Programmer's Guide to the SRV Facility

Subroutines Providing Network and Message Support for DICOM SOP Classes

Stephen M. Moore
Andy S. Gokhale

Mallinckrodt Institute of Radiology
Electronic Radiology Laboratory
510 South Kingshighway Boulevard
St. Louis, Missouri 63110
314/362-6965 (Voice)
314/362-6971 (FAX)

Version 2.10.0

August 3, 1998

This document describes a facility which is used to support the SOP classes defined in the DICOM V3 Standard. These routines understand the format of DICOM messages and the order in which messages are transmitted, but allow application programs to implement the body of a service class.

---

1 Introduction

Part 4 of the DICOM V3 Standard defines a number of SOP Classes. This part of the Standard defines data models and the actions and responses that are expected of Service Class Users (SCUs) and Service Class Providers (SCPs). Part 7 of the Standard defines the parameters that are required and optional for DICOM messages and the structures of these messages. The combination of the service class definitions in Part 4 and the message definitions is needed to implement DICOM SOP classes.

The DICOM Standard defines the terms SCU and SCP and avoids the use of the terms client and server. It is common to think of a client application as the program which initiates the network connection and a server application as the program which accepts the network connection. The DICOM Standard explicitly allows either the SCU or SCP to initiate an Association and send request messages This document describes some examples in terms of the Client/Server model, but does not imply a one to one relationship between a server and an SCP application.

The functions in the SRV facility provide support for DICOM SOP classes but do not fully implement the classes by themselves. Applications use these functions to complete the SOP class implementation. The general model for the functions is shown in Figure 1 below. An application uses a request message and provides a callback function. The SRV facility transmits the request message to the peer application and waits for one or more response messages. For each response, the requester's callback function is called with the response from the peer. The callback function returns values to the SRV facility which tell the facility to maintain the dialog or abort (cancel).

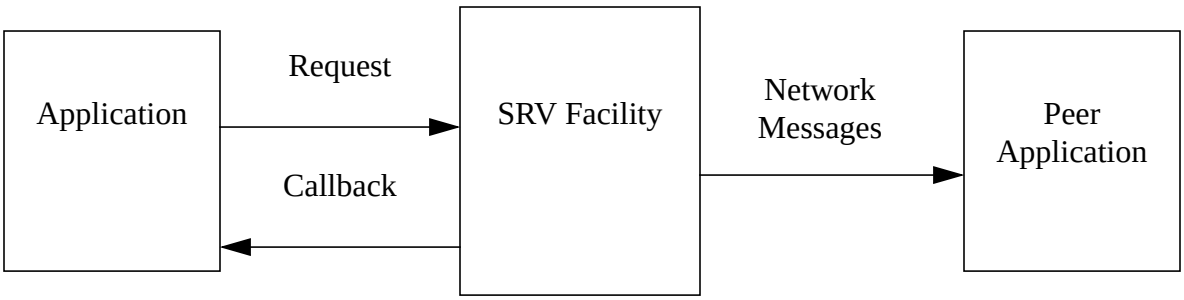

FIGURE 1. Application Sending Request Messages Using SRV Facility

The SRV facility provides a similar functionality for receiving request messages from a peer application. After a "server" process accepts an Association, it sits in a loop and waits for request messages from the peer. After the COMMAND group of a message is received, the server program is notified and calls an appropriate SRV response function. The SRV function will read any dataset which may complete the message and then call a server callback function with the entire message. The callback function processes the message and returns to the SRV facility. The SRV facility is responsible for sending the properly formatted response across the network and may call the callback function again if more work is needed to complete the operation.

Figure 2 shows the calling sequence for a C-FIND request that results in the SCP generating one match. The callback routine has sufficient context information to initiate a database operation. Each time the callback function is invoked, it finds a new match or determines there are no more entries that match the search criteria.

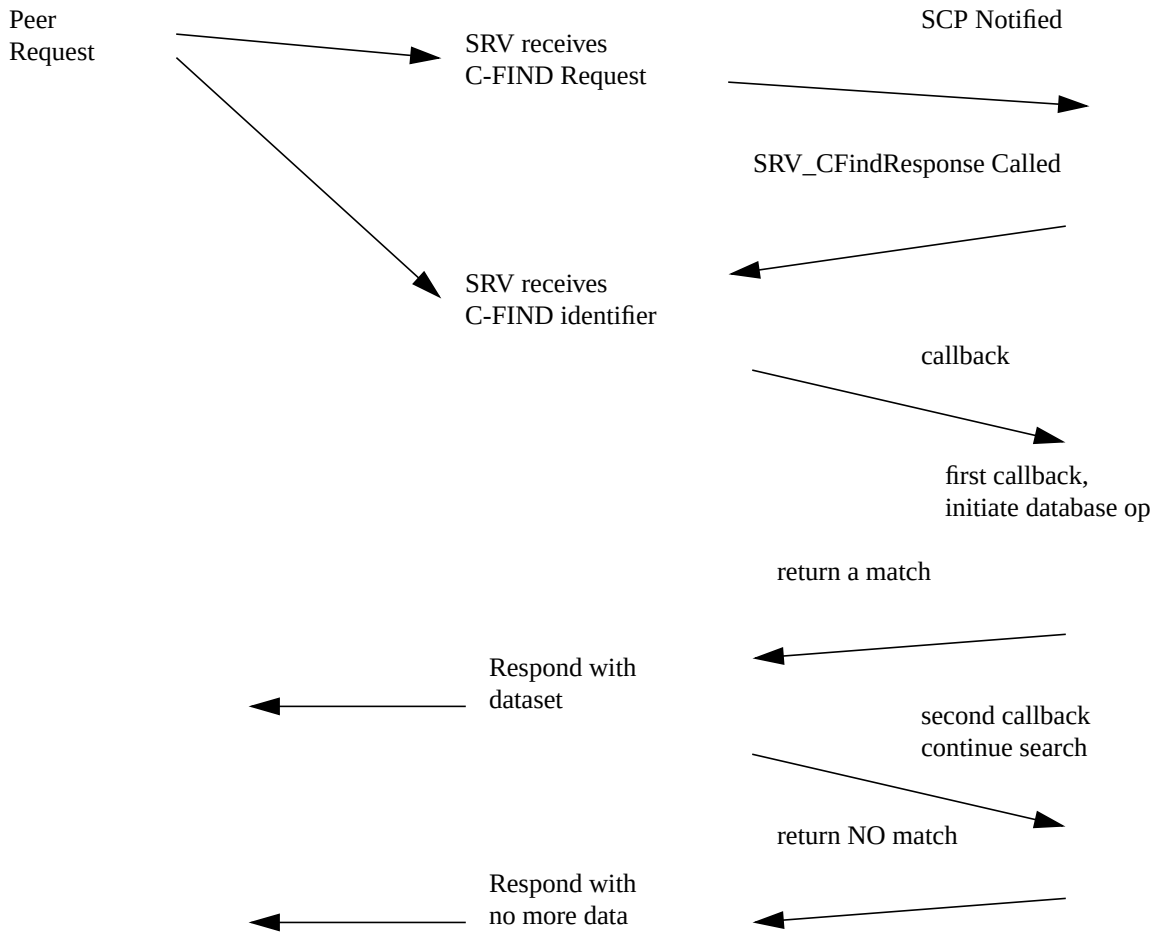

FIGURE 2. Timing Diagram of a Simple C-FIND Sequence

The SRV routines know the content of DICOM messages and use the MSG facility to build and parse COMMANDS. The SRV routines also use the DUL routines to communicate with peer applications. Users of this facility use SRV routines to determine which service classes are supported and then invoke DUL routines to actually request or accept Associations. Once an Association is established, the user communicates with the SRV routines until it is time to tear down the Association.

