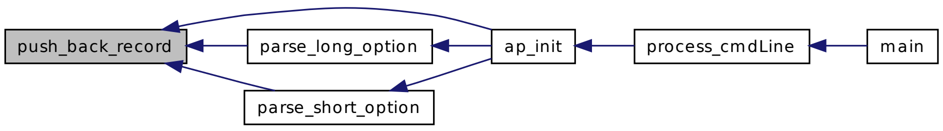
References ap_resize_buffer(), ap_Record::argument, ap_Record::code, Arg_parser::data, and Arg_parser::data_size.

Referenced by ap_init(), parse_long_option(), and parse_short_option().

Here is the call graph for this function:



Here is the caller graph for this function:



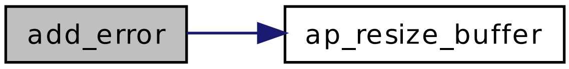
7.4.1.3 char add_error (Arg_parser ∗ *ap*, const char ∗ *msg*)

Definition at line 54 of file carg_parser.cpp.

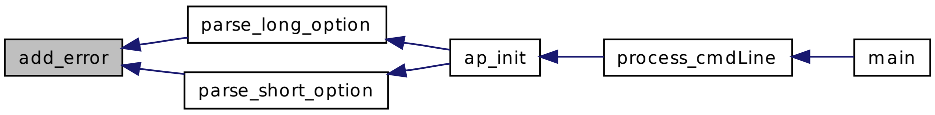
References ap_resize_buffer(), Arg_parser::error, and Arg_parser::error_size.

Referenced by parse_long_option(), and parse_short_option().

Here is the call graph for this function:



Here is the caller graph for this function:



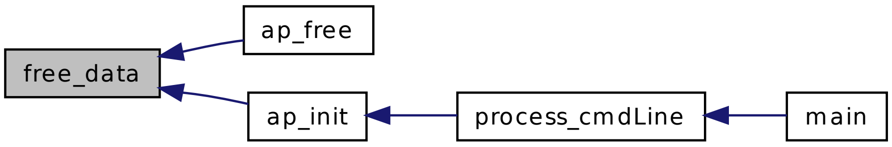
7.4.1.4 void free_data (Arg_parser * *ap*)

Definition at line 65 of file carg_parser.cpp.

References ap_Record::argument, Arg_parser::data, and Arg_parser::data_size.

Referenced by ap_free(), and ap_init().

Here is the caller graph for this function:



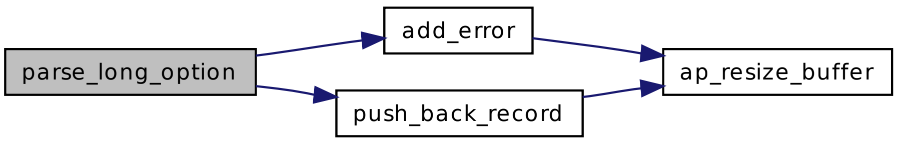
7.4.1.5 char parse_long_option (Arg_parser ∗ *ap*, const char ∗const *opt*, const char ∗const *arg*, const ap_Option *options[ ]*, int ∗ *argindp*)

Definition at line 74 of file carg_parser.cpp.

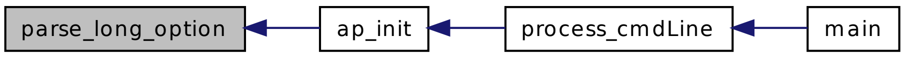
References add_error(), ap_no, ap_yes, ap_Option::code, and push_back_record().

Referenced by ap_init().

Here is the call graph for this function:



Here is the caller graph for this function:



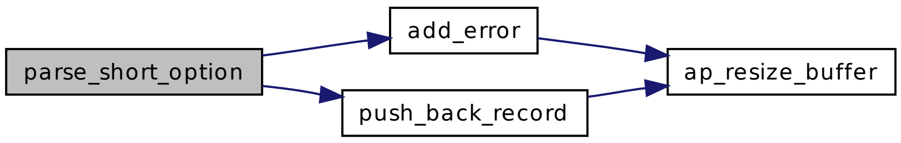
7.4.1.6 char parse_short_option (Arg_parser ∗ *ap*, const char ∗const *opt*, const char ∗const *arg*, const ap_Option *options[ ]*, int ∗ *argindp*)

Definition at line 146 of file carg_parser.cpp.

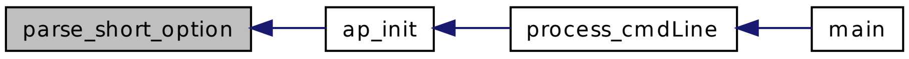
References add_error(), ap_no, ap_yes, ap_Option::code, and push_back_record().

Referenced by ap_init().

Here is the call graph for this function:



Here is the caller graph for this function:



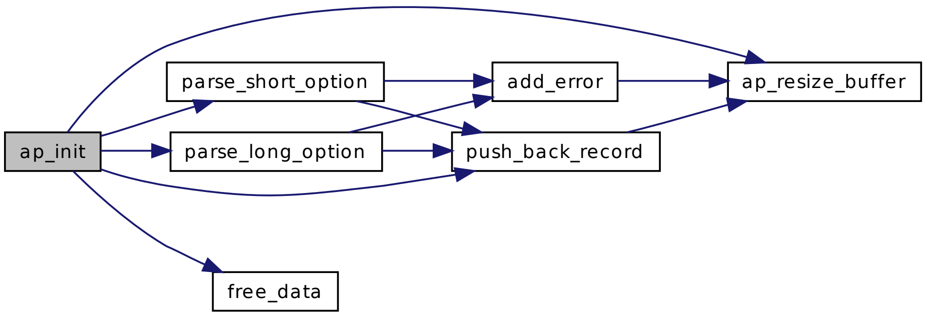
7.4.1.7 char ap_init (Arg_parser ∗ *ap,* const int *argc,* const char ∗const *argv[ ],* const ap_Option *options[ ],* const char *in_order*)

Definition at line 194 of file carg_parser.cpp.

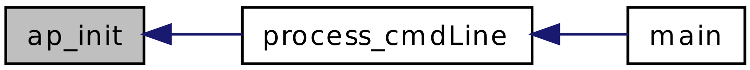
References ap_resize_buffer(), Arg_parser::data, Arg_parser::data_size, Arg_parser::error, Arg_parser::error_size, free_data(), parse_long_option(), parse_short_option(), and push_-back_record().

Referenced by process_cmdLine().

Here is the call graph for this function:



Here is the caller graph for this function:

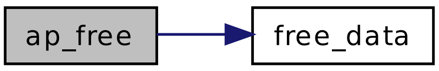


7.4.1.8 void ap_free (Arg_parser ∗ *ap*)

Definition at line 253 of file carg_parser.cpp.

References Arg_parser::error, Arg_parser::error_size, and free_data().

Here is the call graph for this function:



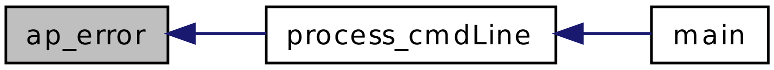
7.4.1.9 const char∗ ap_error (const Arg_parser ∗ *ap*)

Definition at line 261 of file carg_parser.cpp.

References Arg_parser::error.

Referenced by process_cmdLine().

