

Remote Command Message (RCM)

Release 27.9

Generated by Doxygen 1.7.3

Mon Jan 10 2011 23:17:18

Contents

1	Module Index	1
1.1	Modules	1
2	Data Structure Index	3
2.1	Data Structures	3
3	File Index	5
3.1	File List	5
4	Module Documentation	7
4.1	RcmClient	7
4.1.1	Detailed Description	11
4.1.2	Define Documentation	14
4.1.2.1	RcmClient_DEFAULTPOOLID	14
4.1.2.2	RcmClient_DISCRETEJOBID	15
4.1.2.3	RcmClient_E_EXECASYNCNOTENABLED	15
4.1.2.4	RcmClient_E_EXECFAILED	15
4.1.2.5	RcmClient_E_INVALIDFXNIDX	15
4.1.2.6	RcmClient_E_INVALIDHEAPID	15
4.1.2.7	RcmClient_E_IPCERROR	15
4.1.2.8	RcmClient_E_JOBIDNOTFOUND	15
4.1.2.9	RcmClient_E_LOSTMSG	16
4.1.2.10	RcmClient_E_MSGALLOCFAILED	16
4.1.2.11	RcmClient_E_MSGFXNERROR	16
4.1.2.12	RcmClient_E_MSGQCREATEFAILED	16
4.1.2.13	RcmClient_E_MSGQOPENFAILED	16
4.1.2.14	RcmClient_E_SERVERERROR	16
4.1.2.15	RcmClient_E_SERVERNOTFOUND	17
4.1.2.16	RcmClient_E_SYMBOLNOTFOUND	17
4.1.3	Typedef Documentation	17
4.1.3.1	RcmClient_CallbackFxn	17
4.1.4	Function Documentation	17
4.1.4.1	RcmClient_acquireJobId	17
4.1.4.2	RcmClient_addSymbol	17
4.1.4.3	RcmClient_alloc	18

4.1.4.4	RcmClient_checkForError	19
4.1.4.5	RcmClient_construct	20
4.1.4.6	RcmClient_create	20
4.1.4.7	RcmClient_delete	21
4.1.4.8	RcmClient_destruct	21
4.1.4.9	RcmClient_exec	21
4.1.4.10	RcmClient_execAsync	22
4.1.4.11	RcmClient_execCmd	23
4.1.4.12	RcmClient_execDpc	23
4.1.4.13	RcmClient_execNoWait	24
4.1.4.14	RcmClient_exit	24
4.1.4.15	RcmClient_free	24
4.1.4.16	RcmClient_getSymbolIndex	24
4.1.4.17	RcmClient_init	25
4.1.4.18	RcmClient_releaseJobId	25
4.1.4.19	RcmClient_removeSymbol	25
4.1.4.20	RcmClient_waitUntilDone	25
4.2	RcmServer	26
4.2.1	Detailed Description	28
4.2.2	Define Documentation	28
4.2.2.1	RcmServer_E_SYMBOLNOTFOUND	28
4.2.2.2	RcmServer_E_SYMBOLSTATIC	29
4.2.2.3	RcmServer_E_SYMBOLTABLEFULL	29
4.2.3	Typedef Documentation	29
4.2.3.1	RcmServer_MsgFxn	29
4.2.4	Function Documentation	29
4.2.4.1	RcmServer_addSymbol	29
4.2.4.2	RcmServer_construct	30
4.2.4.3	RcmServer_create	30
4.2.4.4	RcmServer_delete	31
4.2.4.5	RcmServer_destruct	31
4.2.4.6	RcmServer_exit	31
4.2.4.7	RcmServer_init	32
4.2.4.8	RcmServer_removeSymbol	32
4.2.4.9	RcmServer_start	32
4.3	RcmTypes	33
4.3.1	Detailed Description	34
5	Data Structure Documentation	37
5.1	RcmClient_Message Struct Reference	37
5.1.1	Detailed Description	38
5.1.2	Field Documentation	38
5.1.2.1	data	38
5.1.2.2	dataSize	38
5.1.2.3	jobId	38
5.1.2.4	poolId	38

5.2	RcmClient_Packet Struct Reference	38
5.2.1	Detailed Description	39
5.2.2	Field Documentation	39
5.2.2.1	desc	39
5.2.2.2	message	39
5.2.2.3	msgId	39
5.2.2.4	msgqHeader	40
5.3	RcmClient_Params Struct Reference	40
5.3.1	Detailed Description	40
5.3.2	Field Documentation	40
5.3.2.1	callbackNotification	40
5.3.2.2	heapId	41
5.4	RcmClient_Struct Struct Reference	41
5.4.1	Detailed Description	41
5.5	RcmServer_FxnDesc Struct Reference	42
5.5.1	Detailed Description	42
5.5.2	Field Documentation	42
5.5.2.1	addr	42
5.5.2.2	name	42
5.6	RcmServer_FxnDescAry Struct Reference	42
5.6.1	Detailed Description	43
5.7	RcmServer_Params Struct Reference	43
5.7.1	Detailed Description	44
5.7.2	Field Documentation	44
5.7.2.1	fxns	44
5.7.2.2	osPriority	44
5.7.2.3	priority	45
5.7.2.4	workerPools	45
5.8	RcmServer_Struct Struct Reference	45
5.8.1	Detailed Description	46
5.9	RcmServer_ThreadPoolDesc Struct Reference	46
5.9.1	Detailed Description	47
5.9.2	Field Documentation	47
5.9.2.1	osPriority	47
5.9.2.2	priority	47
5.10	RcmServer_ThreadPoolDescAry Struct Reference	47
5.10.1	Detailed Description	48
6	File Documentation	49
6.1	ti/sdo/rcm/RcmClient.h File Reference	49
6.1.1	Detailed Description	53
6.2	ti/sdo/rcm/RcmServer.h File Reference	53
6.2.1	Detailed Description	56
6.3	ti/sdo/rcm/RcmTypes.h File Reference	56
6.3.1	Detailed Description	58

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

RcmClient	7
RcmServer	26
RcmTypes	33

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

- [RcmClient_Message](#) (Remote Command Message structure) 37
- [RcmClient_Packet](#) (Remote Command Message (RCM) packet structure) . . 38
- [RcmClient_Params](#) (Instance create parameters) 40
- [RcmClient_Struct](#) (Opaque client structure large enough to hold an instance object) 41
- [RcmServer_FxnDesc](#) (Function descriptor) 42
- [RcmServer_FxnDescAry](#) (Function descriptor array) 42
- [RcmServer_Params](#) (RcmServer Instance create parameters) 43
- [RcmServer_Struct](#) (Opaque client structure large enough to hold an instance object) 45
- [RcmServer_ThreadPoolDesc](#) (Worker pool descriptor) 46
- [RcmServer_ThreadPoolDescAry](#) (Worker pool descriptor array) 47

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

- ti/sdo/rcm/[RcmClient.h](#) (Remote Command Message Client Module. An RcmClient is used for sending messages to an RcmServer for processing) 49
- ti/sdo/rcm/[RcmServer.h](#) (Remote Command Message Server Module. An RcmServer processes inbound messages received from an RcmClient) 53
- ti/sdo/rcm/[RcmTypes.h](#) (Remote Command Message (RCM) common definitions between Client and Server) 56

Chapter 4

Module Documentation

4.1 RcmClient

Remote Command Message Client Module. An RcmClient is used for sending messages to an RcmServer for processing.

Data Structures

- struct [RcmClient_Message](#)
Remote Command Message structure.
- struct [RcmClient_Params](#)
Instance create parameters.
- struct [RcmClient_Struct](#)
Opaque client structure large enough to hold an instance object.

Defines

- #define [RcmClient_S_SUCCESS](#) (0)
Success return code.
- #define [RcmClient_E_FAIL](#) (-1)
General failure return code.
- #define [RcmClient_E_EXECASYNCNOTENABLED](#) (-2)

The client has not been configured for asynchronous notification.

- #define [RcmClient_E_EXECFAILED](#) (-3)
The client was unable to send the command message to the server.
- #define [RcmClient_E_INVALIDHEAPID](#) (-4)
A heap id must be provided in the create params.
- #define [RcmClient_E_INVALIDFXNIDX](#) (-5)
Invalid function index.
- #define [RcmClient_E_MSGFXNERROR](#) (-6)
Message function error.
- #define [RcmClient_E_IPCERROR](#) (-7)
An unknown error has been detected from the IPC layer.
- #define [RcmClient_E_LISTCREATEFAILED](#) (-8)
Failed to create the list object.
- #define [RcmClient_E_LOSTMSG](#) (-9)
The expected reply message from the server was lost.
- #define [RcmClient_E_MSGALLOCFAILED](#) (-10)
Insufficient memory to allocate a message.
- #define [RcmClient_E_MSGQCREATEFAILED](#) (-11)
The client message queue could not be created.
- #define [RcmClient_E_MSGQOPENFAILED](#) (-12)
The server message queue could not be opened.
- #define [RcmClient_E_SERVERERROR](#) (-13)
The server returned an unknown error code.
- #define [RcmClient_E_SERVERNOTFOUND](#) (-14)
The server specified in the create params was not found.
- #define [RcmClient_E_SYMBOLNOTFOUND](#) (-15)
The given symbol was not found in the server symbol table.
- #define [RcmClient_E_NOMEMORY](#) (-16)

There is insufficient memory left in the heap.

- #define `RcmClient_E_JOBIDNOTFOUND` (-17)
The given job id was not found on the server.
- #define `RcmClient_INVALIDFXNIDX` ((UInt32)(0xFFFFFFFF))
Invalid function index.
- #define `RcmClient_INVALIDHEAPID` ((UInt16)(0xFFFF))
Invalid heap id.
- #define `RcmClient_INVALIDMSGID` (0)
Invalid message id.
- #define `RcmClient_DEFAULTPOOLID` ((UInt16)(0x8000))
Default worker pool id.
- #define `RcmClient_DISCRETEJOBID` (0)
Invalid job stream id.

Typedefs

- typedef struct RcmClient_Object_tag * `RcmClient_Handle`
RcmClient instance object handle.
- typedef Void(* `RcmClient_CallbackFxn`)(RcmClient_Message *, Ptr)
Callback function type.