Applications and SRV routines cooperate when accepting messages from a peer application. As each message is received, a structure is allocated which contains the COMMAND portion of the message. The structure is defined by the MSG facility. The SRV functions will read the data set (if present) which completes the message and add that data to the MSG structure. These structures are released (freed) by SRV request and response routines when they are no longer needed. User applications should not attempt to free these structures.

Each SRV request function has an argument which is the address of a response structure which has been allocated (statically or dynamically) by the caller. If this address is not NULL, a copy of the final response message is placed at this address. Only the status part of the MSG structure is copied for the caller. No variable length items (like lists) are copied. This feature allows the caller to use this copy of the response message to print status information for debug purposes. All work should be completed in the callback functions. It is anticipated that future versions of the SRV functions will operate asynchronously and will invoke the user callback functions without returning this extra copy of the final response message.

2 Data Structures

The SRV facility makes use of the data structures defined by the MSG facility. There are no data structures which are defined explicitly for the SRV facility.

3 Include Files

Any applications that use these routines should include the following files in their code in the order given below.

```
#include “dicom.h”
#include “lst.h”
#include “dicom_objects.h”
#include “dulprotocol.h”
#include “dicom_messages.h”
#include “dicom_services.h”
```

4 Return Values

The following returns are possible from the SRV facility:

SRV_NORMAL	Normal return from SRV routine.
SRV_UNSUPPORTEDSERVICE	User requested a service class not supported by the SRV facility.
SRV_UNSUPPORTEDTRANSFERSYNTAX	None of the transfer syntaxes for the proposed presentation context are supported.

SRV_PEERREQUESTEDRELEASE	When reading the next command, the SRV function detected the peer application released the Association.
SRV_PEERABORTEDASSOCIATION	When reading the next command, the SRV function detected the peer aborted the Association.
SRV_READPDVFAILED	An error occurred while reading a PDV.
SRV_RECEIVEFAILED	SRV function failed to receive a PDV fragment.
SRV_UNEXPECTEDPRESENTATIONCONTEXTID	SRV routine encountered an unexpected presentation context ID when reading a PDU.
SRV_UNEXPECTEDPDVTYPE	SRV routine encountered an unexpected PDV type when reading a PDU.
SRV_SENDFAILED	
SRV_NOSERVICEINASSOCIATION	User function requested an SOP class that was not accepted during Association negotiation.
SRV_FILECREATEFAILED	SRV routine failed to create a file.
SRV_LISTFAILURE	SRV routine failed due to failure in LST facility.
SRV_MALLOCFAILURE	SRV function failed to allocate memory.
SRV_PRESENTATIONCONTEXTERROR	SRV function failed to create a new presentation context (using the DUL facility).
SRV_PARSEFAILED	SRV function failed to parse a PDU.
SRV_UNSUPPORTEDCOMMAND	SRV function read a COMMAND group with a command value that it did not understand.
SRV_NOTRANSFERSYNTAX	User requested a transfer syntax that is not supported.
SRV_NOCALLBACK	User invoked a request or response function without supplying a callback function.
SRV_ILLEGALPARAMETER	SRV function detected an illegal parameter in an argument or structure presented by the caller. Often indicates the type field in a MSG structure has not been initialized.
SRV_OBJECTBUILDFAILED	SRV routine failed to translate a structure into a DICOM Information Object.
SRV_REQUESTFAILED	An SRV request function failed.
SRV_RESPONSEFAILED	An SRV response function failed.

SRV_UNEXPECTEDCOMMAND	SRV function read a COMMAND group from the network with an unexpected command value.
SRV_CALLBACKABORTEDSERVICE	User callback aborted service by not returning SRV_NORMAL.
SRV_OBJECTACCESSFAILED	SRV function failed to extract an attribute from a DICOM Information Object.
SRV_QUERYLEVELATTRIBUTEMISSING	SRV function failed to find the Query Level attribute in a COMMAND.
SRV_ILLEGALQUERYLEVELATTRIBUTE	SRV function detected an illegal value in the Query Level attribute in a COMMAND.
SRV_PRESENTATIONCTXREJECTED	SRV facility rejected a proposed Presentation Context.
SRV_NETWORKTIMEOUT	A complete command or data set was not received within the timeout period.

5 SRV Routines

This section provides detailed documentation for each SRV facility routine.

SRV_AcceptServiceClass

Name

SRV_AcceptServiceClass - determine if the SRV facility can accept a proposed service class and build the appropriate response for the Association Accept message.

Synopsis

CONDITION SRV_AcceptServiceClass(DUL_PRESENTATIONCONTEXT *requestedCtx,
DUL_SC_ROLE role, DUL_ASSOCIATESERVICEPARAMETERS *params)

requestedCtx	The presentation context for the service which has been requested by the Requesting Application. This context includes the UID of the service class as well as proposed tran fer syntax UIDs.
role	Role proposed by the application for this service class.
params	The list of service parameters for the Association which is being negotiated. If the service class is accepted, a new presentation context will be added to the list of accepted services in this structure.

Description

SRV_AcceptServiceClass is called by an application which is accepting requests for Associations. This function should be called one time for each SOP Class that is proposed by the requesting application. SRV_AcceptServiceClass determines if the proposed SOP Class is supported by this facility and if at least one of the proposed transfer syntaxes are supported. If these conditions are met, a new DUL_PRESENTATIONCONTEXTITEM is allocated and added to the list of accepted presentation contexts in the caller's params structure.

If the facility does not support the SOP Class or any of the proposed transfer syntaxes, a new DUL_PRESENTATIONCONTEXTITEM is still allocated, but with a failed code placed in the result field. This item is added to the caller's list of accepted presentation contexts (but with the failed result) so the presentation context can be returned in the DUL accept PDU and notify the requestor why the SOP class was rejected.

Notes

The caller's list of accepted presentation contexts is used by other functions in this facility. When the caller wishes to send or receive messages, the SRV routines will examine this list to determine if the SOP class is supported.

This function only accepts Presentation Contexts which offer the DICOM explicit Little Endian transfer syntax.

Return Values

SRV_NORMAL	SRV_LISTFAILURE
SRV_PRESENTATIONCONTEXTERROR	SRV_UNSUPPORTEDTRANSFERSYNTAX
SRV_PRESENTATIONCTXREJECTED	SRV_UNSUPPORTEDSERVICE

SRV_CEchoRequest

Name

SRV_CEchoRequest - request a peer to provide the Verification Service Class by sending an ECHO request and waiting for an ECHO reply.