Here is the caller graph for this function:



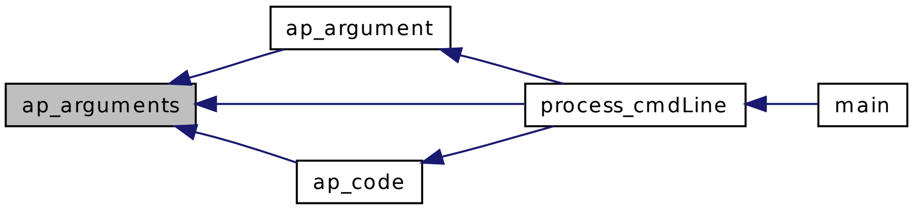
7.4.1.10 int ap_arguments (const Arg_parser ∗ *ap*)

Definition at line 264 of file carg_parser.cpp.

References Arg_parser::data_size.

Referenced by ap_argument(), ap_code(), and process_cmdLine().

Here is the caller graph for this function:



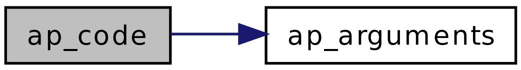
7.4.1.11 int ap_code (const Arg_parser ∗ *ap,* const int *i*)

Definition at line 267 of file carg_parser.cpp.

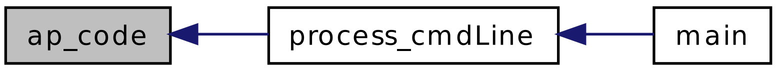
References ap_arguments(), ap_Record::code, and Arg_parser::data.

Referenced by process_cmdLine().

Here is the call graph for this function:



Here is the caller graph for this function:



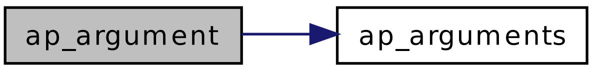
7.4.1.12 const char∗ ap_argument (const Arg_parser ∗ *ap*, const int *i*)

Definition at line 274 of file carg_parser.cpp.

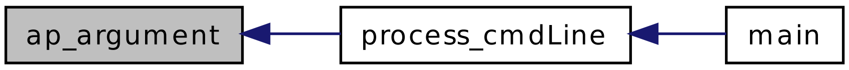
References ap_arguments(), ap_Record::argument, and Arg_parser::data.

Referenced by process_cmdLine().

Here is the call graph for this function:



Here is the caller graph for this function:



7.5 src/arg_parser/carg_parser.h File Reference

Classes

- struct ap_Option
- struct ap_Record
- struct Arg_parser

Enumerations

- enum ap_Has_arg { ap_no, ap_yes, ap_maybe }

Functions

- char ap_init (Arg_parser ∗ap, const int argc, const char ∗const argv[ ], const ap_-Option options[ ], const char in_order)
- void ap_free (Arg_parser ∗ap)
- const char ∗ ap_error (const Arg_parser ∗ap)
- int ap_arguments (const Arg_parser ∗ap)
- int ap_code (const Arg_parser ∗ap, const int i)
- const char ∗ ap_argument (const Arg_parser ∗ap, const int i)

7.5.1 Enumeration Type Documentation

7.5.1.1 enum ap_Has_arg

Enumerator:

ap_no

ap_yes

ap_maybe

Definition at line 42 of file carg_parser.h.

7.5.2 Function Documentation

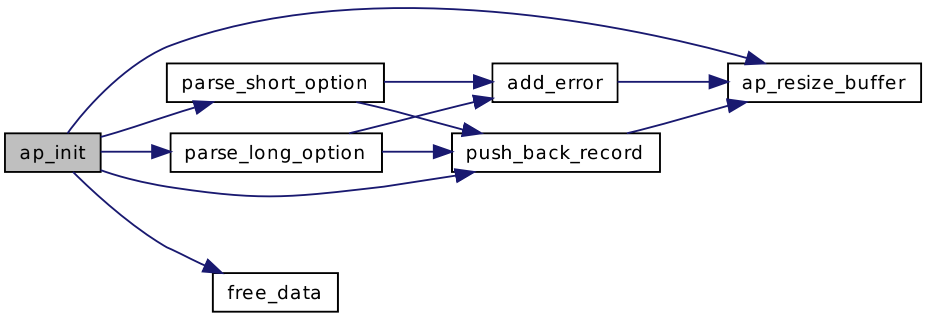
7.5.2.1 char ap_init (Arg_parser ∗ *ap*, const int *argc*, const char ∗const *argv[ ]*, const ap_Option *options[ ]*, const char *in_order*)

Definition at line 194 of file carg_parser.cpp.

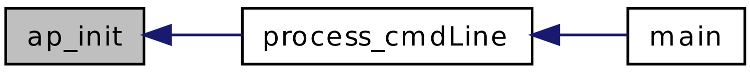
References ap_resize_buffer(), Arg_parser::data, Arg_parser::data_size, Arg_parser::error, Arg_parser::error_size, free_data(), parse_long_option(), parse_short_option(), and push_-back_record().

Referenced by process_cmdLine().

Here is the call graph for this function:



Here is the caller graph for this function:

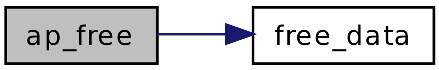


7.5.2.2 void ap_free (Arg_parser ∗ *ap*)

Definition at line 253 of file carg_parser.cpp.

References Arg_parser::error, Arg_parser::error_size, and free_data().

Here is the call graph for this function:



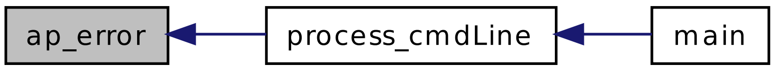
7.5.2.3 const char∗ ap_error (const Arg_parser ∗ *ap*)

Definition at line 261 of file carg_parser.cpp.

References Arg_parser::error.

Referenced by process_cmdLine().

Here is the caller graph for this function:



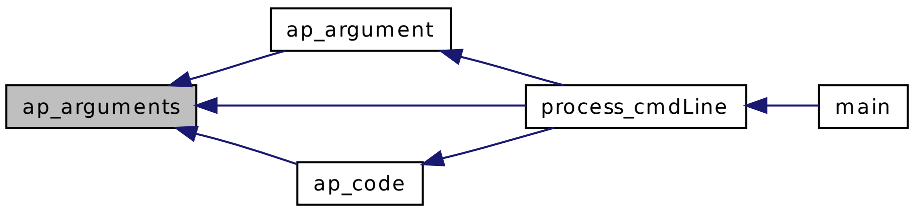
7.5.2.4 int ap_arguments (const Arg_parser ∗ *ap*)

Definition at line 264 of file carg_parser.cpp.

References Arg_parser::data_size.

Referenced by ap_argument(), ap_code(), and process_cmdLine().

Here is the caller graph for this function:



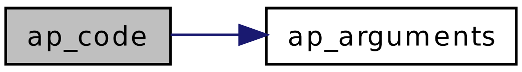
7.5.2.5 int ap_code (const Arg_parser ∗ *ap*, const int *i*)

Definition at line 267 of file carg_parser.cpp.

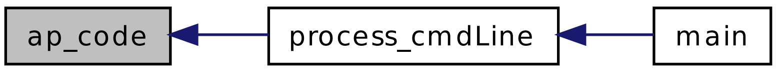
References ap_arguments(), ap_Record::code, and Arg_parser::data.

Referenced by process_cmdLine().

Here is the call graph for this function:



Here is the caller graph for this function:



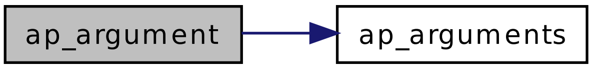
7.5.2.6 const char∗ ap_argument (const Arg_parser ∗ *ap*, const int *i*)

Definition at line 274 of file carg_parser.cpp.