Functions

- Int `RcmClient_acquireJobId` (RcmClient_Handle handle, UInt16 *jobId)
Get a job id from the server.
- Int `RcmClient_addSymbol` (RcmClient_Handle handle, String name, Fxn addr, UInt32 *index)
Add a symbol and its address to the server table.
- Int `RcmClient_alloc` (RcmClient_Handle handle, UInt32 dataSize, RcmClient_Message **message)
Allocate a message from the heap configured for this instance.

- Int `RcmClient_checkForError` (`RcmClient_Handle` handle, `RcmClient_Message` `**returnMsg`)
Check if an error message has been returned from the server.
- Int `RcmClient_construct` (`RcmClient_Struct` `*structPtr`, String server, const `RcmClient_Params` `*params`)
Initialize a new instance object inside the provided structure.
- Int `RcmClient_create` (String server, const `RcmClient_Params` `*params`, `RcmClient_Handle` `*handle`)
Create an RcmClient instance.
- Int `RcmClient_delete` (`RcmClient_Handle` `*handlePtr`)
Delete an RcmClient instance.
- Int `RcmClient_destruct` (`RcmClient_Struct` `*structPtr`)
Finalize the instance object inside the provided structure.
- Int `RcmClient_exec` (`RcmClient_Handle` handle, `RcmClient_Message` `*cmdMsg`, `RcmClient_Message` `**returnMsg`)
Execute a command message on the server.
- Int `RcmClient_execAsync` (`RcmClient_Handle` handle, `RcmClient_Message` `*cmdMsg`, `RcmClient_CallbackFxn` callback, Ptr appData)
Execute a command message and use a callback for notification.
- Int `RcmClient_execCmd` (`RcmClient_Handle` handle, `RcmClient_Message` `*cmdMsg`)
Execute a one-way command message on the server.
- Int `RcmClient_execDpc` (`RcmClient_Handle` handle, `RcmClient_Message` `*cmdMsg`, `RcmClient_Message` `**returnMsg`)
Execute a deferred procedure call on the server.
- Int `RcmClient_execNoWait` (`RcmClient_Handle` handle, `RcmClient_Message` `*cmdMsg`, UInt16 `*msgId`)
Submit a command message to the server and return immediately.
- Void `RcmClient_exit` (Void)
Finalize the RcmClient module.
- Int `RcmClient_free` (`RcmClient_Handle` handle, `RcmClient_Message` `*msg`)

Free the given message.

- Int `RcmClient_getSymbolIndex` (`RcmClient_Handle` handle, String name, UInt32 *index)
Return the function index from the server.
- Void `RcmClient_init` (Void)
Initialize the RcmClient module.
- Void `RcmClient_Params_init` (`RcmClient_Params` *params)
Initialize the instance create params structure.
- Int `RcmClient_releaseJobId` (`RcmClient_Handle` handle, UInt16 jobId)
Return a job id to the server and release all resources.
- Int `RcmClient_removeSymbol` (`RcmClient_Handle` handle, String name)
Remove a symbol and from the server function table.
- Int `RcmClient_waitUntilDone` (`RcmClient_Handle` handle, UInt16 msgId, `RcmClient_Message` **returnMsg)
Block until the specified message has been executed.

4.1.1 Detailed Description

Remote Command Message Client Module. An RcmClient is used for sending messages to an RcmServer for processing. The RcmClient module is used to send messages to an RcmServer for processing. An RcmClient instance must be created which will communicate with a single RcmServer instance. An RcmServer instance can receive messages from many RcmClient instances but an RcmClient instance can send messages to only one RcmServer instance.

Error Handling

Errors may be raised at various points in the execution flow of a single message. These are the types of errors raised by RCM:

- RCM Errors
- User Errors
- Message Function Errors
- Library Function Errors

The following diagram illustrates the call flow and highlights the points at which errors can be raised.

RCM errors are raised by the RCM implementation. For example, when calling `RcmClient_exec()`, it may raise an error if it is unable to send the message. This will be reported immediately to the caller, the server is never involved. This is highlighted in the diagram at point [A].

RCM errors may also be raised by the `RcmServer`. Once `RcmClient_exec()` sends a message, it waits for the return message. When the server receives the message, it has to decode its type. If this fails, the server records this error in the message and sends it back to the client. This is highlighted at point [B] in the diagram. When `RcmClient_exec()` receives the return message, it inspects the status field and recognizes the error. It then raises an appropriate error to the caller.

User errors are raised by RCM as a result of an invalid runtime condition. They are typically detected and raised by the server and then sent back to the client for processing. For example, when the server receives a message, it will lookup the function index in its table and invoke the corresponding message function. If the index is invalid, this is detected by the server and reported as an error. The message function is never invoked. The message is returned to the client and `RcmClient_exec()` will raise the appropriate error. This error is raised at point [B] in the diagram.

Message function errors are raised by the message function itself. This is highlighted at point [C] in the diagram. In the body of the message function, an error is encountered either when calling a framework function or when unmarshalling the arguments from the message payload. In both cases, the message function will abort and return an error value as its return value. The error value must be less than zero (< 0). When control returns to the server, it inspects the message function's return value. If the message was sent by `RcmClient_exec()`, then the server simply stores the return value in the message and returns it to the client. However, if the message was sent by `RcmClient_execCmd()`, the behavior is slightly different. If the return value is < 0 (error condition), then the server behaves the same as for the `RcmClient_exec()` case. However, if there is no error, the server simply frees the message. The message is never returned to the client.

Library functions errors are raised by the library functions invoked by the message function, highlighted at point [D] in the diagram. It is the responsibility of the message function to detect a library function error and to marshal the appropriate error code into the message payload. The message function must also return an error value (< 0) so that server can do proper error handling as described in the preceding paragraph. It is the stub function's responsibility to unmarshal the error value from the message payload and take the appropriate action.

Understanding the types of errors raised by RCM and where they are raised is important when writing the upper layer software. The following code fragment illustrates how to check for the errors discussed above. See [Error Handling](#error_handling) above.

```
RcmClient_Message *msg;

// send message
status = RcmClient_exec(h, msg, &msg);

// check for error
switch (status) {

    case RcmClient_S_SUCCESS:
        // no error, all is well
        break;

    case RcmClient_E_EXECFAILED:
        // RCM error, unable to send message
        :
        break;

    case RcmClient_E_INVALIDFXNIDX:
        // user error, bad function index
        :
        break;

    case RcmClient_E_MSGFXNERROR:
        // decode message function error
        msgFxnErr = msg->result;
        :
        // decode library function error
        libFxnErr = msg->data[0];
        :
        break;

}

// free message if one was returned
if (msg != NULL) {
    RcmClient_free(h, msg);
    msg = NULL;
}
```

The upper layer software begins by calling `RcmClient_exec()` to send the message to the server. If RCM encounters a transport error and is unable to send the message, an `RcmClient_E_EXECFAILED` error is returned.

If the server encounters an error while decoding the message, say an invalid function index, the server returns the message and an `RcmClient_E_INVALIDFXNIDX` error is returned.

If the message function or the library function encounter an error, the message is returned and an `RcmClient_E_MSGFXNERROR` error is returned. The upper layer software must decode the results field to see what error the message function returned. If the library function returned an error, its error code must be unmarshalled from the payload. This is illustrated by decoding the first word in the payload (`msg->data[0]`). The actual marshal format is implementation specific.

The message function must return ≥ 0 for success, < 0 for error. Here is a very simple message function which simply turns on an LED.

```
Int32 Led_On(UInt32 size, UInt32 *data)
{
    Ptr led = <addr>;

    *led = 1; // turn on LED
    return(0); // success
}
```

In this example, the message payload is not used. The message function does all the work, it just sets a bit in the LED control register, and returns success.

This next example illustrates a very simple RPC style skel function. It unmarshalls two arguments and invokes the library function. If the library function succeeds, the message function returns 0, otherwise it returns -1 to indicate failure. In both cases, the library function's return value is marshalled into the payload. The calling stub function can unmarshal the return value to get the library function's error code.

```
Int32 MediaSkel_process(UInt32 size, UInt32 *data)
{
    Int a, b, x;
    Int32 status = 0; // success

    // unmarshal args
    a = (Int)data[1];
    b = (Int)data[2];

    // invoke library function, returns >=0 on success
    x = Media_process(a, b);

    // marshal return value
    data[0] = (Int32)x;

    // return status
    return(x >= 0 ? 0 : -1);
}
```

See also

[RcmServer](#)
RCM Overview

4.1.2 Define Documentation

4.1.2.1 #define RcmClient_DEFAULTPOOLID ((UInt16)(0x8000))

Default worker pool id.

The default worker pool is used to process all anonymous messages. When a new message is allocated, the pool id property is initialized to this value.

4.1.2.2 #define RcmClient.DISCRETEJOBID (0)

Invalid job stream id.

All discrete messages must have their jobId property set to this value. When a new message is allocated, the jobId property is initialized to this value.

4.1.2.3 #define RcmClient.E.EXECASYNCNOTENABLED (-2)

The client has not been configured for asynchronous notification.

In order to use the [RcmClient_execAsync\(\)](#) function, the RcmClient must be configured with callbackNotification set to true in the instance create parameters.

4.1.2.4 #define RcmClient.E.EXECFAILED (-3)

The client was unable to send the command message to the server.

An IPC transport error occurred. The message was never sent to the server.

4.1.2.5 #define RcmClient.E.INVALIDFXNIDX (-5)

Invalid function index.

An [RcmClient_Message](#) was sent to the server which contained a function index value (in the fxnIdx field) that was not found in the server's function table.

4.1.2.6 #define RcmClient.E.INVALIDHEAPID (-4)

A heap id must be provided in the create params.

When an RcmClient instance is created, a heap id must be given in the create params. This heap id must be registered with MessageQ before calling [RcmClient_create\(\)](#).

4.1.2.7 #define RcmClient.E.IPCERROR (-7)

An unknown error has been detected from the IPC layer.

Check the error log for additional information.

4.1.2.8 #define RcmClient.E.JOBIDNOTFOUND (-17)

The given job id was not found on the server.