Synopsis

CONDITION SRV_CEchoRequest(DUL_ASSOCIATIONKEY **association,
DUL_ASSOCIATESERVICEPARAMETERS *params,
MSG_C_ECHO_REQ *echoRequest, MSG_C_ECHO_RESP *echoReply,
CONDITION (*callback)(), void *ctx, char *dirName)

association	Key which describes the Association used for transmitting the ECHO request and receiving the ECHO reply.
params	The Parameters which define the service classes that are available on this Association.
echoRequest	Pointer to structure where the user defines the parameters which are needed to create an ECHO request command (as defined in Part 7 of the DICOM standard).
echoResponse	Address of an MSG_C_ECHO_RESP structure allocated by the caller. This function will receive the echo response from the peer. If this address is not NULL, a copy of the response message is stored at that address.
callback	Address of user callback function to be called with ECHO Response from SCP.
ctx	Pointer to user context information which will be passed to the callback function. Caller uses this variable to contain any context required for callback function.
dirName	Name for directory where files may be created for large data sets.

Description

SRV_CEchoRequest assists an application that wants to be an SCU of the Verification SOP class. This function constructs a C-ECHO-REQ Message and sends it to the peer application which is acting as the SCP for the Verification class. This function waits for the response from the peer application and invokes the caller's callback function.

The arguments to the callback function are:

```
MSG_C_ECHO_REQ      *echoRequest
MSG_C_ECHO_RESP     *echoResponse
void                *ctx
```

The first two arguments are MSG structures that contain the C_ECHO Request and C_ECHO Response messages. The final argument is the caller's ctx variable that is passed to SRV_CEchoRequest.

The callback function should return SRV_NORMAL. Any other return value will cause the SRV facility to abort the Association.

Return Values

SRV_NORMAL	SRV_OBJECTBUILDFAILED
SRV_NOCALLBACK	SRV_REQUESTFAILED
SRV_UNSUPPORTEDSERVICE	SRV_CALLBACKABORTEDSERVICE
SRV_ILLEGALPARAMETER	

SRV_CEchoResponse

Name

SRV_CEchoResponse - provide the Verification Service class by sending an ECHO response to a peer.

Synopsis

```
CONDITION SRV_CEchoResponse(DUL_ASSOCIATIONKEY **association,
                DUL_PRESENTATIONCONTEXT *presentationCtx,
                MSG_ECHO_REQ **echoRequest, MSG_C_ECHO_RESP *echoReply,
                CONDITION (*callback)(), void *ctx, char *dirName)
```

association	Key which describes the Association used for transmitting the ECHO reply.
presentationCtx	Pointer to presentation context to be used when sending the ECHO response.
echoRequest	Address of a pointer to structure containing the parameters in the ECHO request which was received by the application.
echoReply	Pointer to structure in the user's area which will be filled in with the parameters of the ECHO response command by this function. After the parameters are filled in, the ECHO response is sent to the peer which requested the verification.
callback	Address of user callback function to be called with ECHO Response from SCP.
ctx	Pointer to user context information which will be passed to the callback function. Caller uses this variable to contain any context required for callback.
dirName	Name for directory where files may be created for large data sets.

Description

SRV_CEchoResponse assists an application that wants to be an SCP of the Verification SOP class. When an application receives an ECHO Response message, it calls this function with the ECHO request message and other parameters. *SRV_CEchoResponse* checks the caller's parameters and invokes the user's callback function. In the callback function, the caller fills in the parameters of the ECHO Response message and then returns to the SRV function. The arguments to the callback function are:

```
MSG_C_ECHO_REQUES          *echoRequest
MSG_C_ECHO_RESP            *echoResponse
void                       *ctx
DUL_PRESENTATIONCONTEXT *pc
```

The first two arguments are MSG structures that contain the C_ECHO Request and C_ECHO Response messages. The third argument is the caller's ctx variable that is passed to *SRV_CEchoResponse*. The pc argument gives the callback function a reference to the presentation context which describes this SOP class. The callback function should return SRV_NORMAL. Any other return value will cause the SRV facility to abort the Association.

Notes

The caller passes the address of a pointer to the MSG_C_ECHO_REQ message received by the application. *SRV_CEchoResponse* frees the echo request and writes NULL into the caller's pointer.

Return Values

```
SRV_NORMAL                    SRV_NOCALLBACK
SRV_ILLEGALPARAMETER          SRV_CALLBACKABORTEDSERVICE
SRV_OBJECTBUILDFAILED         SRV_RESPONSEFAILED
SRV_CALLBACKABORTEDSERVICE
```

SRV_CFindRequest

Name

SRV_CFindRequest - support the query service class as an SCU by handling network messages.

Synopsis

```
CONDITION SRV_CFindRequest(DUL_ASSOCIATIONKEY **association,
                DUL_ASSOCIATESERVICEPARAMETERS *params,
                MSG_C_FIND_REQ *findRequest, MSG_C_FIND_RESP *findResponse,
                CONDITION (*callback)(), void *ctx,  char *dirName)
```

association	The key for the association used to transmit the find request and receive the find response.
params	Parameters which defines the service classes that are available on this Association.
findRequest	Pointer to structure in caller's memory which contains the find request.
findResponse	Address of an MSG_C_FIND_RESP structure allocated by the caller. This function will receive the find response from the peer. If this address is not NULL, a copy of the response is stored at that address.
callback	Address of user routine which is called one time for each response received for the ne work.
ctx	User context information which is supplied during call to callback function.
dirName	Name for directory where files may be created for large data sets.

Description

SRV_CFindRequest assists an application that wants to be an SCU of the Query SOP class. This function constructs a C_FIND_REQ Message and sends it to the peer application which is acting as the SCP for the query class. This function waits for the responses from the peer application and invokes the user's callback function one time for each response.

The arguments to the callback function are:

```
MSG_C_FIND_REQ  *findRequest
MSG_C_FIND_RESP *findResponse
int             responseCount
char            *abstractSyntax
char            *queryLevelString
void            *callbackCtx
```

where

findRequest	Pointer to MSG structure with C_FIND request.
findResponse	Pointer to MSG structure with C_FIND response.
responseCount	Number of times callback function has been called for this query (starts at 1).
abstractSyntax	A character string which identifies the abstract syntax of the SOP Class of the query.
queryLevelString	A character string which identifies one of the four levels in the hierarchical query model.
callbackCtx	User's callbackCtx argument which is used to maintain context information in the cal back function.

Notes

The callback function should return SRV_NORMAL. Any other value will cause the SRV facility to discontinue the query.

Return Values

```
SRV_NORMAL
SRV_NOCALLBACK
SRV_ILLEGALPARAMETER
SRV_NOSERVICEINASSOCIATION
SRV_OBJECTACCESSFAILED
SRV_OBJECTBUILDFAILED
SRV_REQUESTFAILED
SRV_UNEXPECTEDCOMMAND
SRV_CALLBACKABORTEDSERVICE
```

SRV_CFindResponse

Name

SRV_CFindResponse - support the query service class as an SCP by handling network messages.