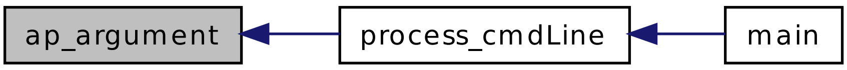
References ap_arguments(), ap_Record::argument, and Arg_parser::data.

Referenced by process_cmdLine().

Here is the call graph for this function:



Here is the caller graph for this function:



7.6 src/auto_config.h File Reference

Defines

- #define HAVE_ARPA_INET_H 1
- #define HAVE_INTTYPES_H 1
- #define HAVE_MALLOC 1
- #define HAVE_MEMORY_H 1
- #define HAVE_NETINET_IN_H 1
- #define HAVE_REALLOC 1
- #define HAVE_SOCKET 1
- #define HAVE_STDBOOL_H 1
- #define HAVE_STDDEF_H 1
- #define HAVE_STDINT_H 1
- #define HAVE_STDLIB_H 1

- #define HAVE_STRINGS_H 1
- #define HAVE_STRING_H 1
- #define HAVE_SYS_SOCKET_H 1
- #define HAVE_SYS_STAT_H 1
- #define HAVE_SYS_TYPES_H 1
- #define HAVE_UNISTD_H 1
- #define HAVE__BOOL 1
- #define PACKAGE "r168"
- #define PACKAGE_BUGREPORT "nosenko@rec-etu.com"
- #define PACKAGE_NAME "src/main.cpp"
- #define PACKAGE_STRING "src/main.cpp 0.1"
- #define PACKAGE_TARNAME "src-main-cpp"
- #define PACKAGE_URL ""
- #define PACKAGE_VERSION "0.1"
- #define STDC_HEADERS 1
- #define VERSION "0.1"

7.6.1 Define Documentation

7.6.1.1 #define HAVE_ARPA_INET_H 1

Definition at line 5 of file auto_config.h.

7.6.1.2 #define HAVE_INTTYPES_H 1

Definition at line 8 of file auto_config.h.

7.6.1.3 #define HAVE_MALLOC 1

Definition at line 12 of file auto_config.h.

7.6.1.4 #define HAVE_MEMORY_H 1

Definition at line 15 of file auto_config.h.

7.6.1.5 #define HAVE_NETINET_IN_H 1

Definition at line 18 of file auto_config.h.

7.6.1.6 #define HAVE_REALLOC 1

Definition at line 22 of file auto_config.h.

7.6.1.7 #define HAVE_SOCKET 1

Definition at line 25 of file auto_config.h.

7.6.1.8 #define HAVE_STDBOOL_H 1

Definition at line 28 of file auto_config.h.

7.6.1.9 #define HAVE_STDDEF_H 1

Definition at line 31 of file auto_config.h.

7.6.1.10 #define HAVE_STDINT_H 1

Definition at line 34 of file auto_config.h.

7.6.1.11 #define HAVE_STDLIB_H 1

Definition at line 37 of file auto_config.h.

7.6.1.12 #define HAVE_STRINGS_H 1

Definition at line 40 of file auto_config.h.

7.6.1.13 #define HAVE_STRING_H 1

Definition at line 43 of file auto_config.h.

7.6.1.14 #define HAVE_SYS_SOCKET_H 1

Definition at line 46 of file auto_config.h.

7.6.1.15 #define HAVE_SYS_STAT_H 1

Definition at line 49 of file auto_config.h.

7.6.1.16 #define HAVE_SYS_TYPES_H 1

Definition at line 52 of file auto_config.h.

7.6.1.17 #define HAVE_UNISTD_H 1

Definition at line 55 of file auto_config.h.

7.6.1.18 #define HAVE__BOOL 1

Definition at line 58 of file auto_config.h.

7.6.1.19 #define PACKAGE "r168"

Definition at line 61 of file auto_config.h.

7.6.1.20 #define PACKAGE_BUGREPORT "nosenko@rec-etu.com"

Definition at line 64 of file auto_config.h.

7.6.1.21 #define PACKAGE_NAME "src/main.cpp"

Definition at line 67 of file auto_config.h.

7.6.1.22 #define PACKAGE_STRING "src/main.cpp 0.1"

Definition at line 70 of file auto_config.h.

7.6.1.23 #define PACKAGE_TARNAME "src-main-cpp"

Definition at line 73 of file auto_config.h.

7.6.1.24 #define PACKAGE_URL ""

Definition at line 76 of file auto_config.h.

7.6.1.25 #define PACKAGE_VERSION "0.1"

Definition at line 79 of file auto_config.h.

7.6.1.26 #define STDC_HEADERS 1

Definition at line 82 of file auto_config.h.

7.6.1.27 #define VERSION "0.1"

Definition at line 85 of file auto_config.h.

7.7 src/buffer/buffer.cpp File Reference

Class buffer implementation.

```
#include <string.h>
#include <stdio.h>
#include "../../../rcsLib/ortsTypes/ortsTypes.h"
#include "buffer.h"
```

7.7.1 Detailed Description

Class buffer implementation.

Author

Vladimir A. Nosenko (nosenko@ieee.org)

Date

December, 2010

Copyright (c) 2010 Vladimir A.Nosenko.

The license and distribution terms for this file may be found in the file LICENSE in this distribution

Definition in file buffer.cpp.

7.8 src/buffer/buffer.h File Reference

Class buffer interface header.

Classes

- class buffer

  Simple queue of bytes.

7.8.1 Detailed Description

Class buffer interface header.

Author

Vladimir A. Nosenko (nosenko@ieee.org)

Date

December, 2010

Definition in file buffer.h.

7.9 src/buffer/ssBuffer.cpp File Reference

Class ssBuffer implementation.

```
#include <string.h>
#include <stdio.h>
#include <deque>
#include <netinet/in.h>
#include <arpa/inet.h>
#include "../../../rcsLib/ortsTypes/ortsTypes.h"
#include "ssBuffer.h"
```

7.9.1 Detailed Description

Class ssBuffer implementation.

Author

Vladimir A. Nosenko (nosenko@ieee.org)

Date

December, 2010

Definition in file ssBuffer.cpp.

7.10 src/buffer/ssBuffer.h File Reference

Class ssBuffer interface header.

Classes

- struct ssBlock

    ssBuffer list entry.

- class ssBuffer

    list (deque) implementaion for storing ssBlock elements.

Typedefs

- typedef struct ssBlock ssBlock

7.10.1 Detailed Description

Class ssBuffer interface header.

Author

Vladimir A. Nosenko (nosenko@ieee.org)

Date

December, 2010

Copyright (c) 2010 Vladimir A.Nosenko.

The license and distribution terms for this file may be found in the file LICENSE in this distribution

Definition in file ssBuffer.h.