When releasing a job id with a call to [RcmClient_releaseJobId\(\)](#), this error return value indicates that the given job id was not previously allocated with a call to [RcmClient_acquireJobId\(\)](#).

4.1.2.9 **#define RcmClient_E_LOSTMSG (-9)**

The expected reply message from the server was lost.

A command message was sent to the RcmServer but the reply message was not received. This is an internal error.

4.1.2.10 **#define RcmClient_E_MSGALLOCFAILED (-10)**

Insufficient memory to allocate a message.

The message heap cannot allocate a buffer of the requested size. The reported size is the requested data size and the underlying message header size.

4.1.2.11 **#define RcmClient_E_MSGFXNERROR (-6)**

Message function error.

There was an error encountered in either the message function or the library function invoked by the message function. The semantics of the error code are implementation dependent.

4.1.2.12 **#define RcmClient_E_MSGQCREATEFAILED (-11)**

The client message queue could not be created.

Each RcmClient instance must create its own message queue for receiving return messages from the RcmServer. The creation of this message queue failed, thus failing the RcmClient instance creation.

4.1.2.13 **#define RcmClient_E_MSGQOPENFAILED (-12)**

The server message queue could not be opened.

Each RcmClient instance must open the server's message queue. This error is raised when an internal error occurred while trying to open the server's message queue.

4.1.2.14 **#define RcmClient_E_SERVERERROR (-13)**

The server returned an unknown error code.

The server encountered an error with the given message but the error code is not recognized by the client.

4.1.2.15 #define RcmClient_E_SERVERNOTFOUND (-14)

The server specified in the create params was not found.

When creating an RcmClient instance, the specified server could not be found. This could occur if the server name is incorrect, or if the RcmClient instance is created before the RcmServer. In such an instance, the client can retry when the RcmServer is expected to have been created.

4.1.2.16 #define RcmClient_E_SYMBOLNOTFOUND (-15)

The given symbol was not found in the server symbol table.

This error could occur if the symbol spelling is incorrect or if the RcmServer is still loading its symbol table.

4.1.3 Typedef Documentation

4.1.3.1 typedef Void(* RcmClient_CallbackFxn)(RcmClient_Message *, Ptr)

Callback function type.

When using callback notification, the application must supply a callback function of this type. The callback will be invoked with the pointer to the [RcmClient_Message](#) returned from the server and the application data pointer supplied in the call to [RcmClient_execAsync\(\)](#).

4.1.4 Function Documentation

4.1.4.1 Int RcmClient_acquireJobId (RcmClient_Handle handle, UInt16 * jobId)

Get a job id from the server.

Acquire a unique job id from the server. The job id is used to associate messages with a common job id. The server will process all messages for a given job id in sequence.

4.1.4.2 Int RcmClient_addSymbol (RcmClient_Handle handle, String name, Fxn addr, UInt32 * index)

Add a symbol and its address to the server table.

This function is used by the client to dynamically load a new function address into the server's function pointer table. The given address must be in the server's address space. The function must already be loaded into the server's memory.

This function is useful when dynamically loading code onto the remote processor (as in the case of DLL's).

Parameters

in	<i>handle</i>	Handle to an instance object
in	<i>name</i>	The function's name.
in	<i>addr</i>	The function's address as specified in the remote processor's address space.
out	<i>index</i>	The function's index value to be used in the RcmClient_Message.fxIdx field.

Return values

<i>RcmClient_S_SUCCESS</i>	Success
<i>RcmClient_E_FAIL</i>	Failure

4.1.4.3 `Int RcmClient_alloc (RcmClient_Handle handle, UInt32 dataSize, RcmClient_Message ** message)`

Allocate a message from the heap configured for this instance.

When a message is allocated, the RcmClient instance is the owner of the message. All messages must be returned to the heap by calling [RcmClient_free\(\)](#).

During a call to all of the exec functions, the ownership of the message is temporarily transferred to the server. If the exec function returns an [RcmClient_Message](#) pointer, then ownership of the message is returned to the instance. For the other exec functions, the client acquires ownership of the return message by calling [RcmClient_waitUntilDone\(\)](#).

A message should not be accessed when ownership has been given away. Once ownership has been reacquired, the message can be either reused or returned to the heap.

Parameters

in	<i>handle</i>	Handle to an instance object
in	<i>dataSize</i>	Specifies (in chars) how much space to allocate for the RcmClient_Message.data array. The actual memory allocated from the heap will be larger as it includes the size of the internal message header.
out	<i>message</i>	A pointer to the allocated message or NULL on error.

4.1.4.4 `Int RcmClient_checkForError (RcmClient_Handle handle, RcmClient_Message ** returnMsg)`

Check if an error message has been returned from the server.

When using `RcmClient_execCmd()` to send messages to the server, the message will be freed by the server unless an error occurs. In the case of an error, the message is returned to the client. Use this function to check for and to retrieve these error messages.

Note that the latency of the return message is dependent on many system factors. In particular, the server's response time to processing a message will be a significant factor. It is possible to call `RcmClient_execCmd()` several times before any error message is returned. There is no way to know when all the messages have been processed.

The return value of `RcmClient_checkForError()` is designed to mimic the return value of `RcmClient_exec()`. When an error message is returned to the caller, the return value of `RcmClient_checkForError()` will be the appropriate error code as if the error had occurred during a call to `RcmClient_exec()`. For example, if a message is prepared with an incorrect function index, the return value from `RcmClient_exec()` would be `RcmClient_E_INVALIDFXNIDX`. However, the same message sent with `RcmClient_execCmd()` will not return an error, because the function does not wait for the return message. When the server receives the message and detects the function index error, it will return the message to the client on a special error queue. The subsequent call to `RcmClient_checkForError()` will pickup this error message and return with a status value of `RcmClient_E_INVALIDFXNIDX`, just as the call to `RcmClient_exec()` would have done.

A return value of `RcmClient_S_SUCCESS` means there are no error messages. This function will never return a message and a success status code at the same time.

When this function returns an error message, the caller must return the message to the heap by calling `RcmClient_free()`.

It is possible that `RcmClient_checkForError()` will return with an error but without an error message. This can happen when an internal error occurs in `RcmClient_checkForError()` before it has checked the error queue. In this case, an error is returned but the `returnMsg` argument will be set to `NULL`.

Parameters

in	<i>handle</i>	Handle to an instance object
out	<i>returnMsg</i>	A pointer to the error message or <code>NULL</code> if there are no error messages in the queue.

Return values

<i>RcmClient_S_SUCCESS</i>	
----------------------------	--

<i>RcmClient_E_- IPCERROR</i>	
<i>RcmClient_E_- INVALIDFXNIDX</i>	
<i>RcmClient_E_- MSGFXNERROR</i>	
<i>RcmClient_E_- SERVERERROR</i>	

4.1.4.5 **Int RcmClient_construct (RcmClient_Struct * structPtr, String server, const RcmClient_Params * params)**

Initialize a new instance object inside the provided structure.

This function is the same as [RcmClient_create\(\)](#) except that it does not allocate memory for the instance object. The instance object is constructed inside the provided structure. Call [RcmClient_destruct\(\)](#) to finalize a constructed instance object.

Parameters

in	<i>structPtr</i>	A pointer to an allocated structure.
in	<i>server</i>	The name of the server that messages will be sent to for executing commands. The name must be a system-wide unique name.
in	<i>params</i>	The create params used to customize the instance object.

See also

[RcmClient_create](#)

4.1.4.6 **Int RcmClient_create (String server, const RcmClient_Params * params, RcmClient_Handle * handle)**

Create an RcmClient instance.

The RcmClient instance is used by the application to send messages to an RcmServer for executing remote functions. A given instance can send messages only to the server it was configured for. If an application needs to send messages to multiple servers, then create an RcmClient instance for each server.

The assigned server to this instance must already exist and be running before creating RcmClient instances which send messages to it.

Parameters

in	<i>server</i>	The name of the server that messages will be sent to for executing commands. The name must be a system-wide unique name.
in	<i>params</i>	The create params used to customize the instance object.
out	<i>handle</i>	An opaque handle to the created instance object.

4.1.4.7 Int RcmClient_delete (RcmClient_Handle * *handlePtr*)

Delete an RcmClient instance.

Parameters

in, out	<i>handlePtr</i>	Handle to the instance object to delete.
---------	------------------	--

4.1.4.8 Int RcmClient_destruct (RcmClient_Struct * *structPtr*)

Finalize the instance object inside the provided structure.

Parameters

in	<i>structPtr</i>	A pointer to the structure containing the instance object to finalize.
----	------------------	--

4.1.4.9 Int RcmClient_exec (RcmClient_Handle *handle*, RcmClient_Message * *cmdMsg*, RcmClient_Message ** *returnMsg*)

Execute a command message on the server.

The message is sent to the server for processing. This call will block until the remote function has completed. When this function returns, the message will contain the return value of the remote function as well as a possibly modified context.

After calling exec, the message can be either reused for another call to exec or it can be freed.

Parameters

in	<i>handle</i>	Handle to an instance object
in	<i>cmdMsg</i>	Pointer to an RcmClient_Message structure.
out	<i>returnMsg</i>	A pointer to the return message or NULL on error. The client must free this message. The return value of the message function is stored in the results field. The return value of the library function is marshalled into the data field.

Return values

<i>RcmClient_S_- SUCCESS</i>	
<i>RcmClient_E_- EXECFAILED</i>	
<i>RcmClient_E_- INVALIDFXNIDX</i>	
<i>RcmClient_E_- LOSTMSG</i>	
<i>RcmClient_E_- MSGFXNERROR</i>	
<i>RcmClient_E_- SERVERERROR</i>	

4.1.4.10 `Int RcmClient.execAsync (RcmClient_Handle handle, RcmClient_Message * cmdMsg, RcmClient_CallbackFxn callback, Ptr appData)`

Execute a command message and use a callback for notification.

The message is sent to the server for execution, but this call does not wait for the remote function to execute. This call returns as soon as the message has been dispatched to the transport. Upon returning from this function, the ownership of the message has been lost; do not access the message at this time.

When the remote function completes, the given callback function is invoked by this RcmClient instance's callback server thread. The callback function is used to asynchronously notify the client that the remote function has completed.