Synopsis

CONDITION SRV_CFindProvide(DUL_ASSOCIATIONKEY **association,
DUL_PRESENTATIONCONTEXT *ctx, MSG_C_FIND_REQ *findRequest,
MSG_C_FIND_RESP *findResponse, CONDITION (*callback)(),
void *callbackCtx, char *dirName)

association	The key for the Association on which the FIND request was received and will be used to transmit the FIND response.
ctx	Pointer to the presentation context for this FIND request.
findRequest	Address of a pointer to the structure which contains the FIND request received by the application.
findResponse	Pointer to structure in caller's space used to hold the FIND response message.
callback	Address of callback routine which is used to invoke database query and provide subs quent database retrievals.
callbackCtx	Pointer to any context information required by the caller's callback function.
dirName	Name for directory where files may be created for large data sets.

Description

SRV_CFindResponse is used by an application which is acting as an SCP of the query service. When an application receives a C-FIND Request message, it calls this function with the C-FIND request and other parameters. SRV_CFindResponse checks the caller's parameters and polls the network, waiting for an identifier which contains the query.

Once *SRV_CFindResponse* has read the identifier from the network, it creates an empty DCM_OBJECT in the identifier of the response message. The user's callback routine is invoked with the following parameters:

```
MSG_C_FIND_REQ  *findRequest
MSG_C_FIND_RESP *findResponse
int             responseCount
char            *abstractSyntax
char            *queryLevelString
void            *callbackCtx
```

where

findRequest	Pointer to MSG structure with C_FIND request.
findResponse	Pointer to MSG structure with C_FIND response.
responseCount	Number of times callback function has been called for this query (starts at 1).
abstractSyntax	A character string which identifies the abstract syntax of the SOP Class of the query.
queryLevelString	A character string which identifies one of the four levels in the hierarchical query model.
callbackCtx	User's callbackCtx argument which is used to maintain context information in the cal back function.

If the *responseCount* is 1, the callback function initiates a new database search. When the first response is received, the caller modifies the elements in the identifier in the response message and returns. *SRV_CFindResponse* takes the identifier, formats a C-FIND response message, and transmits the message to the requesting peer application. After the response is sent to the SCU application, SRV_CFindResponse invokes the callback function again.

If the *responseCount* is any value other than 1, the callback function continues the database search. For each match, the caller modifies the elements in the identifier in the response message and returns. The SRV function sends the proper message to the peer application for each response.

The user indicates the search is complete by placing the appropriate status value in the status field of the response message. The callback function should always return SRV_NORMAL. Any other value will cause *SRV_CFindRequest* to abort the Association.

Notes

The caller passes the address of a pointer to the MSG_C_FIND_REQ message received by the application. *SRV_CFindResponse* frees the echo request and writes NULL into the caller's pointer.

Return Values

```
SRV_NORMAL
SRV_NOCALLBACK
SRV_ILLEGALPARAMETER
SRV_RESPONSEFAILED
SRV_QUERYLEVELATTRIBUTEMISSING
SRV_ILLEGALQUERYLEVELATTRIBUTE
SRV_CALLBACKABORTEDSERVICE
```

SRV_CMoveRequest

Name

SRV_CMoveRequest - support the query/retrieve (MOVE) service as an SCU by providing network support.

Synopsis

```
CONDITION SRV_CMoveRequest(DUL_ASSOCIATIONKEY **association,
                DUL_ASSOCIATESERVICEPARAMETERS *params,
                MSG_C_MOVE_REQ *moveRequest, MSG_C_MOVE_RESP *moveResponse,
                CONDITION (*callback)(), void *ctx, char *dirName)
```

association	The key used to transmit the move request and to receive all move responses.
params	The structure which contains parameters which defines the association (and supported services).
moveRequest	Pointer to structure containing the request message to be transmitted to an SCP.
moveResponse	Address of an MSG_C_MOVE_RESP structure allocated by the caller. This function will receive the move response from the peer application. If this address is not NULL, a copy of the response message is stored at that address.
callback	Address of user function which is called for each move response received from an SCP.
ctx	User context information provided when caller's callback function is called.
dirName	Name for directory where files may be created for large data sets.

Description

SRV_CMoveRequest assists an application that wants to be an SCU of the Query/Retrieve SOP class (MOVE). This function constructs a C-MOVE-REQ Message and sends it to the peer application which is acting as the SCP for the Query/Retrieve SOP class. This function waits for the responses from the peer application and invokes the caller's callback function one time for each response. These responses are a number of "pending" responses followed by one "final" response.

The arguments to the *callback* function are:

```
MSG_C_MOVE_REQ  *moveRequest
MSG_C_MOVE_RESP *moveResponse
int             responseCount
char            *abstractSyntax
char            *queryLevelString
void            *callbackCtx
```

where

moveRequest	Pointer to MSG structure with C-MOVE request.
moveResponse	Pointer to MSG structure with C-MOVE response.
responseCount	Number of times callback function has been called for this query (starts at 1).
abstractSyntax	A character string which identifies the abstract syntax of the SOP Class of the query.
queryLevelString	A character string which identifies one of the four levels in the hierarchical query model.
callbackCtx	User's callbackCtx argument which is used to maintain context information in the cal back function.

On each invocation of the callback function, the user should examine the contents of the status field. This will indicate if the response message is a “pending” response or a “final” response.

The callback function should return SRV_NORMAL. Any other return value will cause the SRV facility to abort the Association.

Notes

Return Values

```
SRV_NORMAL
SRV_NOCALLBACK
SRV_ILLEGALPARAMETER
SRV_OBJECTACCESSFAILED
SRV_NOSERVICEINASSOCIATION
SRV_OBJECTBUILDFAILED
SRV_REQUESTFAILED
SRV_UNEXPECTEDCOMMAND
SRV_CALLBACKABORTEDSERVICE
```

SRV_CMoveResponse

Name

SRV_CMoveResponse - support the query/retrieve service (MOVE) as an SCP by providing network support.

Synopsis

```
CONDITION SRV_CMoveResponse(DUL_ASSOCIATIONKEY **association,
                DUL_PRESENTATIONCONTEXT *ctx,  MSG_C_MOVE_REQ **request,
                MSG_C_MOVE_RESP *response,
                CONDITION (*callback)(), void *callbackCtx, char *dirName)
```

association	The key for the association on which the find request was received and will be used to transmit the find response.
ctx	The presentation context on which the request was received.
request	Address of a pointer to the structure which contains the MOVE request received by the application.
response	Pointer to structure in caller's space used to hold the MOVE response message.
callback	Address of callback routine which is used to invoke database query and to store images to remote destination.
callbackCtx	Pointer to any context information required by the caller's callback function.
dirName	Name for directory where files may be created for large data sets.

Description

SRV_CMoveRequest assists an application that wants to be an SCP of the Query/Retrieve SOP class (MOVE). When an application receives a C-MOVE Request message, it calls this function with the C-MOVE request and other parameters. SRV_CMoveResponse checks the caller's parameters and polls the network, waiting for an identifer which contains the dataset identifying the images to be moved.