7.10.2 Typedef Documentation

7.10.2.1 typedef struct ssBlock ssBlock

7.11 src/config.h File Reference

7.12 src/console_out.cpp File Reference

aided functions to process_cmdLine

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "auto_config.h"
#include "./arg_parser/carg_parser.h"
```

Defines

- #define REVISION 0

  programm revision number

Functions

- void show_help (const char verbose)

  Show cmdline help information (--help)

- void show_version ()

  Show programm version information (--version)

- void show_error (const char ∗msg, const int errcode, const char help)

  Show cmdLine parser error.

- void internal_error (const char ∗msg)

  Show cmdLine internal error.

- const char ∗ optname (const int code, const ap_Option options[ ])

  Convert code with option from cmdLine argument to char buffer.

Variables

- char PROGVERSION [255] = "0.1"

  version of programm

- char Program_name [255] = " "

  name of programm

- char program_name [255] = "ss_Service"

  filename of programm

- char program_year [255] = "2010"

  copyright year

7.12.1 Detailed Description

aided functions to process_cmdLine

Author

Vladimir A. Nosenko (`nosenko@ieee.org`)

Date

December, 2010

Copyright (c) 2010 Vladimir A.Nosenko.

The license and distribution terms for this file may be found in the file LICENSE in this distribution

Definition in file console_out.cpp.

7.12.2 Define Documentation

7.12.2.1 #define REVISION 0

programm revision number

- revision by default is 0

Definition at line 24 of file console_out.cpp.

Referenced by show_version().

7.12.3 Function Documentation

7.12.3.1 void show_help (const char *verbose*)

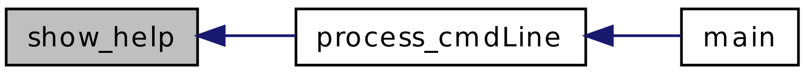
Show cmdline help information (--help)

Definition at line 36 of file console_out.cpp.

References program_name, and Program_name.

Referenced by process_cmdLine().

Here is the caller graph for this function:



7.12.3.2 void show_version ( )

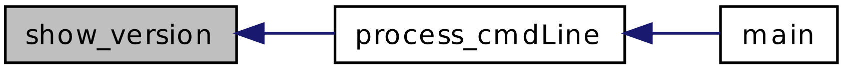
Show programm version information (--version)

Definition at line 58 of file console_out.cpp.

References Program_name, PROGVERSION, and REVISION.

Referenced by process_cmdLine().

Here is the caller graph for this function:



7.12.3.3 void show_error (const char ∗ *msg,* const int *errcode,* const char *help*)

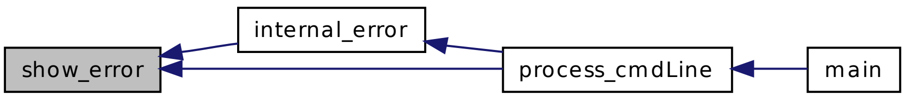
Show cmdLine parser error.

Definition at line 72 of file console_out.cpp.

References program_name.

Referenced by internal_error(), and process_cmdLine().

Here is the caller graph for this function:



7.12.3.4 void internal_error (const char ∗ *msg*)

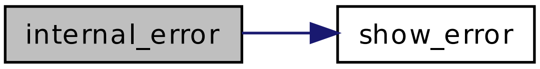
Show cmdLine internal error.

Definition at line 86 of file console_out.cpp.

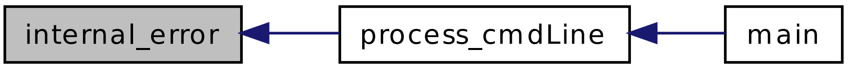
References show_error().

Referenced by process_cmdLine().

Here is the call graph for this function:



Here is the caller graph for this function:



7.12.3.5 const char∗ optname (const int *code,* const ap_Option *options[]*)

Convert code with option from cmdLine argument to char buffer.

Definition at line 97 of file console_out.cpp.

References ap_Option::code, and ap_Option::name.

7.12.4 Variable Documentation

7.12.4.1 char PROGVERSION[255] = "0.1"

version of programm

Definition at line 27 of file console_out.cpp.

Referenced by show_version().

7.12.4.2 char Program_name[255] = " "

name of programm

Definition at line 28 of file console_out.cpp.

Referenced by show_help(), and show_version().

7.12.4.3 char program_name[255] = "ss_Service"

filename of programm

Definition at line 29 of file console_out.cpp.

Referenced by show_error(), and show_help().

7.12.4.4 char program_year[255] = "2010"

copyright year

Definition at line 30 of file console_out.cpp.

7.13 src/console_out.h File Reference

aided functions interface header

Functions

- void show_help (const char verbose)

  Show cmdline help information (--help)

- void show_version ()

  Show programm version information (--version)

- void show_error (const char ∗msg, const int errcode, const char help)

  Show cmdLine parser error.

- void internal_error (const char ∗msg)

  Show cmdLine internal error.

- const char ∗ optname (const int code, const ap_Option options[])

  Convert code with option from cmdLine argument to char buffer.

Variables

- char PROGVERSION []

  version of programm

- char Program_name []

  name of programm

- char program_name []

filename of programm

- char program_year []

  copyright year

7.13.1 Detailed Description

aided functions interface header

Author

Vladimir A. Nosenko (nosenko@ieee.org)

Date

December, 2010

Copyright (c) 2010 Vladimir A.Nosenko.

The license and distribution terms for this file may be found in the file LICENSE in this distribution

Definition in file console_out.h.

7.13.2 Function Documentation

7.13.2.1 void show_help (const char *verbose*)

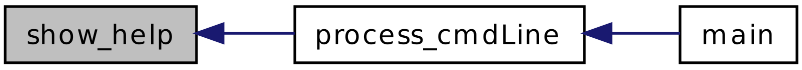
Show cmdline help information (--help)

Definition at line 36 of file console_out.cpp.

References program_name, and Program_name.

Referenced by process_cmdLine().

Here is the caller graph for this function:



7.13.2.2 void show_version ( )

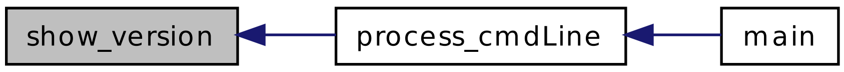
Show programm version information (--version)

Definition at line 58 of file console_out.cpp.

References Program_name, PROGVERSION, and REVISION.

Referenced by process_cmdLine().

Here is the caller graph for this function:



7.13.2.3 void show_error (const char * *msg,* const int *errcode,* const char *help*)

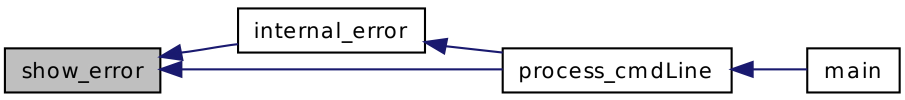
Show cmdLine parser error.

Definition at line 72 of file console_out.cpp.

References program_name.

Referenced by internal_error(), and process_cmdLine().

Here is the caller graph for this function:



7.13.2.4 void internal_error (const char * *msg*)

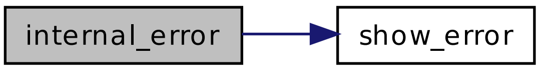
Show cmdLine internal error.

Definition at line 86 of file console_out.cpp.

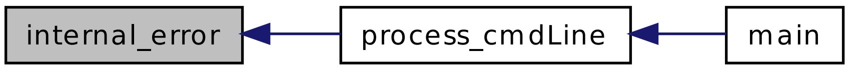
References show_error().

Referenced by process_cmdLine().

Here is the call graph for this function:



Here is the caller graph for this function:



7.13.2.5 const char∗ optname (const int *code*, const ap_Option *options[]*)

Convert code with option from cmdLine argument to char buffer.

Definition at line 97 of file console_out.cpp.

References ap_Option::code, and ap_Option::name.

7.13.3 Variable Documentation

7.13.3.1 char PROGVERSION[ ]

version of programm

Definition at line 27 of file console_out.cpp.

Referenced by show_version().

7.13.3.2 char Program_name[ ]

name of programm

Definition at line 28 of file console_out.cpp.

Referenced by show_help(), and show_version().

7.13.3.3 char program_name[ ]

filename of programm

Definition at line 29 of file console_out.cpp.

Referenced by show_error(), and show_help().

7.13.3.4 char program_year[ ]

copyright year

Definition at line 30 of file console_out.cpp.

7.14 src/deqUdp/deqUdp.cpp File Reference

Class deqUdp implementation.