The RcmClient instance must be create with callbackNotification set to true in order to use this function.

Parameters

in	<i>handle</i>	Handle to an instance object
in	<i>cmdMsg</i>	Pointer to an RcmClient_Message structure.
in	<i>callback</i>	A callback function pointer supplied by the application. It will be invoked by the callback server thread to notify the application that the remote function has completed.
in	<i>appData</i>	A private data pointer supplied by the application. This allows the application to provide its own context when receiving the callback.

4.1.4.11 `Int RcmClient_execCmd (RcmClient_Handle handle, RcmClient_Message * cmdMsg)`

Execute a one-way command message on the server.

The message is sent to the server for processing but this function does not wait for the return message. This function is non-blocking. The server will process the message and then free it, unless an error occurs. The return value from the remote function is discarded.

If an error occurs on the server while processing the message, the server will return the message to the client. Use [RcmClient_checkForError\(\)](#) to collect these return error messages.

When this function returns, ownership of the message has been transferred to the server. Do not access the message after this function returns, it could cause cache inconsistencies or a memory access violation. The `execCmd` replaces the `execNoReply` functionality. `RcmClient_execNoReply` is no longer supported.

Parameters

in	<i>handle</i>	Handle to an instance object
in	<i>cmdMsg</i>	Pointer to an RcmClient_Message structure.

Return values

<i>RcmClient_S_SUCCESS</i>	
<i>RcmClient_E_IPCERROR</i>	

4.1.4.12 `Int RcmClient_execDpc (RcmClient_Handle handle, RcmClient_Message * cmdMsg, RcmClient_Message ** returnMsg)`

Execute a deferred procedure call on the server.

The return field of the message is not used.

Parameters

in	<i>handle</i>	Handle to an instance object
in	<i>cmdMsg</i>	Pointer to an RcmClient_Message structure.
out	<i>returnMsg</i>	A pointer to the return message or NULL on error. The client must free this message.

4.1.4.13 Int RcmClient_execNoWait (RcmClient_Handle *handle*, RcmClient_Message * *cmdMsg*, UInt16 * *msgId*)

Submit a command message to the server and return immediately.

The message is sent to the server for execution but this call does not wait for the remote function to execute. The call returns as soon as the message has been dispatched to the transport. Upon returning from this function, the ownership of the message has been lost; do not access the message at this time.

Using this call to execute a remote message does not require a callback server thread. The application must call [RcmClient_waitUntilDone\(\)](#) to get the return message from the remote function.

Parameters

in	<i>handle</i>	Handle to an instance object
in	<i>cmdMsg</i>	A pointer to an RcmClient_Message structure.
out	<i>msgId</i>	Pointer used for storing the message id. Use the message id in a call to RcmClient_WaitUntilDone() to retrieve the return value of the remote function.

4.1.4.14 Void RcmClient_exit (Void)

Finalize the RcmClient module.

This function is used to finalize the RcmClient module. Any resources acquired by [RcmClient_init\(\)](#) will be released. Do not call any RcmClient functions after calling [RcmClient_exit\(\)](#).

This function must be serialized by the caller.

4.1.4.15 Int RcmClient_free (RcmClient_Handle *handle*, RcmClient_Message * *msg*)

Free the given message.

Parameters

in	<i>handle</i>	Handle to an instance object
	<i>msg</i>	Pointer to an RcmClient_Message structure.

4.1.4.16 Int RcmClient_getSymbolIndex (RcmClient_Handle *handle*, String *name*, UInt32 * *index*)

Return the function index from the server.

Query the server for the given function name and return its index. Use the index in the `fxnIdx` field of the [RcmClient_Message](#) struct.

Parameters

in	<i>handle</i>	Handle to an instance object
in	<i>name</i>	The function's name.
out	<i>index</i>	The function's index.

4.1.4.17 Void RcmClient_init (Void)

Initialize the RcmClient module.

This function is used to initialize the RcmClient module. Call this function before calling any other RcmClient function.

This function must be serialized by the caller

4.1.4.18 Int RcmClient_releaseJobId (RcmClient_Handle handle, UInt16 jobId)

Return a job id to the server and release all resources.

Parameters

in	<i>handle</i>	Handle to an instance object
in	<i>jobId</i>	The job id to be released

4.1.4.19 Int RcmClient_removeSymbol (RcmClient_Handle handle, String name)

Remove a symbol and from the server function table.

Useful when unloading a DLL from the server.

Parameters

in	<i>handle</i>	Handle to an instance object
in	<i>name</i>	The function name.

4.1.4.20 Int RcmClient_waitUntilDone (RcmClient_Handle handle, UInt16 msgId, RcmClient_Message ** returnMsg)

Block until the specified message has been executed.

This function will wait until the remote function invoked by the specified message has

completed. Upon return from this call, the message will contain the return value and the return context of the remote function.

Parameters

in	<i>handle</i>	Handle to an instance object
in	<i>msgId</i>	The message ID to wait for.
out	<i>returnMsg</i>	A pointer to the return message or NULL on error. The client must free this message.

4.2 RcmServer

Remote Command Message Server Module. An RcmServer processes inbound messages received from an RcmClient.

Data Structures

- struct [RcmServer_FxnDesc](#)
Function descriptor.
- struct [RcmServer_FxnDescAry](#)
Function descriptor array.
- struct [RcmServer_ThreadPoolDesc](#)
Worker pool descriptor.
- struct [RcmServer_ThreadPoolDescAry](#)
Worker pool descriptor array.
- struct [RcmServer_Params](#)
RcmServer Instance create parameters.
- struct [RcmServer_Struct](#)
Opaque client structure large enough to hold an instance object.

Defines

- #define [RcmServer_S_SUCCESS](#) (0)
Success return code.
- #define [RcmServer_E_FAIL](#) (-1)

General failure return code.

- #define `RcmServer_E_NOMEMORY` (-2)
There was insufficient memory left on the heap.
- #define `RcmServer_E_SYMBOLNOTFOUND` (-3)
The given symbol was not found in the server's symbol table.
- #define `RcmServer_E_SYMBOLSTATIC` (-4)
The given symbols is in the static table, it cannot be removed.
- #define `RcmServer_E_SYMBOLTABLEFULL` (-5)
The server's symbol table is full.

Typedefs

- typedef `Int32(* RcmServer_MsgFxn)(UInt32, UInt32 *)`
Remote function type.
- typedef struct `RcmServer_Object_tag * RcmServer_Handle`
RcmServer instance object handle.

Functions

- Int `RcmServer_addSymbol` (`RcmServer_Handle` handle, String name, `RcmServer_MsgFxn` addr, `UInt32 *index`)
Add a symbol to the server's function table.
- Int `RcmServer_construct` (`RcmServer_Struct *structPtr`, String name, const `RcmServer_Params *params`)
Initialize a new instance object inside the provided structure.
- Int `RcmServer_create` (String name, `RcmServer_Params *params`, `RcmServer_Handle *handle`)
Create an RcmServer instance.
- Int `RcmServer_delete` (`RcmServer_Handle *handlePtr`)
Delete an RcmServer instance.
- Int `RcmServer_destruct` (`RcmServer_Struct *structPtr`)

Finalize the instance object inside the provided structure.

- Void [RcmServer_exit](#) (Void)
Finalize the RcmServer module.
- Void [RcmServer_init](#) (Void)
Initialize the RcmServer module.
- Void [RcmServer_Params_init](#) (RcmServer_Params *params)
Initialize the instance create params structure.
- Int [RcmServer_removeSymbol](#) (RcmServer_Handle handle, String name)
Remove a symbol and from the server's function table.
- Int [RcmServer_start](#) (RcmServer_Handle handle)
Start the server.

4.2.1 Detailed Description

Remote Command Message Server Module. An RcmServer processes inbound messages received from an RcmClient. The RcmServer module is used to create an server instance. RcmClients send their messages to only one RcmServer instance for processing. The server instance can be created with a function dispatch table and additional tables can be added as requested by the clients.

Diagnostics

Diags_INFO - full detail log events Diags_USER1 - message processing

See also

[RcmClient](#)
RCM Overview

4.2.2 Define Documentation

4.2.2.1 #define RcmServer_E_SYMBOLNOTFOUND (-3)

The given symbol was not found in the server's symbol table.

This error could occur if the symbol spelling is incorrect or if the RcmServer is still loading its symbol table.

4.2.2.2 #define RcmServer_E_SYMBOLSTATIC (-4)

The given symbols is in the static table, it cannot be removed.

All symbols installed at instance create time are added to the static symbol table. They cannot be removed. The static symbol table must remain intact for the lifespan of the server instance.

4.2.2.3 #define RcmServer_E_SYMBOLTABLEFULL (-5)

The server's symbol table is full.

The symbol table is full. You must remove some symbols before any new symbols can be added.

4.2.3 Typedef Documentation

4.2.3.1 typedef Int32(* RcmServer_MsgFxn)(UInt32, UInt32 *)

Remote function type.

All functions executed by the RcmServer must be of this type. Typically, these functions are simply wrappers to the vendor function. The server invokes this remote function by passing in the [RcmClient_Message.dataSize](#) field and the address of the [RcmClient_Message.data](#) array.

```
RcmServer_MsgFxn fxn = ...;
RcmClient_Message *msg = ...;
msg->result = (*fxn)(msg->dataSize, msg->data);
```

4.2.4 Function Documentation

4.2.4.1 Int RcmServer_addSymbol (RcmServer_Handle handle, String name, RcmServer_MsgFxn addr, UInt32 * index)

Add a symbol to the server's function table.

This function adds a new symbol to the server's function table. This is useful for supporting Dynamic Load Libraries (DLLs).

Parameters

in	<i>handle</i>	Handle to an instance object.
in	<i>name</i>	The function's name.
in	<i>addr</i>	The function's address in the server's address space.
out	<i>index</i>	The function's index value to be used in the RcmClient_Message.fxndx field.

Return values

<i>RcmClient_S_- SUCCESS</i>	
<i>RcmServer_E_- NOMEMORY</i>	
<i>RcmServer_E_- SYMBOLTABLEFULL</i>	

4.2.4.2 Int RcmServer_construct (RcmServer_Struct * structPtr, String name, const RcmServer_Params * params)

Initialize a new instance object inside the provided structure.