Once *SRV_CMoveResponse* has read the identifier from the network, it invokes the user's callback routine with the following parameters:

MSG_C_MOVE_REQ *moveRequest

```
MSG_C_MOVE_RESP *moveResponse
int             responseCount
char            *abstractSyntax
char            *queryLevelString
void            *callbackCtx
```

where

moveRequest	Pointer to MSG structure with C-MOVE request.
moveResponse	Pointer to MSG structure with C-MOVE response.
responseCount	Number of times callback function has been called for this query (starts at 1).
abstractSyntax	A character string which identifies the abstract syntax of the SOP Class of the query.
queryLevelString	A character string which identifies one of the four levels in the hierarchical query model.
callbackCtx	User's callbackCtx argument which is used to maintain context information in the cal back function.

If the *responseCount* is 1, the *callback* function should initiate a new database search to find the images that match the keys found in the move command. The *callback* function can then establish an association with the destination and transmit one or more images. Each time the *callback* function returns, *SRV_CMoveResponse* sends a status message to the application that invoked the move. If the final image has not be sent, *SRV_CMoveResponse* invokes the *callback* function again with a *responseCount* that has been incremented. This means it is up to the *callback* function to maintain context and to know which images have been transmitted.

The *callback* function indicates that there are more images to be transmitted by returning with a status value in the move response status (moveResponse->status) that is pending (ie. MSG_K_C_MOVE_SUBOPERATIONSCONTINUING). The *callback* function indicates that the last image has been transmitted by setting moveResponse->status to a final value (e.g., MSG_K_C_SUCCESS).

Notes

The *callback* function should return SRV_NORMAL. Any other return value will cause the SRV facility to abort the Association.

The caller passes the address of a pointer to the MSG_C_MOVE_REQ message received by the application. *SRV_CMoveResponse* frees the move request and writes NULL into the caller's pointer. The callback function can send one or more images during each invocation. *SRV_CMoveResponse* makes no assumptions about how many images are transmitted. If the *callback* function updates the count fields in the response message (remain completedSubOperations, failedSubOperations, warningSubOperations and sets the appropriate bits in the response message structure, *SRV_CFindResponse* will include these values in the pending responses that are sent to the peer that initiated the reqeust.

On each invocation, the callback function should examine the status value in *moveResponse*. A value of MSG_K_CANCEL means that *SRV_CMoveResponse* has detected a cancel request from the application that initiated the move. The callback function should stop sending images to the destination and perform any cleanup.

Return Values

```
SRV_NORMAL
SRV_NOCALLBACK
SRV_ILLEGALPARAMETER
SRV_RESPONSEFAILED
SRV_QUERYLEVELATTRIBUTEMISSING
SRV_ILLEGALQUERYLEVELATTRIBUTE
SRV_CALLBACKABORTEDSERVICE
```

SRV_CStoreRequest

Name

SRV_CStoreRequest - request a peer application to store an object by sending a store command and the object.

Synopsis

```
CONDITION SRV_StoreRequest(DUL_ASSOCIATIONKEY **association,
                DUL_ASSOCIATESERVICEPARAMETERS *params,
                MSG_C_STORE_REQ *storeRequest, MSG_C_STORE_RESP *storeResponse,
                CONDITION (*callback)(), void *callbackCtx, char *dirName)
```

association	Key which describes the Association used for transmitting the object.
params	he parameters which define the service classes which are supported on this Association.
storeRequest	Pointer to structure where the user defines the parameters which are needed to create a STORE command (as defined in Part 7 of the DICOM standard).
storeReponse	Address of an MSG_C_STORE_RESP structure allocated by the caller. This function will receive the store response from the peer. If this address is not NULL, a copy of the response message is stored at that address.
callback	User callback function which is called periodically while the object is transmitted to the peer application. This mechanism allows the caller to monitor progress and cancel the transmission.
callbackCtx	A pointer to user context information. This pointer is passed to the callback function as a parameter.
dirName	Name for directory where files may be created for large data sets.

Description

SRV_CStoreRequest assists an application that wants to be an SCU of one of the Storage SOP classes. This function constructs a C-STORE_REQ Message and sends it to the peer application which is acting as the SCP for the storage SOP class. After the request message is sent, *SRV_CStoreRequest* sends the data set which contains the object of the store request.

The user specifies the data set for the operation by placing a legal DICOM Information Object in the MSG_C_STORE_REQ structure or by including a file name in the structure that points to a DICOM Information Object.

The function calculates the number of bytes that are present in the data set and calls the user *callback* function during the send process. The callback function is called after each P-DATA PDU is sent over the network connection.

The arguments to the *callback* function are:

```
MSG_C_STORE_REQ   *storeRequest
MSG_C_STORE_RESP  *storeResponse
unsigned long     bytesTransmitted
unsigned long     totalBytes
void              *callbackCtx
```

where

storeRequest	Pointer to MSG structure with C_STORE request.
storeResponse	Pointer to MSG structure with C_STORE response.
bytesTransmitted	Number of bytes transmitted so far.
totalBytes	Total number of bytes in data set.
callbackCtx	User's callbackCtx argument in the callback function.

On each invocation of the *callback* function, the user should examine the *storeResponse* pointer. This pointer will be NULL during the store process. After *SRV_CStoreRequest* completes the process of sending the image, it waits for the C-STORE RESPONSE from the peer. When this process is received, the *callback* function is called a final time with the response message.

Notes

The *callback* function should return SRV_NORMAL. Any other return value will cause the SRV facility to abort the Association.

Return Values

```
SRV_NORMAL
SRV_NOCALLBACK
SRV_ILLEGALPARAMETER
SRV_NOSERVICEINASSOCIATION
SRV_OBJECTBUILDFAILED
SRV_REQUESTFAILED
SRV_UNEXPECTEDCOMMAND
SRV_CALLBACKABORTEDSERVICE
```

SRV_CStoreResponse

Name

SRV_CStoreResponse - support the Storage Service Class by accepting an object from the network and storing it in a disk file.

Synopsis

```
CONDITION SRV_CStoreResponse(DUL_ASSOCIATIONKEY **association,
                DUL_PRESENTATIONCONTEXT *ctx, MSG_C_STORE_REQ **storeRequest,
                MSG_C_STORE_RESP *storeReply, char *fileName,
                CONDITION (*callback)(), void *callbackCtx, char *dirName)
```

association	Key which describes the Association used for transmitting the command.
ctx	Presentation context to be used when receiving the object.
storeRequest	Address of a pointer to the MSG_C_STORE_REQ structure which was received from the peer application.
storeReply	Pointer to structure in caller area that will be filled by this function with the parameters used in the store reply which is sent to the peer application after the object is received.
fileName	Name of the file which should be used to store the object received from the network.
callback	User callback routine which is invoked during the storage process.
callbackCtx	Pointer to any context information required by the user's callback function.
dirName	Name for directory where files may be created for large data sets.

Description

SRV_CStoreResponse assists an application that wants to be an SCP of one of the storage SOP classes. When an application receives a C-STORE REQ Message, it calls this function with the request message and other parameters. This function opens the file name specified by the caller and receives the data set from the network.

SRV_CStoreResponse estimates the size of the incoming data set from the SOP Class in the Request message. Based on this estimate, *SRV_CStoreResponse* invokes the user *callback* function approximately ten times. (Since the size is only an estimate, the *callback* can be invoked more or less than ten times).

Once the entire data set is received, the *callback* function is invoked one final time. At this last *callback*, the Store Response structure will contain a DICOM Information Object which was created by opening the data set that was just received. The *callback* function should examine the Information Object. In this last callback, the *callback* function should set status values in the Response message.