```
#include <arpa/inet.h>
#include <deque>
#include "../../../rcsLib/ortsTypes/ortsTypes.h"
#include "../buffer/ssBuffer.h"
#include "../../../udp_port/udp_port.h"
#include "deqUdp.h"
```

7.14.1 Detailed Description

Class deqUdp implementation.

Author

Vladimir A. Nosenko (nosenko@ieee.org)

Date

December, 2010

Copyright (c) 2010 Vladimir A.Nosenko.

The license and distribution terms for this file may be found in the file LICENSE in this distribution

Definition in file deqUdp.cpp.

7.15 src/deqUdp/deqUdp.h File Reference

Class deqUdp interface header.

Classes

- class deqUdp

  udp communications (based on udp_port) with queues for listening and sending

7.15.1 Detailed Description

Class deqUdp interface header.

Author

Vladimir A. Nosenko (nosenko@ieee.org)

Date

December, 2010

Copyright (c) 2010 Vladimir A.Nosenko.

The license and distribution terms for this file may be found in the file LICENSE in this distribution

Definition in file deqUdp.h.

7.16 src/functions/commonFuncsMgr.cpp File Reference

Class commonFuncsMgr interface header.

```
#include <pthread.h>
#include <netinet/in.h>
#include <queue>
#include "../../rcsLib/ortsTypes/ortsTypes.h"
#include "../buffer/ssBuffer.h"
#include "../../rcsLib/rcsCmd/rcsCmd.h"
#include "../../udp_port/udp_port.h"
#include "../srvAppLayer/functionNode/param_desc.h"
#include "../srvAppLayer/functionNode/functionNode.h"
#include "../srvAppLayer/srvAppLayer.h"
#include "commonFuncsMgr.h"
```

```
#include "functions.h"
```

Functions

- void ∗ equipListenPolling (void ∗user)

  Thread to polling listen udp for equipment data.

7.16.1 Detailed Description

Class commonFuncsMgr interface header.

Author

Vladimir A. Nosenko (`nosenko@ieee.org`)

Date

December, 2010

Definition in file commonFuncsMgr.cpp.

7.16.2 Function Documentation

7.16.2.1 void∗ equipListenPolling (void ∗ *user*)

Thread to polling listen udp for equipment data.

Calls equipListenProcessing for decoding data received from equipment

Return values

user	

Todo

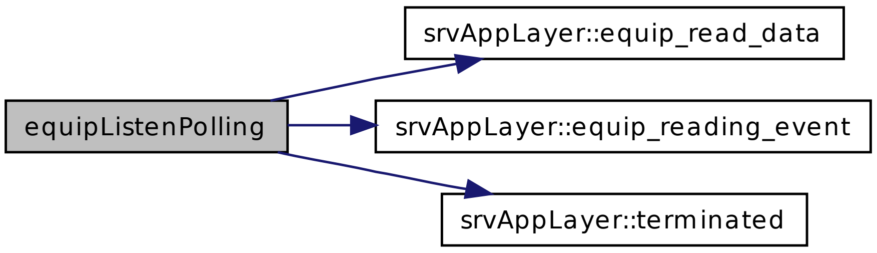
Listening equipment answer - status vector:

Definition at line 35 of file commonFuncsMgr.cpp.

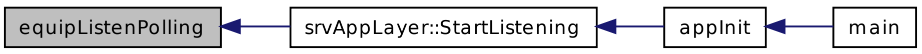
References app, srvAppLayer::equip_read_data(), srvAppLayer::equip_reading_event(), and srvAppLayer::terminated().

Referenced by srvAppLayer::StartListening().

Here is the call graph for this function:


srvAppLayer::equip_read_data
equipListenPolling
srvAppLayer::equip_reading_event
srvAppLayer::terminated

Here is the caller graph for this function:



7.17 src/functions/commonFuncsMgr.h File Reference

Class commonFuncsMgr interface header.

Classes

- class commonFuncsMgr

    common functions manager implementation (set of function independs on target)

Functions

- void ∗ equipListenPolling (void ∗)

    Thread to polling listen udp for equipment data.

7.17.1 Detailed Description

Class commonFuncsMgr interface header.

Author

Vladimir A. Nosenko (`nosenko@ieee.org`)

Date

December, 2010

Copyright (c) 2010 Vladimir A.Nosenko.

The license and distribution terms for this file may be found in the file LICENSE in this distribution

Definition in file commonFuncsMgr.h.

7.17.2 Function Documentation

7.17.2.1 void∗ equipListenPolling (void ∗ *user*)

Thread to polling listen udp for equipment data.

Todo

add to commonFuncsMgr class as static method

Calls equipListenProcessing for decoding data received from equipment

Return values

user	

Todo

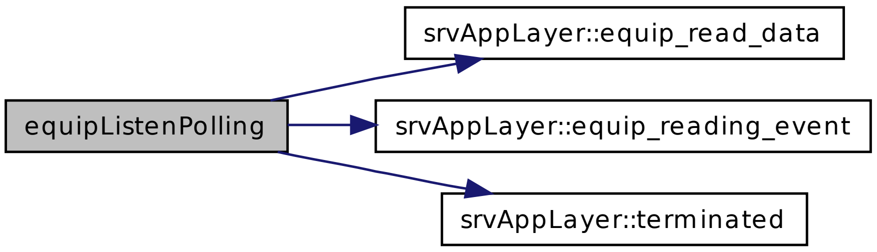
Listening equipment answer - status vector:

Definition at line 35 of file commonFuncsMgr.cpp.

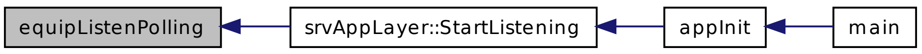
References app, srvAppLayer::equip_read_data(), srvAppLayer::equip_reading_event(), and srvAppLayer::terminated().

Referenced by srvAppLayer::StartListening().

Here is the call graph for this function:


equipListenPolling
srvAppLayer::equip_read_data
srvAppLayer::equip_reading_event
srvAppLayer::terminated

Here is the caller graph for this function:



7.18 src/functions/specFuncsMgr.h File Reference

Class specFuncsMgr interface header.

Classes

- class specFuncsMgr

  special functions set manager.

7.18.1 Detailed Description

Class specFuncsMgr interface header.

Author

Vladimir A. Nosenko (`nosenko@ieee.org`)

Date

December, 2010

Definition in file specFuncsMgr.h.

7.19 src/main.cpp File Reference

Programm entry point.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <arpa/inet.h>
#include <pthread.h>
#include <signal.h>
#include <sys/reboot.h>
#include <deque>
#include "../../rcsLib/ortsTypes/ortsTypes.h"
#include "buffer/ssBuffer.h"
#include "../../udp_port/udp_port.h"
#include "arg_parser/carg_parser.h"
#include "console_out.h"
#include "../../rcsLib/rcsCmd/rcsCmd.h"
#include "ICAppLayer/FunctionNode/param_desc.h"
#include "ICAppLayer/FunctionNode/FunctionNode.h"
#include "ICAppLayer/ICAppLayer.h"
#include "Functions/CommonFuncs.h"
#include "Functions/SpecFuncs.h"
#include "Functions/functions.h"
#include "SIG_handler.h"
#include "global.h"
```

Functions

- errType process_cmdLine (int argc, char ∗argv[ ])

  Parsing commandline arguments.

- errType fileRead (char ∗fname, BYTE ∗∗buffer, size_t ∗sz)

  useless function in this programm.

- void dbg_hex_print (BYTE ∗buffer, size_t len)

  prints hex bytes from buffer with size len.

- errType appInit (void)

  Initialize srvAppLayer subsystem.

- errType appDeinit (void)

  Deinitialize srvAppLayer subsystem.