This function is the same as [RcmServer_create\(\)](#) except that it does not allocate memory for the instance object. The instance object is constructed inside the provided structure. Call [RcmServer_destruct\(\)](#) to finalize a constructed instance object.

Parameters

in	<i>structPtr</i>	A pointer to an allocated structure.
in	<i>name</i>	The name of the server. The RcmClient will locate a server instance using this name. It must be unique to the system.
in	<i>params</i>	The create params used to customize the instance object.

Return values

<i>RcmClient_S_- SUCCESS</i>	
<i>RcmServer_E_FAIL</i>	
<i>RcmServer_E_- NOMEMORY</i>	

See also

[RcmServer_create](#)

4.2.4.3 Int RcmServer_create (String name, RcmServer_Params * params, RcmServer_Handle * handle)

Create an RcmServer instance.

A server instance is used to execute functions on behalf of an RcmClient instance. There can be multiple server instances on any given CPU. The servers typically reside on a remote CPU from the RcmClient instance.

Parameters

in	<i>name</i>	The name of the server. The RcmClient will locate a server instance using this name. It must be unique to the system.
in	<i>params</i>	The create params used to customize the instance object.
out	<i>handle</i>	An opaque handle to the created instance object.

Return values

<i>RcmClient_S_- SUCCESS</i>	
<i>RcmServer_E_FAIL</i>	
<i>RcmServer_E_- NOMEMORY</i>	

4.2.4.4 Int RcmServer_delete (RcmServer_Handle * handlePtr)

Delete an RcmServer instance.

Parameters

in, out	<i>handlePtr</i>	Handle to the instance object to delete.
---------	------------------	--

4.2.4.5 Int RcmServer_destruct (RcmServer_Struct * structPtr)

Finalize the instance object inside the provided structure.

Parameters

in	<i>structPtr</i>	A pointer to the structure containing the instance object to finalize.
----	------------------	--

4.2.4.6 Void RcmServer_exit (Void)

Finalize the RcmServer module.

This function is used to finalize the RcmServer module. Any resources acquired by [RcmServer_init\(\)](#) will be released. Do not call any RcmServer functions after calling [RcmServer_exit\(\)](#).

This function must be serialized by the caller.

4.2.4.7 Void RcmServer_init (Void)

Initialize the RcmServer module.

This function is used to initialize the RcmServer module. Call this function before calling any other RcmServer function.

This function must be serialized by the caller

4.2.4.8 Int RcmServer_removeSymbol (RcmServer_Handle handle, String name)

Remove a symbol and from the server's function table.

Useful when unloading a DLL from the server.

Parameters

in	<i>handle</i>	Handle to an instance object.
in	<i>name</i>	The function's name.

Return values

<i>RcmClient_S_- SUCCESS</i>	
<i>RcmServer_E_- SYMBOLNOTFOUND</i>	
<i>RcmServer_E_- SYMBOLSTATIC</i>	

4.2.4.9 Int RcmServer_start (RcmServer_Handle handle)

Start the server.

The server is created in stop mode. It will not start processing messages until it has been started.

Parameters

in	<i>handle</i>	Handle to an instance object.
----	---------------	-------------------------------

Return values

<i>RcmClient_S_- SUCCESS</i>	
<i>RcmServer_E_FAIL</i>	

4.3 RcmTypes

Remote Command Message Type definitions. RCM types are common definitions between RCM server and RCM client.

Data Structures

- struct [RcmClient_Packet](#)
Remote Command Message (RCM) packet structure.

Defines

- #define [RcmClient_Desc_RCM_MSG](#) 0x1
RCM Client Exec Message.
- #define [RcmClient_Desc_DPC](#) 0x2
RCM Client deferred procedure call.
- #define [RcmClient_Desc_SYM_ADD](#) 0x3
RCM Client symbol add message.
- #define [RcmClient_Desc_SYM_IDX](#) 0x4
RCM Client query symbol index.
- #define [RcmClient_Desc_CMD](#) 0x5
RCM Client cmd message (one-way)
- #define [RcmClient_Desc_JOB_ACQ](#) 0x6
RCM Client acquire a job id.
- #define [RcmClient_Desc_JOB_REL](#) 0x7
RCM Client release a job id.
- #define [RcmClient_Desc_TYPE_MASK](#) 0x0F00
RCM Client BIT field mask.
- #define [RcmClient_Desc_TYPE_SHIFT](#) 8
RCM Client BIT field shift width.
- #define [RcmServer_Status_SUCCESS](#) ((UInt16)0)

RCM Server success.

- #define `RcmServer_Status_INVALID_FXN` ((UInt16)1)
RCM Server invalid fxn index.
- #define `RcmServer_Status_SYMBOL_NOT_FOUND` ((UInt16)2)
RCM Server symbol not found.
- #define `RcmServer_Status_INVALID_MSG_TYPE` ((UInt16)3)
RCM Server invalid msg type.
- #define `RcmServer_Status_MSG_FXN_ERR` ((UInt16)4)
RCM Server message function error.
- #define `RcmServer_Status_Error` ((UInt16)5)
RCM Server general failure.
- #define `RcmServer_Status_Unprocessed` ((UInt16)6)
RCM Server deferred procedure call.
- #define `RcmServer_Status_JobNotFound` ((UInt16)7)
RCM Server job id not found.
- #define `RcmServer_Status_PoolNotFound` ((UInt16)8)
RCM Server pool id not found.

Functions

- `Void * _memset` (Void *, Int c, Int n)
- `Int _strcmp` (Char *s, Char *t)
- `Void _strcpy` (Char *s, Char *t)
- `Int _strlen` (const Char *s)

4.3.1 Detailed Description

Remote Command Message Type definitions. RCM types are common definitions between RCM server and RCM client. Remote Command Message - message based remote function execution

The Remote Command Message (RCM) package provides a client/server implementation for executing functions on a remote processor. The package contains the following modules.

- [RcmClient](#) - Client Module
- [RcmServer](#) - Server Module

This package provides modules for a client/server based implementation for executing functions on a remote processor. The IPC layer is based on messages. The application uses the [RcmClient](#) module to allocate a message, specify the remote function and its arguments, and then send the message to the server for execution. When the message returns, it contains the remote function's return values. The server is typically on a remote processor, but local server instances (on the same processor as the client) are also supported.

The following image provides an overview of a typical use case. An application running on the local cpu creates an [RcmClient](#) instance. Using the [RcmClient](#) instance handle, the application has access to all the necessary operations for executing a message on the remote server. The application must fill in the message with the remote function index, as supplied by the server, and the context needed by the remote function to execute.

The [RcmClient](#) module is used to create instance objects. Each instance can send messages only to the server specified at create time. You create an instance for each server you want to send messages to. At create time the heap must be specified. The heap is used by the instance for allocating messages.

There are four types of execute commands:

- synchronous blocking
- synchronous non-blocking
- synchronous deferred
- asynchronous non-blocking

Here is an overview of the exec commands. See the [RcmClient](#) module for more details on each function.

- [RcmClient_exec\(\)](#) - synchronous blocking This function sends a message to the server and waits for the server to execute the message and to send the return message back to the client.
- [RcmClient_execNoWait\(\)](#) - synchronous non-blocking This function sends a message to the server but does not wait for the return message. Control returns to the caller as soon as the message has been delivered to the underlying transport. The server will execute the message and send the return message back to the client where it is placed in a mailbox. The client collects the return message with a call to [RcmClient_waitUntilDone\(\)](#) which is a blocking call. This is useful when the client has more work to do while the remote function is executing. It is also efficient because it does not require a local callback thread.

- [RcmClient_execDpc\(\)](#) - synchronous deferred This function sends a message to the server and waits for the return message. When the server receives the message, it does not immediately execute it. Instead, it copies the message context into a local buffer and sends the message back to the client. Only after the message has been sent to the client, does the server execute the function. This is useful for server shutdown. This allows the client to regain ownership of the message and to return it to the heap and allows the server to shutdown without owning any resources allocated by the client.
- [RcmClient_execAsync\(\)](#) - asynchronous non-blocking This function sends a message to the server but does not wait for the return message. Control returns to the caller as in the synchronous non-blocking case above. The server will execute the message and send the return message back to the client for delivery to a callback thread. When the return message arrives, the callback thread will invoke an application callback to notify the application that the return message has arrived. This is the only asynchronous part of the sequence as the application will be preempted by the callback thread. It is the application's responsibility to periodically monitor a flag (set by the callback) to complete the message delivery.

The package descriptor field. Note that the format is different for out-bound and in-bound messages.

RcmClient to RcmServer Packet

There are four types of execute commands:

- Bits [15:12] reserved
- Bits [11: 8] message type
- Bits [7: 0] client protocol version

RcmServer to RcmClient Packet

There are four types of execute commands:

- Bits [15:12] reserved
- Bits [11: 8] server status code
- Bits [7: 0] server protocol version

Chapter 5

Data Structure Documentation

5.1 RcmClient_Message Struct Reference

Remote Command Message structure.

```
#include <RcmClient.h>
```

Data Fields

- UInt16 `poolId`
The worker pool id that will process this message.
- UInt16 `jobId`
The job id associated with this message.
- UInt32 `fxnIdx`
The index of the remote function to execute.
- Int32 `result`
The return value of the remote message function.
- UInt32 `dataSize`
The size of the data buffer (in chars).
- UInt32 `data` [1]
The data buffer containing the message payload.

5.1.1 Detailed Description

Remote Command Message structure. An RcmClient needs to fill in this message before sending it to the RcmServer for execution.

5.1.2 Field Documentation

5.1.2.1 UInt32 data[1]

The data buffer containing the message payload.

The size of this field is dataSize chars. The space is allocated by the call to the [RcmClient_alloc\(\)](#) function.

5.1.2.2 UInt32 dataSize

The size of the data buffer (in chars).

This field should be considered as read-only. It is set by the call to the [RcmClient_alloc\(\)](#) function.

5.1.2.3 UInt16 jobId

The job id associated with this message.