After the final callback, this function creates a C_STORE Response message and sends it to the requesting application.

The arguments to the callback function are:

```
MSG_C_STORE_REQ           *storeRequest
MSG_C_STORE_RESP          *storeResponse
unsigned long             bytesReceived
unsigned long             totalBytes
DCM_Object                **object
void                      *callbackCtx
DUL_PRESENTATIONCONTEXT   *pc
```

where

storeRequest	Pointer to MSG structure with C_STORE request.
storeResponse	Pointer to MSG structure with C_STORE response.
bytesReceived	Number of bytes received so far.
totalBytes	Estimate of number of bytes in object based on SOP class.
object	Handle to the image received. Will be non-NULL after entire image received.
callbackCtx	User's callbackCtx argument.
pc	Reference to presentation context for this SOP Class.

Notes

The *callback* function should return SRV_NORMAL. Any other return value will cause the SRV facility to abort the Association.

The caller passes the address of a pointer to the MSG_C_STORE_REQ received by the application. *SRV_CStoreResponse* frees the store request and writes NULL into the caller's pointer.

Return Values

```
SRV_NORMAL
SRV_NOCALLBACK
SRV_FILECREATEFAILED
SRV_RESPONSEFAILED
SRV_UNEXPECTEDPDVTYPE
SRV_OBJECTBUILDFAILED
SRV_CALLBACKABORTEDSERVICE
```

SRV_Debug

Name

SRV_Debug - change the state of debugging information for the SRV facility

Synopsis

void SRV_Debug(BOOLEAN flag)

flag Flag which indicates if debug information should be enabled (TRUE) or (FALSE).

Description

SRV_Debug sets an internal flag in the SRV facility which is used to control output of debug messages. When enabled, each routine in the facility prints useful messages to standard output which can be used to trace the progress of SRV functions.

The caller should pass TRUE to enable debugging and FALSE to disable.

Notes

Return Values

None

SRV_MessageIDIn

Name

SRV_MessageIDIn - Function to reclaim ID messages after they have been used.

Synopsis

void SRV_MessageIDIn(unsigned short messageID)

messageID The message ID to be returned to the system.

Description

The SRV facility maintains a set of message IDs which are used in the COMMAND group of a DICOM message. *SRV_MessageIDIn* is called to return message IDs to the set after they have been used. This function in only called after all network references to *messageID* are complete.

Notes

Return Values

None

SRV_MessageIDOut

Name

SRV_MessageIDOut - Get a unique message ID which can be used in a DICOM command.

Synopsis

unsigned short SRV_MessageIDOut(void)

Description

The SRV facility maintains a set of message IDs which are used in the COMMAND group of a DICOM message. *SRV_MessageIDOut* is called to obtain the next unique ID from the set. This ID should be returned to the SRV facility via *SRV_MessageIDIn* after it is used .

Notes

Return Values

A unique message ID

SRV_NActionRequest

Name

SRV_NActionRequest - support the N-ACTION command as an SCU by providing network support.

Synopsis

CONDITION SRV_NActionRequest(DUL_ASSOCIATIONKEY **association,
DUL_ASSOCIATESERVICEPARAMETERS *params, char *SOPClass,
MSG_N_ACTION_REQ *actionRequest, MSG_N_ACTION_RESP *actionResponse,
CONDITION (*actionCallback)(), void *actionCtx, char *dirName)

association	The key used to transmit the N-ACTION request and to receive the N-ACTION response.
params	The structure which contains parameters which define the association (and supported services).
SOPClass	UID of the SOP class used when the association was negotiated. Because this can be a meta class, it may not be the same as the class UID in the N-ACTION request.
actionRequest	Pointer to structure containing the N-ACTION request to be transmitted to an SCP.
actionResponse	Address of an MSG_N_ACTION_RESP structure allocated by the caller. This functio will receive the action response from the peer. If this address is not NULL, a copy of the response message is stored at that address.
actionCallback	Address of user callback function to be called with the N-ACTION response from SCP.
actionCtx	Pointer to user context information which will be passed to the callback function. Caller uses this variable to contain any context required for the callback function.
dirName	Name for directory where files may be created for large data sets.

Description

SRV_NActionRequest assists an application that wants to be an SCU of a number of SOP classes. This function constructs an N-ACTION-REQ message and sends it to the peer application which is acting as an SCP for a SOP class. This function waits for the response from the peer application and invokes the caller's callback function.

The arguments to the callback function are:

```
MSG_N_ACTION_REQ  *actionRequest
MSG_N_ACTION_RESP *actionResponse
void              *ctx
```

The first two arguments are MSG structures that contain the N_ACTION Request and N-ACTION Response messages respectively. The final argument is the caller's context variable that is passed to *SRV_NActionRequest*.

The callback function should return SRV_NORMAL. Any other return value will cause the SRV facility to abort the Association.

Notes

The caller is responsible for explicitly setting the following fields in the N-ACTION request message:

```
type
messageID
classUID
dataSetType
instanceUID
actionTypeID
```

Return Values

```
SRV_NORMAL
SRV_NOCALLBACK
SRV_UNSUPPORTEDSERVICE
SRV_ILLEGALPARAMETER
SRV_OBJECTBUILDFAILED
SRV_UNEXPECTEDCOMMAND
SRV_CALLBACKABORTEDSERVICE
SRV_REQUESTFAILED
```

SRV_NActionResponse

Name

SRV_NActionResponse - support the N-ACTION command as an SCP by providing network support.

Synopsis

```
CONDITION SRV_NActionResponse(DUL_ASSOCIATIONKEY **association,
                DUL_PRESENTATIONCONTEXT *presentationCtx,
                MSG_N_ACTION_REQ **actionRequest, MSG_N_ACTION_RESP *actionResponse,
                CONDITION (*actionCallback)(), void *actionCtx, char *dirName)
```

association	The key used for the association which carried the N-ACTION request and will carry the N-ACTION response.
presentationCtx	Presentation context for this N-ACTION request.
actionRequest	Address of a pointer to the MSG_N_ACTION_REQ structure which was received from the peer application.
actionResponse	Pointer to structure in user's memory which will be used to create the N-ACTION response.
actionCallback	Address of user's callback function which is called to generate the N-ACTION response.
actionCtx	User context information passed to the user actionCallback function.
dirName	Name for directory where files may be created for large data sets.

Description

SRV_NActionResponse assists an application that wants to be an SCP of a number of SOP classes. When an application receives an N-ACTION request message, it calls this function with the N-ACTION request and other parameters. SRV_NActionResponse checks the caller's parameters and calls the user's callback function. In the callback function, the caller fills in the parameters of the N-ACTION response message and then returns to the SRV function.

After the callback function returns, SRV_NActionResponse constructs a N-ACTION Response message and sends it to the peer application which sent the request message.

The arguments to the callback function are:

```
MSG_N_ACTION_REQ          *actionRequest
MSG_N_ACTION_RESP         *actionResponse
void                      *ctx
DUL_PRESENTATIONCONTEXT   *pc
```

The first two arguments are MSG structures that contain the N_ACTION Request and N-ACTION Response messages respectively. The third argument is the caller's context variable that is passed to *SRV_NActionResponse*. The presentation context describes the SOP class that was negotiated for this message.