- int main (int argc, char ∗argv[ ])

  Programm entrypoint.

7.19.1 Detailed Description

Programm entry point.

Author

Vladimir A. Nosenko (nosenko@ieee.org)

Date

December, 2010

Copyright (c) 2010 Vladimir A.Nosenko.

The license and distribution terms for this file may be found in the file LICENSE in this distribution

Definition in file main.cpp.

7.19.2 Function Documentation

7.19.2.1 errType process_cmdLine (int *argc,* char ∗ *argv[ ]*)

Parsing commandline arguments.

Parameters

in	*argc*	- count of arguments strings
in	*argv[ ]*	- array of arguments strings

Return values

err_result_ok	- if execution was successful
err_not_found	- if no arguments found
err_result_error	- if parsing was unsuccessful

Todo

reorganize process to external library

1. Define arguments type: with (*ap_yes*) or without (*ap_no*) parameters
2. Initialize arguments parser ap_init
3. Check for parsing errors ap_error
4. Execute all arguments after it parsing

    - get code of argument ap_code
    - switch with argument code value
    - execute

4. Execute only arguments with parameters after it parsing

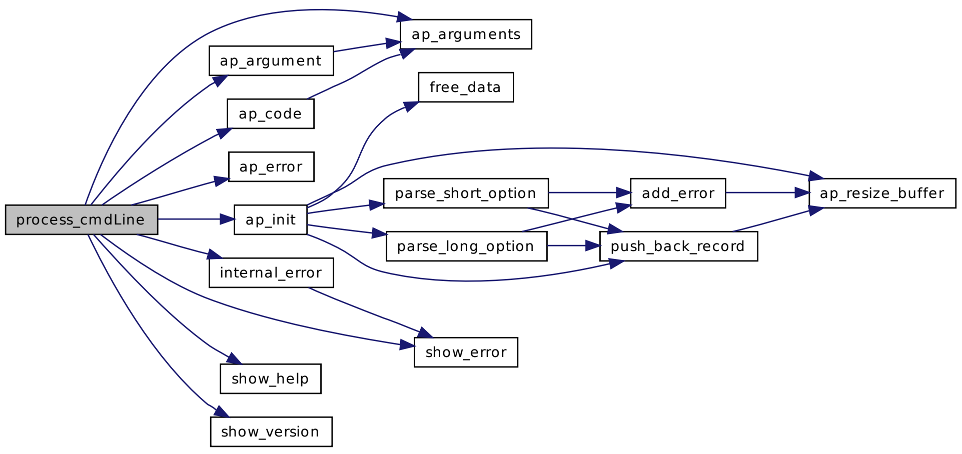
    - get code of argument ap_code
    - get argument parameter ap_argument
    - switch with argument code value
    - execute

Definition at line 51 of file main.cpp.

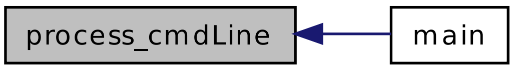
References ap_argument(), ap_arguments(), ap_code(), ap_error(), ap_init(), ap_no, ap_yes, eq_ip_addr, eq_udp_listen_port, eq_udp_sending_port, equipAddr, internal_-error(), show_error(), show_help(), show_version(), verbose_level, and wUdp.

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



7.19.2.2 errType fileRead (char ∗ *fname,* BYTE ∗∗ *buffer,* size_t ∗ *sz*)

useless function in this programm.

stays from good old times

Todo

reorganize function to reading xml-files for future purposes

Definition at line 147 of file main.cpp.

7.19.2.3 void dbg_hex_print (BYTE ∗ *buffer,* size_t *len*)

prints hex bytes from *buffer* with size *len*.

Todo

use this function in new debug print system

Definition at line 188 of file main.cpp.

7.19.2.4 errType appInit (void)

Initialize srvAppLayer subsystem.

result copied from srvAppLayer::StartListening

Return values

err_result_ok	- execution was successful
err_sock_error	- problems with communications subsystem

Starting main programm threads srvAppLayer::StartListening():

1. Prepare queues for sending and listening to/from clients
2. Send & Listen threads for clients communication
3. Listen thread for equipment communication

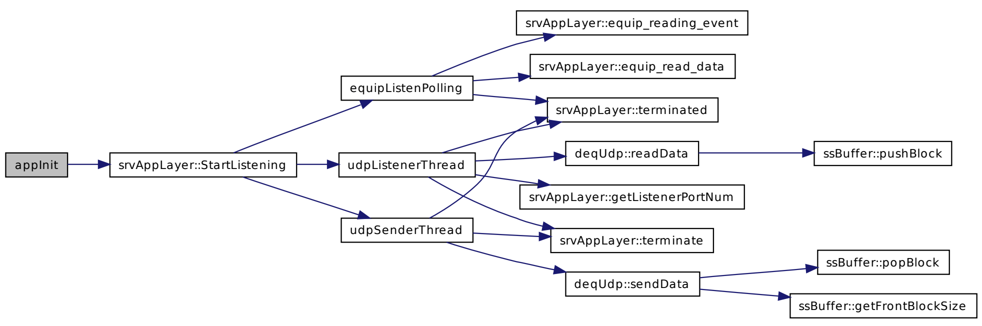
If threads started successfully - starts service specific function initialize srvInit()

Definition at line 203 of file main.cpp.

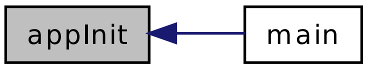
References app, srvAppLayer::StartListening(), and verbose_level.

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



7.19.2.5 errType appDeinit (void)

Deinitialize srvAppLayer subsystem.

Return values

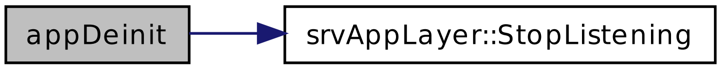
always	return err_result_ok, why not?

Definition at line 230 of file main.cpp.

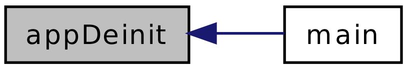
References app, and srvAppLayer::StopListening().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



7.19.2.6 int main (int *argc*, char ∗ *argv[]*)

Programm entrypoint.

Return values

EXIT_FAILURE	
err_not_init	

1. Process command line arguments **argc** and **argv**[] in process_cmdLine
   - if arguments parsing is unsuccessfull exiting from programm
2. Check arguments:
   - check for missing one of exact argument
   - check for equipment communication settings:
     - sending port need to be not equal to listen port values
     - sending or listen port neet to be not equal to client listen port
   - check for sending port number or listening port number was far from client port number on one port number that reserved for client sending port.
3. Install system signals handlers installSIGhandlers()
4. Initialize application appInit()
4. Start functions generate from declarations
   - for common functions commonFuncsMgr::startCommonFuncs()
   - for special functions specFuncsMgr::startSpecFuncs()