All messages belonging to a job id must have this field set to that id. Use the value RcmClient_DISCRETEJOBID if the message does not belong to any job.

5.1.2.4 UInt16 poolId

The worker pool id that will process this message.

The message will be processed by a worker thread from the worker pool specified in this field. The default value is the default pool id.

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

- [ti/sdo/rcm/RcmClient.h](#)

5.2 RcmClient_Packet Struct Reference

Remote Command Message (RCM) packet structure.

```
#include <RcmTypes.h>
```

Data Fields

- MessageQ_MsgHeader [msgqHeader](#)
Message Header.
- UInt16 [desc](#)
Descriptor.
- UInt16 [msgId](#)
Message ID.
- [RcmClient_Message message](#)
RCM Message.

5.2.1 Detailed Description

Remote Command Message (RCM) packet structure. The packet structure (actual message send to server)

5.2.2 Field Documentation

5.2.2.1 UInt16 desc

Descriptor.

The Message type or protocol OR RCM server status

5.2.2.2 RcmClient_Message message

RCM Message.

RCM Message from client to server. Message body (5 words + payload)

5.2.2.3 UInt16 msgId

Message ID.

The Message ID used by RCM client and server

5.2.2.4 MessageQ_MsgHeader msgqHeader

Message Header.

The Message header used by MessageQ module (8 words)

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

- [ti/sdo/rcm/RcmTypes.h](#)

5.3 RcmClient_Params Struct Reference

Instance create parameters.

```
#include <RcmClient.h>
```

Data Fields

- UInt16 [heapId](#)
The heapId used by this instance for allocating messages.
- Bool [callbackNotification](#)
Asynchronous callback notification support.

5.3.1 Detailed Description

Instance create parameters.

5.3.2 Field Documentation

5.3.2.1 Bool callbackNotification

Asynchronous callback notification support.

When remote functions submitted with [RcmClient_execAsync\(\)](#) complete, the given callback function is invoked. The callback function executes in the context of this RcmClient instance's callback server thread.

This config param must be set to true when using [RcmClient_execAsync\(\)](#) to execute remote functions.

When set to false, the callback server thread is not created.

5.3.2.2 UInt16 heapId

The heapId used by this instance for allocating messages.

If sending messages to a remote server, the specified heap must be compatible with the transport used for delivering messages to the remote processor.

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

- [ti/sdo/rcm/RcmClient.h](#)

5.4 RcmClient_Struct Struct Reference

Opaque client structure large enough to hold an instance object.

```
#include <RcmClient.h>
```

Data Fields

- `xdc_runtime_knl_GateThread_Struct_f1`
- `Ptr_f2`
- `Ptr_f3`
- `UInt16_f4`
- `Ptr_f5`
- `UInt32_f6`
- `Bool_f7`
- `UInt16_f8`
- `Ptr_f9`
- `Ptr_f10`
- `Ptr_f11`
- `Ptr_f12`

5.4.1 Detailed Description

Opaque client structure large enough to hold an instance object. Use this structure to define an embedded RcmClient object.

See also

[RcmClient_construct](#)

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

- [ti/sdo/rcm/RcmClient.h](#)

5.5 RcmServer_FxnDesc Struct Reference

Function descriptor.

```
#include <RcmServer.h>
```

Data Fields

- String [name](#)
The name of the function.
- [RcmServer_MsgFxn addr](#)
The function address in the server's address space.

5.5.1 Detailed Description

Function descriptor. This function descriptor is used internally in the server. Its values are defined either at config time by the server's [RcmServer_Params.fxns](#) create parameter or at runtime through a call to [RcmClient_addSymbol\(\)](#).

5.5.2 Field Documentation

5.5.2.1 RcmServer_MsgFxn addr

The function address in the server's address space.

The server will ultimately branch to this address.

5.5.2.2 String name

The name of the function.

The name is used for table lookup, it does not search the actual symbol table. You can provide any string as long as it is unique and the client uses the same name for lookup.

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

- [ti/sdo/rcm/RcmServer.h](#)

5.6 RcmServer_FxnDescAry Struct Reference

Function descriptor array.

```
#include <RcmServer.h>
```

Data Fields

- Int [length](#)
The length of the array.
- [RcmServer_FxnDesc](#) * [elem](#)
Pointer to the array.

5.6.1 Detailed Description

Function descriptor array.

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

- [ti/sdo/rcm/RcmServer.h](#)

5.7 RcmServer_Params Struct Reference

RcmServer Instance create parameters.

```
#include <RcmServer.h>
```

Data Fields

- Thread_Priority [priority](#)
Server thread priority.
- Int [osPriority](#)
Server thread priority (OS-specific).
- SizeT [stackSize](#)
The stack size in bytes of the server thread.
- String [stackSeg](#)
The server thread stack placement.
- [RcmServer_ThreadPoolDesc](#) [defaultPool](#)
The default thread pool used for anonymous messages.

- [RcmServer_ThreadPoolDescAry workerPools](#)
Array of thread pool descriptors.
- [RcmServer_FxnDescAry fxns](#)
Array of function names to install into the server.

5.7.1 Detailed Description

RcmServer Instance create parameters.

5.7.2 Field Documentation

5.7.2.1 RcmServer_FxnDescAry fxns

Array of function names to install into the server.

The functions declared with this instance parameter are statically bound to the server, they persist for the life of the server. They cannot be removed with a call to [RcmServer_removeSymbol\(\)](#).

To specify a function address, use the & character in the string as in the following example.

```
RcmServer_FxnDesc serverFxnAry[] = {
    { "LED_on",          LED_on  },
    { "LED_off",        LED_off  }
};

#define serverFxnAryLen (sizeof serverFxnAry / sizeof serverFxnAry[0])

RcmServer_FxnDescAry Server_fxnTab = {
    serverFxnAryLen,
    serverFxnAry
};

RcmServer_Params rcmServerP;
RcmServer_Params_init(&rcmServerP);
rcmServerP.fxns.length = Server_fxnTab.length;
rcmServerP.fxns.elem = Server_fxnTab.elem;

RcmServer_Handle rcmServerH;
rval = RcmServer_create("ServerA", &rcmServerP, &(rcmServerH));
```

5.7.2.2 Int osPriority

Server thread priority (OS-specific).

This value is Operating System specific. It determines the execution priority of the server thread. If this attribute is set, it takes precedence over the priority attribute below. The server thread reads the incoming message and then either executes the message function (in-band execution) or dispatches the message to a thread pool (out-of-band execution). The server thread then wait on the next message.

5.7.2.3 Thread.Priority priority

Server thread priority.

This value is Operating System independent. It determines the execution priority of the server thread. The server thread reads the incoming message and then either executes the message function (in-band execution) or dispatches the message to a thread pool (out-of-band execution). The server thread then wait on the next message.

5.7.2.4 RcmServer_ThreadPoolDescAry workerPools

Array of thread pool descriptors.

The worker pools declared with this instance parameter are statically bound to the server, they persist for the life of the server. They cannot be removed with a call to `RcmServer_deletePool()`. However, worker threads may be created or deleted at runtime.

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

- [ti/sdo/rcm/RcmServer.h](#)

5.8 RcmServer_Struct Struct Reference

Opaque client structure large enough to hold an instance object.

```
#include <RcmServer.h>
```

Data Fields

- GateThread_Struct_f1
- Ptr_f2
- Ptr_f3
- Ptr_f4
- struct {
 - Int_f1
 - Ptr_f2
- }_f5

- [Ptr_f6](#) [9]
- [UInt16_f7](#)
- [UInt16_f8](#)
- [Bool_f9](#)
- [Int_f10](#)
- [Ptr_f11](#) [4]
- [Ptr_f12](#)

5.8.1 Detailed Description

Opaque client structure large enough to hold an instance object. Use this structure to define an embedded RcmServer object.

See also

[RcmServer_construct](#)

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

- [ti/sdo/rcm/RcmServer.h](#)

5.9 RcmServer_ThreadPoolDesc Struct Reference

Worker pool descriptor.

```
#include <RcmServer.h>
```

Data Fields

- String [name](#)
The name of the worker pool.
- UInt [count](#)
The number of worker threads in the pool.
- Thread_Priority [priority](#)
The priority of all threads in the worker pool.
- Int [osPriority](#)
The priority (OS-specific) of all threads in the worker pool.

- SizeT [stackSize](#)
The stack size in bytes of a worker thread.
- String [stackSeg](#)
The worker thread stack placement.

5.9.1 Detailed Description

Worker pool descriptor. Use this data structure to define a worker pool to be created either at the server create time or dynamically at runtime.

5.9.2 Field Documentation

5.9.2.1 Int osPriority

The priority (OS-specific) of all threads in the worker pool.

This value is Operating System specific. It determines the execution priority of all the worker threads in the pool. If this property is set, it takes precedence over the priority property above.

5.9.2.2 Thread.Priority priority

The priority of all threads in the worker pool.

This value is Operating System independent. It determines the execution priority of all the worker threads in the pool.

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

- [ti/sdo/rcm/RcmServer.h](#)

5.10 RcmServer_ThreadPoolDescAry Struct Reference

Worker pool descriptor array.

```
#include <RcmServer.h>
```

Data Fields

- Int [length](#)

The length of the array.

- [RcmServer_ThreadPoolDesc](#) * elem

Pointer to the array.

5.10.1 Detailed Description

Worker pool descriptor array.

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

- [ti/sdo/rcm/RcmServer.h](#)

Chapter 6

File Documentation

6.1 ti/sdo/rcm/RcmClient.h File Reference

Remote Command Message Client Module. An RcmClient is used for sending messages to an RcmServer for processing.

```
#include <xdc/runtime/knl/GateThread.h>
```

Data Structures

- struct [RcmClient_Message](#)
Remote Command Message structure.
- struct [RcmClient_Params](#)
Instance create parameters.
- struct [RcmClient_Struct](#)
Opaque client structure large enough to hold an instance object.

Defines

- #define [RcmClient_S_SUCCESS](#) (0)
Success return code.
- #define [RcmClient_E_FAIL](#) (-1)
General failure return code.