The callback function should return SRV_NORMAL. Any other return value will cause the SRV facility to abort the Association.

Notes

The callback function is responsible for explicitly setting the following fields in the N-ACTION response message:

```
type
messageIDRespondedTo
classUID
dataSetType
instanceUID
actionTypeID
```

The caller passes the address of a pointer to the MSG_N_ACTION_REQ received by the application. *SRV_NActionResp* frees the action request and writes NULL into the caller's pointer.

Return Values

```
SRV_NORMAL
SRV_NOCALLBACK
SRV_ILLEGALPARAMETER
SRV_OBJECTBUILDFAILED
SRV_CALLBACKABORTEDSERVICE
```

SRV_NCreateRequest

Name

SRV_NCreateRequest - support the N-CREATE command as an SCU by providing network support.

Synopsis

CONDITION SRV_NCreateRequest(DUL_ASSOCIATIONKEY ** association,
DUL_ASSOCIATESERVICEPARAMETERS *params, char *SOPClass,
MSG_N_CREATE_REQ *createRequest, MSG_N_CREATE_RESP *createResponse,
CONDITION (*createCallback)(), void *createCtx, char *dirName)

association	The key used to transmit the N-CREATE request and to receive the N-CREATE response.
params	The structure which contains the parameters which define the association (and supported classes).
SOPClass	UID of the SOP class used when the association was negotiated. Because this can be a meta class, it may not be the same as the class UID in the N-CREATE request.
createRequest	Pointer to structure containing the N-CREATE response. When the N-CREATE is received, memory for a structure will be allocated and the address of the structure will be passed back to the caller.
createResponse	Address of an MSG_N_CREATE_RESP structure allocated by the caller. This functio will receive the create response from the peer. If this address is not NULL, a copy of the response message is stored at that address.
createCallback	Address of user callback function to be called with the N-CREATE response from SCP.
createCtx	Pointer to user context information which will be passed to the callback function. Caller uses this variable to contain any context required for the callback function.
dirName	Name for directory where files may be created for large data sets.

Description

SRV_NCreateRequest assists an application that wants to be an SCU of a number of SOP classes. This function constructs an N-CREATE-REQ message and sends it to the peer application which is acting as an SCP for a SOP class. This function waits for the response from the peer application and invokes the caller's callback function.

The arguments to the callback function are:

```
MSG_N_CREATE_REQ  *createRequest
MSG_N_CREATE_RESP *createResponse
void              *ctx
```

The first two arguments are MSG structures that contain the N_Create Request and N-Create Response messages respectively. The final argument is the caller's context variable that is passed to *SRV_NCreateRequest*.

The callback function should return SRV_NORMAL. Any other return value will cause the SRV facility to abort the Association.

Notes

The caller is responsible for explicitly setting the following fields in the N-CREATE request message:

```
type
messageID
classUID
dataSetType
instanceUID
dataSet
```

Return Values

```
SRV_NORMAL
SRV_NOCALLBACK
SRV_UNSUPPORTEDSERVICE
SRV_ILLEGAPARAMETER
SRV_OBJECTBUILDFAILED
SRV_UNEXPECTEDCOMMAND
SRV_CALLBACKABORTEDSERVICE
SRV_REQUESTFAILED
```

SRV_NCreateResponse

Name

SRV_NCreateResponse - support the N-CREATE command as an SCP by providing network support.

Synopsis

```
CONDITION SRV_NCreateResponse(DUL_ASSOCIATIONKEY **association,
                DUL_PRESENTATIONCONTEXT *presentationCtx,
                MSG_N_CREATE_REQ **createRequest, MSG_N_CREATE_RESP *createResponse,
                CONDITION (*createCallback)(), void *createCtx, char *dirName)
```

association	The key used for the association which carried the N-CREATE request and will carry the N-CREATE response.
presentationCtx	Presentation context for this N-CREATE request.
createRequest	Address of a pointer to structure containing the N-CREATE request received from the peer application.
createResponse	Pointer to structure in user's memory that will be used to construct response.
createCallback	Address of user's callback function which is called to generate the N-CREATE response.
createCtx	User context information passed to user createCallback function.
dirName	Name for directory where files may be created for large data sets.

Description

SRV_NActionResponse assists an application that wants to be an SCP of a number of SOP classes. When an application receives an N-CREATE request message, it calls this function with the N-CREATE request and other parameters. *SRV_NCreateResponse* checks the caller's parameters and calls the user's callback function. In the callback function, the caller fills in the parameters of the N-CREATE response message and then returns to the SRV function. After the callback function returns, SRV_NCreateResponse constructs a N-CREATE Response message and sends it to the peer application which sent the request message.

The arguments to the callback function are:

```
MSG_N_CREATE_REQ  *createRequest
MSG_N_CREATE_RESP *createResponse
void               *ctx
```

The first two arguments are MSG structures that contain the N_CREATE Request and N-CREATE Response messages respectively. The third argument is the caller's context variable that is passed to *SRV_NCreateResponse*. The presentation context describes the SOP class that was negotiated.

The callback function should return SRV_NORMAL. Any other return value will cause the SRV facility to abort the Association.

Notes

The callback function is responsible for explicitly setting the following fields in the N-CREATE response message:

```
type
messageIDRespondedTo
classUID
dataSetType
instanceUID
dataSet
```

The caller passes the address of a pointer to the MSG_N_CREATE_REQ received by the application. *SRV_NCreateResponse* frees the action request and writes NULL into the caller's pointer.

Return Values

```
SRV_NORMAL
SRV_NOCALLBACK
SRV_RESPONSEFAILED
SRV_ILLEGALPARAMETER
SRV_OBJECTBUILDFAILED
SRV_CALLBACKABORTEDSERVICE
```

SRV_NDeleteRequest

Name

SRV_NDeleteRequest - Support the N-DELETE service as an SCU by providing network support.

Synopsis

CONDITION SRV_NDeleteRequest(DUL_ASSOCIATIONKEY **association,
DUL_ASSOCIATESERVICEPARAMETERS *params, char *SOPClass,
MSG_N_DELETE_REQ *deleteRequest, MSG_N_DELETE_RESP *deleteResponse, CONDITION
(*deleteCallback)(), void *deleteCtx, char *dirName)

association	The key used to transmit the N-DELETE request and to receive the N-DELETE response. params The structure which contains the parameters which define the association (and supported services).
SOPClass	UID of the SOP class used when the association was negotiated. Because this can be a meta class, it may not be the same as the class UID in the N-DELETE request.
deleteRequest	Pointer to structure containing the N-DELETE request to be transmitted to an SCP.
deleteResponse	Address of an MSG_N_DELETE_RESP structure allocated by the caller. This function will receive the delete response from the peer. If this address is not NULL, a copy of the response message is stored at that address.
deleteCallback	Address of user callback function to be called with the N-DELETE response from SCP.
deleteCtx	Pointer to user context information which will be passed to teh callback function. Caller uses this variable to contain any context required for the callback function.
dirName	Name for directory where files may be created for large data sets.