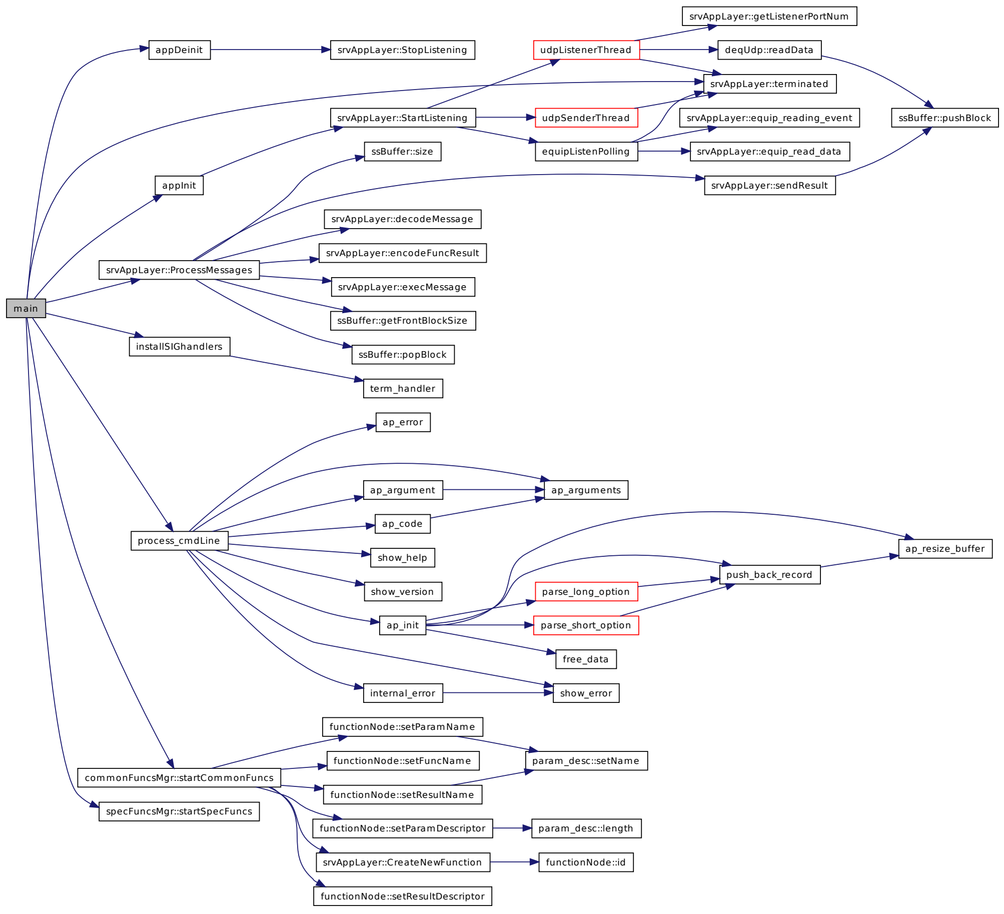
5. Main programm loop srvAppLayer::ProcessMessages() while not terminated by signal srvAppLayer::terminated()

6. Deinitialize application appDeinit()

Definition at line 244 of file main.cpp.

References app, appDeinit(), appInit(), AppTerminated, eq_udp_listen_port, eq_udp_sending_port, equipAddr, installSIGhandlers(), process_cmdLine(), srvAppLayer::ProcessMessages(), commonFuncsMgr::startCommonFuncs(), specFuncsMgr::startSpecFuncs(), srvAppLayer::terminated(), and wUdp.

Here is the call graph for this function:


srvAppLayer::getListenerPortNum
appDeinit
srvAppLayer::StopListening
udpListenerThread
deqUdp::readData
srvAppLayer::terminated
srvAppLayer::StartListening
udpSenderThread
srvAppLayer::equip_reading_event
ssBuffer::pushBlock
ssBuffer::size
equipListenPolling
srvAppLayer::equip_read_data
appInit
srvAppLayer::sendResult
srvAppLayer::decodeMessage
srvAppLayer::encodeFuncResult
srvAppLayer::ProcessMessages
srvAppLayer::execMessage
main
ssBuffer::getFrontBlockSize
installSIGhandlers
ssBuffer::popBlock
term_handler
ap_error
ap_argument
ap_arguments
ap_code
process_cmdLine
ap_resize_buffer
show_help
push_back_record
show_version
parse_long_option
ap_init
parse_short_option
free_data
internal_error
show_error
functionNode::setParamName
functionNode::setFuncName
param_desc::setName
commonFuncsMgr::startCommonFuncs
functionNode::setResultName
specFuncsMgr::startSpecFuncs
functionNode::setParamDescriptor
param_desc::length
srvAppLayer::CreateNewFunction
functionNode::id
functionNode::setResultDescriptor

7.20 src/SIG_handler.cpp File Reference

System signals handlers manager.

```
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <arpa/inet.h>
#include "../../rcsLib/ortsTypes/ortsTypes.h"
#include "global.h"
```

Functions

- void term_handler (int i)

  *signal **TERMINATE** handling function*

- void installSIGhandlers (funcVoid func)

  signals handlers installer

Variables

- funcVoid SIGTERM_handler

  *pointer to handling function for signal **TERMINATE***

7.20.1 Detailed Description

System signals handlers manager.

Author

Vladimir A. Nosenko (nosenko@ieee.org)

Date

December, 2010

Definition in file SIG_handler.cpp.

7.20.2 Function Documentation

7.20.2.1 void term_handler (int *i*)

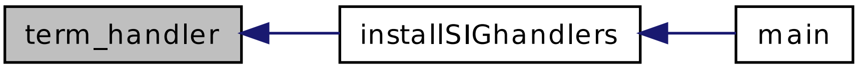
signal **TERMINATE** handling function

Definition at line 29 of file SIG_handler.cpp.

References SIGTERM_handler, and verbose_level.

Referenced by installSIGhandlers().

Here is the caller graph for this function:



7.20.2.2 void installSIGhandlers (funcVoid *func*)

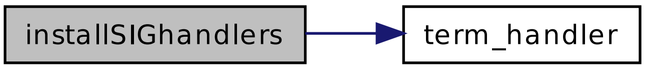
signals handlers installer

Definition at line 39 of file SIG_handler.cpp.

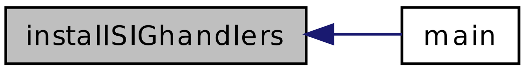
References SIGTERM_handler, and term_handler().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



7.20.3 Variable Documentation

7.20.3.1 funcVoid SIGTERM_handler

pointer to handling function for signal **TERMINATE**

Definition at line 24 of file SIG_handler.cpp.

Referenced by installSIGhandlers(), and term_handler().

7.21 src/SIG_handler.h File Reference

System signals handlers manager interface header.

Functions

- void installSIGhandlers (funcVoid func)

  signals handlers installer

7.21.1 Detailed Description

System signals handlers manager interface header.

Author

Vladimir A. Nosenko (`nosenko@ieee.org`)

Date

December, 2010

Copyright (c) 2010 Vladimir A.Nosenko.

The license and distribution terms for this file may be found in the file LICENSE in this distribution

Definition in file SIG_handler.h.

7.21.2 Function Documentation

7.21.2.1 void installSIGhandlers (funcVoid *func*)

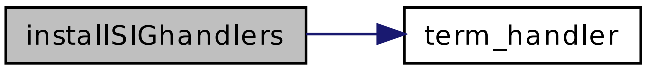
signals handlers installer

Definition at line 39 of file SIG_handler.cpp.

References SIGTERM_handler, and term_handler().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

installSIGhandlers
main

7.22 src/srvAppLayer/functionNode/functionNode.cpp File Reference

Class functionNode implementation.

`#include <stdio.h>`

`#include <string.h>`

`#include "../../../../rcsLib/ortsTypes/ortsTypes.h"`

```
#include "param_desc.h"
#include "../../../../rcsLib/rcsCmd/rcsCmd.h"
#include "functionNode.h"
```

7.22.1 Detailed Description

Class functionNode implementation.

Author

Vladimir A. Nosenko (nosenko@ieee.org)

Date

December, 2010

Copyright (c) 2010 Vladimir A.Nosenko.

The license and distribution terms for this file may be found in the file LICENSE in this distribution

Definition in file functionNode.cpp.

7.23 src/srvAppLayer/functionNode/functionNode.h File Reference

Class functionNode interface header.