- #define `RcmClient_E_EXECASYNCTENABLED` (-2)
The client has not been configured for asynchronous notification.
- #define `RcmClient_E_EXECFAILED` (-3)
The client was unable to send the command message to the server.
- #define `RcmClient_E_INVALIDHEAPID` (-4)
A heap id must be provided in the create params.
- #define `RcmClient_E_INVALIDFXNIDX` (-5)
Invalid function index.
- #define `RcmClient_E_MSGFXNERROR` (-6)
Message function error.
- #define `RcmClient_E_IPCERROR` (-7)
An unknown error has been detected from the IPC layer.
- #define `RcmClient_E_LISTCREATEFAILED` (-8)
Failed to create the list object.
- #define `RcmClient_E_LOSTMSG` (-9)
The expected reply message from the server was lost.
- #define `RcmClient_E_MSGALLOCFAILED` (-10)
Insufficient memory to allocate a message.
- #define `RcmClient_E_MSGQCREATEFAILED` (-11)
The client message queue could not be created.
- #define `RcmClient_E_MSGQOPENFAILED` (-12)
The server message queue could not be opened.
- #define `RcmClient_E_SERVERERROR` (-13)
The server returned an unknown error code.
- #define `RcmClient_E_SERVERNOTFOUND` (-14)
The server specified in the create params was not found.
- #define `RcmClient_E_SYMBOLNOTFOUND` (-15)
The given symbol was not found in the server symbol table.

- #define `RcmClient_E_NOMEMORY` (-16)
There is insufficient memory left in the heap.
- #define `RcmClient_E_JOBIDNOTFOUND` (-17)
The given job id was not found on the server.
- #define `RcmClient_INVALIDFXNIDX` ((UInt32)(0xFFFFFFFF))
Invalid function index.
- #define `RcmClient_INVALIDHEAPID` ((UInt16)(0xFFFF))
Invalid heap id.
- #define `RcmClient_INVALIDMSGID` (0)
Invalid message id.
- #define `RcmClient_DEFAULTPOOLID` ((UInt16)(0x8000))
Default worker pool id.
- #define `RcmClient_DISCRETEJOBID` (0)
Invalid job stream id.

Typedefs

- typedef struct RcmClient_Object_tag * `RcmClient_Handle`
RcmClient instance object handle.
- typedef Void(* `RcmClient_CallbackFxn`)(RcmClient_Message *, Ptr)
Callback function type.

Functions

- Int `RcmClient_acquireJobId` (RcmClient_Handle handle, UInt16 *jobId)
Get a job id from the server.
- Int `RcmClient_addSymbol` (RcmClient_Handle handle, String name, Fxn addr, UInt32 *index)
Add a symbol and its address to the server table.
- Int `RcmClient_alloc` (RcmClient_Handle handle, UInt32 dataSize, RcmClient_Message **message)

Allocate a message from the heap configured for this instance.

- Int `RcmClient_checkForError` (`RcmClient_Handle` handle, `RcmClient_Message` `**returnMsg`)
Check if an error message has been returned from the server.
- Int `RcmClient_construct` (`RcmClient_Struct` `*structPtr`, String server, const `RcmClient_Params` `*params`)
Initialize a new instance object inside the provided structure.
- Int `RcmClient_create` (String server, const `RcmClient_Params` `*params`, `RcmClient_Handle` `*handle`)
Create an RcmClient instance.
- Int `RcmClient_delete` (`RcmClient_Handle` `*handlePtr`)
Delete an RcmClient instance.
- Int `RcmClient_destruct` (`RcmClient_Struct` `*structPtr`)
Finalize the instance object inside the provided structure.
- Int `RcmClient_exec` (`RcmClient_Handle` handle, `RcmClient_Message` `*cmdMsg`, `RcmClient_Message` `**returnMsg`)
Execute a command message on the server.
- Int `RcmClient_execAsync` (`RcmClient_Handle` handle, `RcmClient_Message` `*cmdMsg`, `RcmClient_CallbackFxn` callback, Ptr appData)
Execute a command message and use a callback for notification.
- Int `RcmClient_execCmd` (`RcmClient_Handle` handle, `RcmClient_Message` `*cmdMsg`)
Execute a one-way command message on the server.
- Int `RcmClient_execDpc` (`RcmClient_Handle` handle, `RcmClient_Message` `*cmdMsg`, `RcmClient_Message` `**returnMsg`)
Execute a deferred procedure call on the server.
- Int `RcmClient_execNoWait` (`RcmClient_Handle` handle, `RcmClient_Message` `*cmdMsg`, UInt16 `*msgId`)
Submit a command message to the server and return immediately.
- Void `RcmClient_exit` (Void)
Finalize the RcmClient module.

- Int [RcmClient_free](#) ([RcmClient_Handle](#) handle, [RcmClient_Message](#) *msg)
Free the given message.
- Int [RcmClient_getSymbolIndex](#) ([RcmClient_Handle](#) handle, String name, UInt32 *index)
Return the function index from the server.
- Void [RcmClient_init](#) (Void)
Initialize the RcmClient module.
- Void [RcmClient_Params_init](#) ([RcmClient_Params](#) *params)
Initialize the instance create params structure.
- Int [RcmClient_releaseJobId](#) ([RcmClient_Handle](#) handle, UInt16 jobId)
Return a job id to the server and release all resources.
- Int [RcmClient_removeSymbol](#) ([RcmClient_Handle](#) handle, String name)
Remove a symbol and from the server function table.
- Int [RcmClient_waitUntilDone](#) ([RcmClient_Handle](#) handle, UInt16 msgId, [RcmClient_Message](#) **returnMsg)
Block until the specified message has been executed.

6.1.1 Detailed Description

Remote Command Message Client Module. An RcmClient is used for sending messages to an RcmServer for processing.

See also

[RcmServer.h](#)
[RCM Overview](#)

6.2 ti/sdo/rcm/RcmServer.h File Reference

Remote Command Message Server Module. An RcmServer processes inbound messages received from an RcmClient.

```
#include <xdc/runtime/knl/GateThread.h>  
#include <xdc/runtime/knl/Thread.h>
```

Data Structures

- struct [RcmServer_FxnDesc](#)
Function descriptor.
- struct [RcmServer_FxnDescAry](#)
Function descriptor array.
- struct [RcmServer_ThreadPoolDesc](#)
Worker pool descriptor.
- struct [RcmServer_ThreadPoolDescAry](#)
Worker pool descriptor array.
- struct [RcmServer_Params](#)
RcmServer Instance create parameters.
- struct [RcmServer_Struct](#)
Opaque client structure large enough to hold an instance object.

Defines

- #define [RcmServer_S_SUCCESS](#) (0)
Success return code.
- #define [RcmServer_E_FAIL](#) (-1)
General failure return code.
- #define [RcmServer_E_NOMEMORY](#) (-2)
There was insufficient memory left on the heap.
- #define [RcmServer_E_SYMBOLNOTFOUND](#) (-3)
The given symbol was not found in the server's symbol table.
- #define [RcmServer_E_SYMBOLSTATIC](#) (-4)
The given symbols is in the static table, it cannot be removed.
- #define [RcmServer_E_SYMBOLTABLEFULL](#) (-5)
The server's symbol table is full.

Typedefs

- typedef Int32(* [RcmServer_MsgFxn](#))(UInt32, UInt32 *)
Remote function type.
- typedef struct RcmServer_Object_tag * [RcmServer_Handle](#)
RcmServer instance object handle.

Functions

- Int [RcmServer_addSymbol](#) ([RcmServer_Handle](#) handle, String name, [RcmServer_MsgFxn](#) addr, UInt32 *index)
Add a symbol to the server's function table.
- Int [RcmServer_construct](#) ([RcmServer_Struct](#) *structPtr, String name, const [RcmServer_Params](#) *params)
Initialize a new instance object inside the provided structure.
- Int [RcmServer_create](#) (String name, [RcmServer_Params](#) *params, [RcmServer_Handle](#) *handle)
Create an RcmServer instance.
- Int [RcmServer_delete](#) ([RcmServer_Handle](#) *handlePtr)
Delete an RcmServer instance.
- Int [RcmServer_destruct](#) ([RcmServer_Struct](#) *structPtr)
Finalize the instance object inside the provided structure.
- Void [RcmServer_exit](#) (Void)
Finalize the RcmServer module.
- Void [RcmServer_init](#) (Void)
Initialize the RcmServer module.
- Void [RcmServer_Params_init](#) ([RcmServer_Params](#) *params)
Initialize the instance create params structure.
- Int [RcmServer_removeSymbol](#) ([RcmServer_Handle](#) handle, String name)
Remove a symbol and from the server's function table.
- Int [RcmServer_start](#) ([RcmServer_Handle](#) handle)
Start the server.

6.2.1 Detailed Description

Remote Command Message Server Module. An RcmServer processes inbound messages received from an RcmClient. The RcmServer processes inbound messages received from an RcmClient. After processing a message, the server will return the message to the client.

See also

[RcmClient.h](#)
[RCM Overview](#)

6.3 ti/sdo/rcm/RcmTypes.h File Reference

Remote Command Message (RCM) common definitions between Client and Server.

```
#include <ti/ipc/MessageQ.h>
#include <ti/sdo/rcm/RcmClient.h>
```

Data Structures

- struct [RcmClient_Packet](#)
Remote Command Message (RCM) packet structure.