Classes

- class functionNode

  function node interface header

7.23.1 Detailed Description

Class functionNode interface header.

Author

Vladimir A. Nosenko (nosenko@ieee.org)

Date

December, 2010

Copyright (c) 2010 Vladimir A.Nosenko.

The license and distribution terms for this file may be found in the file LICENSE in this distribution

Definition in file functionNode.h.

7.24 src/srvAppLayer/functionNode/param_desc.cpp File Reference

Class param_desc implementation.

```
#include <stdio.h>
#include <string.h>
#include "../../../../rcsLib/ortsTypes/ortsTypes.h"
#include "param_desc.h"
```

7.24.1 Detailed Description

Class param_desc implementation.

Author

Vladimir A. Nosenko (`nosenko@ieee.org`)

Date

December, 2010

Copyright (c) 2010 Vladimir A.Nosenko.

The license and distribution terms for this file may be found in the file LICENSE in this distribution

Definition in file param_desc.cpp.

7.25 src/srvAppLayer/functionNode/param_desc.h File Reference

Class param_desc interface header.

Classes

- class param_desc

    parameter description

7.25.1 Detailed Description

Class param_desc interface header.

Author

Vladimir A. Nosenko (nosenko@ieee.org)

Date

December, 2010

Copyright (c) 2010 Vladimir A.Nosenko.

The license and distribution terms for this file may be found in the file LICENSE in this distribution

Definition in file param_desc.h.

7.26 src/srvAppLayer/srvAppLayer.cpp File Reference

Class srvAppLayer implementation.

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <pthread.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <iostream>
#include <queue>
#include "../../rcsLib/ortsTypes/ortsTypes.h"
#include "../../rcsLib/rcsCmd/rcsCmd.h"
#include "../global.h"
#include "../buffer/ssBuffer.h"
#include "../../udp_port/udp_port.h"
#include "../deqUdp/deqUdp.h"
#include "functionNode/param_desc.h"
#include "functionNode/functionNode.h"
#include "srvAppLayer.h"
#include "../functions/commonFuncsMgr.h"
```

Functions

- void ∗ udpSenderThread (void ∗user)

  Thread to sending data to clients from functions answer queue.

- void ∗ udpListenerThread (void ∗user)

  Thread to listening requests from clients and form queue of clients requests.

Variables

- bool rcvComplete_flag = false

  todo msc diagramm

- bool sndAllow_flag = false

  todo msc diagramm

- srvAppLayer ∗ app

  One global instance per application.

7.26.1 Detailed Description

Class srvAppLayer implementation.

Author

Vladimir A. Nosenko (nosenko@ieee.org)

Date

December, 2010

Definition in file srvAppLayer.cpp.

7.26.2 Function Documentation

7.26.2.1 void∗ udpSenderThread (void ∗ *user*)

Thread to sending data to clients from functions answer queue.

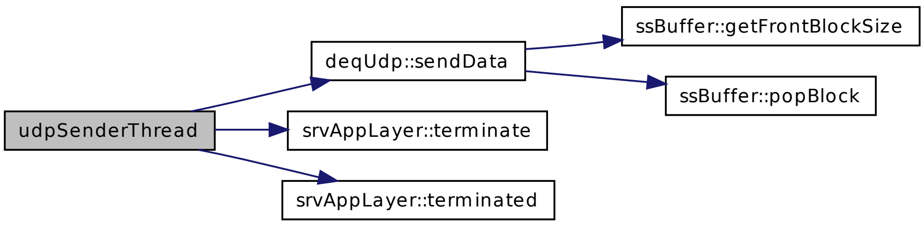
Thread also includes work with port opening, initializing and closing.

Definition at line 50 of file srvAppLayer.cpp.

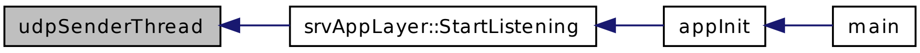
References srvAppLayer::functionsAnswersQueue, deqUdp::sendData(), sndAllow_flag, srvAppLayer::terminate(), srvAppLayer::terminated(), and wUdp.

Referenced by srvAppLayer::StartListening().

Here is the call graph for this function:



Here is the caller graph for this function:



7.26.2.2 void∗ udpListenerThread (void ∗ *user*)

Thread to listening requests from clients and form queue of clients requests.

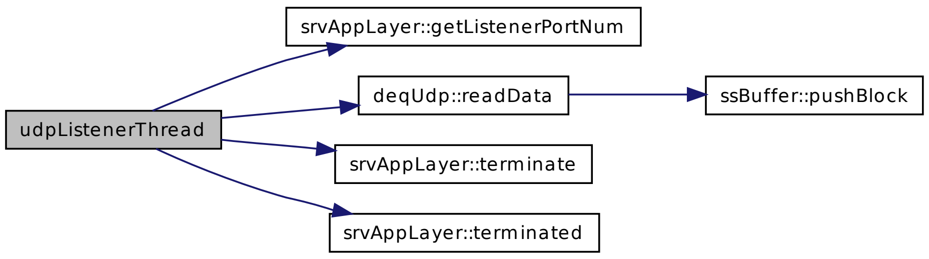
Thread also includes work with port opening, initializing and closing.

Definition at line 102 of file srvAppLayer.cpp.

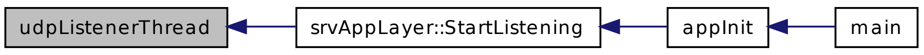
References srvAppLayer::clientsRequestsQueue, srvAppLayer::getListenerPortNum(), rcvComplete_flag, deqUdp::readData(), srvAppLayer::terminate(), srvAppLayer::terminated(), and wUdp.

Referenced by srvAppLayer::StartListening().

Here is the call graph for this function:



Here is the caller graph for this function:



7.26.3 Variable Documentation

7.26.3.1 bool rcvComplete_flag = false

todo msc diagramm

Flag purpose: synchronize state between receiving clients requests thread and reading for received data

Definition at line 39 of file srvAppLayer.cpp.

Referenced by srvAppLayer::ProcessMessages(), and udpListenerThread().

7.26.3.2 bool sndAllow_flag = false

todo msc diagramm

Flag purpose: synchronize state between sending clients answers thread and preparing sending data

Definition at line 41 of file srvAppLayer.cpp.

Referenced by srvAppLayer::ProcessMessages(), and udpSenderThread().

7.26.3.3 srvAppLayer∗ app

One global instance per application.

Definition at line 43 of file srvAppLayer.cpp.

Referenced by appDeinit(), appInit(), equipListenPolling(), and main().

7.27 src/srvAppLayer/srvAppLayer.h File Reference

Class srvAppLayer interface header.

`#include <pthread.h>`

Classes

- struct serviceState

  stateVector_type structural field.

- struct stateVector_type

  Main vector of service base states.

- class srvAppLayer

  Application core layer implementaion.

Typedefs

- typedef struct serviceState serviceState
- typedef struct stateVector_type stateVector_type

Variables

- srvAppLayer ∗ app

  One global instance per application.

7.27.1 Detailed Description

Class srvAppLayer interface header.

Author

Vladimir A. Nosenko (`nosenko@ieee.org`)

Date

December, 2010

Definition in file srvAppLayer.h.

7.27.2 Typedef Documentation

7.27.2.1 typedef struct serviceState serviceState

7.27.2.2 typedef struct stateVector_type stateVector_type

7.27.3 Variable Documentation

7.27.3.1 srvAppLayer* app

One global instance per application.

Definition at line 43 of file srvAppLayer.cpp.

Referenced by appDeinit(), appInit(), equipListenPolling(), and main().