Defines

- #define [RcmClient_Desc_RCM_MSG](#) 0x1
RCM Client Exec Message.
- #define [RcmClient_Desc_DPC](#) 0x2
RCM Client deferred procedure call.
- #define [RcmClient_Desc_SYM_ADD](#) 0x3
RCM Client symbol add message.
- #define [RcmClient_Desc_SYM_IDX](#) 0x4
RCM Client query symbox index.
- #define [RcmClient_Desc_CMD](#) 0x5
RCM Client cmd message (one-way)

- #define `RcmClient_Desc_JOB_ACQ` 0x6
RCM Client acquire a job id.
- #define `RcmClient_Desc_JOB_REL` 0x7
RCM Client release a job id.
- #define `RcmClient_Desc_TYPE_MASK` 0x0F00
RCM Client BIT field mask.
- #define `RcmClient_Desc_TYPE_SHIFT` 8
RCM Client BIT field shift width.
- #define `RcmServer_Status_SUCCESS` ((UInt16)0)
RCM Server success.
- #define `RcmServer_Status_INVALID_FXN` ((UInt16)1)
RCM Server invalid fxn index.
- #define `RcmServer_Status_SYMBOL_NOT_FOUND` ((UInt16)2)
RCM Server symbol not found.
- #define `RcmServer_Status_INVALID_MSG_TYPE` ((UInt16)3)
RCM Server invalid msg type.
- #define `RcmServer_Status_MSG_FXN_ERR` ((UInt16)4)
RCM Server message function error.
- #define `RcmServer_Status_Error` ((UInt16)5)
RCM Server general failure.
- #define `RcmServer_Status_Unprocessed` ((UInt16)6)
RCM Server deferred procedure call.
- #define `RcmServer_Status_JobNotFound` ((UInt16)7)
RCM Server job id not found.
- #define `RcmServer_Status_PoolNotFound` ((UInt16)8)
RCM Server pool id not found.

Functions

- Void * **_memset** (Void *s, Int c, Int n)
- Int **_strcmp** (Char *s, Char *t)
- Void **_strcpy** (Char *s, Char *t)
- Int **_strlen** (const Char *s)

6.3.1 Detailed Description

Remote Command Message (RCM) common definitions between Client and Server.

See also

[RcmTypes.h](#)
[RCM Common definitions](#)

Index

addr
 RcmServer_FxnDesc, 42

callbackNotification
 RcmClient_Params, 40

data
 RcmClient_Message, 38

dataSize
 RcmClient_Message, 38

desc
 RcmClient_Packet, 39

fxns
 RcmServer_Params, 44

heapId
 RcmClient_Params, 40

jobId
 RcmClient_Message, 38

message
 RcmClient_Packet, 39

msgId
 RcmClient_Packet, 39

msgqHeader
 RcmClient_Packet, 39

name
 RcmServer_FxnDesc, 42

osPriority
 RcmServer_Params, 44
 RcmServer_ThreadPoolDesc, 47

poolId
 RcmClient_Message, 38

priority
 RcmServer_Params, 45
 RcmServer_ThreadPoolDesc, 47

RcmClient, 7

RcmClient_acquireJobId
 ti_sdo_rcm_RcmClient, 17

RcmClient_addSymbol
 ti_sdo_rcm_RcmClient, 17

RcmClient_alloc
 ti_sdo_rcm_RcmClient, 18

RcmClient_CallbackFxn
 ti_sdo_rcm_RcmClient, 17

RcmClient_checkForError
 ti_sdo_rcm_RcmClient, 18

RcmClient_construct
 ti_sdo_rcm_RcmClient, 20

RcmClient_create
 ti_sdo_rcm_RcmClient, 20

RcmClient_DEFAULTPOOLID
 ti_sdo_rcm_RcmClient, 14

RcmClient_delete
 ti_sdo_rcm_RcmClient, 21

RcmClient_destruct
 ti_sdo_rcm_RcmClient, 21

RcmClient_DISCRETEJOBID
 ti_sdo_rcm_RcmClient, 14

RcmClient_E_EXECASYNCTOTENABLED
 ti_sdo_rcm_RcmClient, 15

RcmClient_E_EXECFAILED
 ti_sdo_rcm_RcmClient, 15

RcmClient_E_INVALIDFXNIDX
 ti_sdo_rcm_RcmClient, 15

RcmClient_E_INVALIDHEAPID
 ti_sdo_rcm_RcmClient, 15

RcmClient_E_IPCERROR
 ti_sdo_rcm_RcmClient, 15

RcmClient_E_JOBIDNOTFOUND

- ti_sdo_rcm_RcmClient, 15
- RcmClient_E_LOSTMSG
 - ti_sdo_rcm_RcmClient, 16
- RcmClient_E_MSGALLOCFAILED
 - ti_sdo_rcm_RcmClient, 16
- RcmClient_E_MSGFXNERROR
 - ti_sdo_rcm_RcmClient, 16
- RcmClient_E_MSGQCREATEFAILED
 - ti_sdo_rcm_RcmClient, 16
- RcmClient_E_MSGQOPENFAILED
 - ti_sdo_rcm_RcmClient, 16
- RcmClient_E_SERVERERROR
 - ti_sdo_rcm_RcmClient, 16
- RcmClient_E_SERVERNOTFOUND
 - ti_sdo_rcm_RcmClient, 17
- RcmClient_E_SYMBOLNOTFOUND
 - ti_sdo_rcm_RcmClient, 17
- RcmClient_exec
 - ti_sdo_rcm_RcmClient, 21
- RcmClient_execAsync
 - ti_sdo_rcm_RcmClient, 22
- RcmClient_execCmd
 - ti_sdo_rcm_RcmClient, 22
- RcmClient_execDpc
 - ti_sdo_rcm_RcmClient, 23
- RcmClient_execNoWait
 - ti_sdo_rcm_RcmClient, 23
- RcmClient_exit
 - ti_sdo_rcm_RcmClient, 24
- RcmClient_free
 - ti_sdo_rcm_RcmClient, 24
- RcmClient_getSymbolIndex
 - ti_sdo_rcm_RcmClient, 24
- RcmClient_init
 - ti_sdo_rcm_RcmClient, 25
- RcmClient_Message, 37
 - data, 38
 - dataSize, 38
 - jobId, 38
 - poolId, 38
- RcmClient_Packet, 38
 - desc, 39
 - message, 39
 - msgId, 39
 - msgqHeader, 39
- RcmClient_Params, 40
 - callbackNotification, 40
 - heapId, 40
- RcmClient_releaseJobId
 - ti_sdo_rcm_RcmClient, 25
- RcmClient_removeSymbol
 - ti_sdo_rcm_RcmClient, 25
- RcmClient_Struct, 41
- RcmClient_waitUntilDone
 - ti_sdo_rcm_RcmClient, 25
- RcmServer, 26
 - RcmServer_addSymbol
 - ti_sdo_rcm_RcmServer, 29
 - RcmServer_construct
 - ti_sdo_rcm_RcmServer, 30
 - RcmServer_create
 - ti_sdo_rcm_RcmServer, 30
 - RcmServer_delete
 - ti_sdo_rcm_RcmServer, 31
 - RcmServer_destruct
 - ti_sdo_rcm_RcmServer, 31
 - RcmServer_E_SYMBOLNOTFOUND
 - ti_sdo_rcm_RcmServer, 28
 - RcmServer_E_SYMBOLSTATIC
 - ti_sdo_rcm_RcmServer, 28
 - RcmServer_E_SYMBOLTABLEFULL
 - ti_sdo_rcm_RcmServer, 29
 - RcmServer_exit
 - ti_sdo_rcm_RcmServer, 31
 - RcmServer_FxnDesc, 42
 - addr, 42
 - name, 42
 - RcmServer_FxnDescAry, 42
 - RcmServer_init
 - ti_sdo_rcm_RcmServer, 31
 - RcmServer_MsgFxn
 - ti_sdo_rcm_RcmServer, 29
 - RcmServer_Params, 43
 - fxns, 44
 - osPriority, 44
 - priority, 45
 - workerPools, 45
 - RcmServer_removeSymbol
 - ti_sdo_rcm_RcmServer, 32
 - RcmServer_start
 - ti_sdo_rcm_RcmServer, 32
 - RcmServer_Struct, 45

- RcmServer_ThreadPoolDesc, 46
 - osPriority, 47
 - priority, 47
- RcmServer_ThreadPoolDescAry, 47
- RcmTypes, 33

- ti/sdo/rcm/RcmClient.h, 49
- ti/sdo/rcm/RcmServer.h, 53
- ti/sdo/rcm/RcmTypes.h, 56
- ti_sdo_rcm_RcmClient
 - RcmClient_acquireJobId, 17
 - RcmClient_addSymbol, 17
 - RcmClient_alloc, 18
 - RcmClient_CallbackFxn, 17
 - RcmClient_checkForError, 18
 - RcmClient_construct, 20
 - RcmClient_create, 20
 - RcmClient_DEFAULTPOOLID, 14
 - RcmClient_delete, 21
 - RcmClient_destruct, 21
 - RcmClient_DISCRETEJOBID, 14
 - RcmClient_E_EXECASYNCNOTENABLED, 15
 - RcmClient_E_EXECFAILED, 15
 - RcmClient_E_INVALIDFXNIDX, 15
 - RcmClient_E_INVALIDHEAPID, 15
 - RcmClient_E_IPCERROR, 15
 - RcmClient_E_JOBIDNOTFOUND, 15
 - RcmClient_E_LOSTMSG, 16
 - RcmClient_E_MSGALLOCFAILED, 16
 - RcmClient_E_MSGFXNERROR, 16
 - RcmClient_E_MSGQCREATEFAILED, 16
 - RcmClient_E_MSGQOPENFAILED, 16
 - RcmClient_E_SERVERERROR, 16
 - RcmClient_E_SERVERNOTFOUND, 17
 - RcmClient_E_SYMBOLNOTFOUND, 17
 - RcmClient_exec, 21
 - RcmClient_execAsync, 22
 - RcmClient_execCmd, 22
 - RcmClient_execDpc, 23
 - RcmClient_execNoWait, 23
- RcmClient_exit, 24
- RcmClient_free, 24
- RcmClient_getSymbolIndex, 24
- RcmClient_init, 25
- RcmClient_releaseJobId, 25
- RcmClient_removeSymbol, 25
- RcmClient_waitUntilDone, 25
- ti_sdo_rcm_RcmServer
 - RcmServer_addSymbol, 29
 - RcmServer_construct, 30
 - RcmServer_create, 30
 - RcmServer_delete, 31
 - RcmServer_destruct, 31
 - RcmServer_E_SYMBOLNOTFOUND, 28
 - RcmServer_E_SYMBOLSTATIC, 28
 - RcmServer_E_SYMBOLTABLEFULL, 29
 - RcmServer_exit, 31
 - RcmServer_init, 31
 - RcmServer_MsgFxn, 29
 - RcmServer_removeSymbol, 32
 - RcmServer_start, 32
- workerPools
 - RcmServer_Params, 45