Description

SRV_NDeleteRequest assists an application that wants to be an SCU of a number of SOP classes. This function constructs an N-DELETE-REQ message and sends it to the peer application which is acting as an SCP for a SOP class. This function waits for the response from the peer application and invokes the caller's callback function.

The arguments to the callback function are:

```
MSG_N_DELETE_REQ*deleteRequest
MSG_N_DELETE_RESP *deleteResponse
void            *ctx
```

The first two arguments are MSG structures that contain the N-DELETE Request and N-DELETE Response messages respectively. The final argument is the caller's context variable that is passed to *SRV_NDeleteRequest*.

The callback function should return SRV_NORMAL. Any other return value will cause the SRV facility to abort the Association.

Notes

The caller is responsible for explicitly setting the following fields in the N-DELETE request message:

```
type
messageID
```

```
classUID
dataSetType
nstanceUID
```

Return Values

```
SRV_NORMAL
SRV_NOCALLBACK
SRV_UNSUPPORTEDSERVICE
SRV_ILLEGALPARAMETER
SRV_OBJECTBUILDFAILED
SRV_UNEXPECTEDCOMMAND
SRV_CALLBACKABORTEDSERVICE
SRV_REQUESTFAILED
```

SRV_NDeleteResponse

Name

SRV_NDeleteResponse - Support the N-DELETE service as an SCP by providing network support.

Synopsis

```
CONDITION SRV_NDeleteResponse(DUL_ASSOCIATIONKEY **association,
                DUL_PRESENTATIONCONTEXT *presentationCtx,
                MSG_N_DELETE_REQ **deleteRequest, MSG_N_DELETE_RESP *deleteResponse,
                CONDITION (*deleteCallback)(), void *deleteCtx, char *dirName)
```

association	The key used for the association which carried the N-DELETE request and will carry the N-DELETE response.
presentationCtx	Presentation context for this N-DELETE request.
deleteRequest	Address of a pointer to the MSG_N_DELETE_REQ structure which was received from the peer application.
deleteResponse	Pointer to the structure in user's memory which will be used to create the N-DELETE response.
deleteCallback	Address of user's callback function which is used to generate the N- DELETE response.
deleteCtx	User context information passed to user deleteCallback function.
dirName	Name for directory where files may be created for large data sets.

Description

SRV_NDeleteResponse assists an application that wants to be an SCP of a number of SOP classes. When an application receives an N-DELETE request message, it calls this function with the N-DELETE request and other parameters. SRV_NDeleteResponse checks the caller's parameters and calls the user's callback function. In the callback function, the caller fills in the parameters of the N-DELETE response message and then returns to the SRV function.

After the callback function returns, SRV_NDeleteResponse constructs a N-DELETE Response message and sends it to the peer application which sent the request message.

The arguments to the callback function are:

```
MSG_N_DELETE_REQ*deleteRequest
MSG_N_DELETE_RESP  *deleteResponse
void            *ctx
DUL_PRESENTATIONCONTEXT *pc
```

The first two arguments are MSG structures that contain the N-DELETE Request and N-DELETE Response messages respectively. The third argument is the caller's context variable that is passed to *SRV_NDeleteResponse*. The presentation context describes the SOP class.

The callback function should return SRV_NORMAL. Any other return value will cause the SRV facility to abort the Association.

Notes

The callback function is responsible for explicitly setting the following fields in the N-DELETE response message:

```
type
messageIDRespondedTo
classUID
dataSetType
instanceUID
```

The caller passes the address of a pointer to the MSG_N_DELETE_REQ received by the application. *SRV_NDeleteResponse* frees the action request and writes NULL into the caller's pointer.

Return Values

```
SRV_NORMAL
SRV_NOCALLBACK
SRV_RESPONSEFAILED
SRV_ILLEGALPARAMETER
SRV_OBJECTBUILDFAILED
SRV_CALLBACKABORTEDSERVICE
```

SRV_NEventReportRequest

Name

SRV_NEventReportRequest - support the N-EVENT-REPORT command as an SCP by providing network support.

Synopsis

CONDITION SRV_NEventReportRequest(DUL_ASSOCIATIONKEY **association,
DUL_ASSOCIATESERVICEPARAMETERS *params,
MSG_N_EVENT_REPORT_REQ *eventRequest,
MSG_N_EVENT_REPORT_RESP *eventResponse,
CONDITION (*eventCallback)(), void *eventCtx, char *dirName)

association	The key used to transmit the N-EVENT-REPORT request and to receive the N-EVENT-REPORT response.
params	The structure which contains parameters which define the association (and supported services).
eventRequest	Pointer to structure containing the N-EVENT-REPORT request to be transmitted to an SCU.
eventResponse	Address of an MSG_N_EVENT_REPORT_RESP structure allocated by the caller. This function will receive the event report response from the peer. If this address is not NULL, a copy of the response message is stored at that address.
eventCallback	Address of user callback function to be called with the N-EVENT_REPORT response from SCU.
eventCtx	Pointer to user context information which will be passed to the callback function. Caller uses this variable to contain any context required for the callback function.
dirName	Name for directory where files may be created for large data sets.

Description

SRV_NEventReportRequest assists an application that wants to be an SCP of a number of SOP classes. This function constructs an N-EVENT_REPORT request message and sends it to the peer application which is acting as SCU for a SOP class. This function waits for the response from the peer application and invokes the caller's callback function.

The arguments to the callback function are:

```
MSG_N_EVENT_REPORT_REQ    *eventRequest
MSG_N_EVENT_REPORT_RESP   *eventResponse
void                       *ctx
```

The first two arguments are MSG structures that contain the N-EVENT-REPORT Request and N-EVENT-REPORT Response messages respectively. The final argument is the caller's context variable that is passed to *SRV_NEventReportRequest*. The callback function should return SRV_NORMAL. Any other return value will cause the SRV facility to abort the Association.

Notes

The caller is responsible for explicitly setting the following fields in the N-EVENT-REPORT request message:

```
type
messageID
classUID
dataSetType
nstanceUID
eventTypeID
```

Return Values

```
SRV_NORMAL
SRV_NOCALLBACK
SRV_UNSUPPORTEDSERVICE
SRV_ILLEGALPARAMETER
SRV_OBJECTBUILDFAILED
SRV_UNEXPECTEDCOMMAND
SRV_CALLBACKABORTEDSERVICE
SRV_REQUESTFAILED
```

SRV_NEventReportResponse

Name

SRV_NEventReportResponse - support the N-EVENT-REPORT command as an SCU by providing network support.

Synopsis

```
CONDITION SRV_NEventReportResponse(DUL_ASSOCIATIONKEY **association,
                DUL_PRESENTATIONCONTEXT *presentationCtx,
                MSG_N_EVENT_REPORT_REQ **eventRequest,
                MSG_N_EVENT_REPORT_RESP *eventResponse,
                CONDITION (*eventCallback)(), void *eventCtx, char *dirName)
```

association	The key used for the association which carried the N-EVENT-REPORT request and will carry the N-EVENT-REPORT response. presentationCtx Presentation context for this N-EVENT-REPORT request.
eventRequest	Address of a pointer to the MSG_N_EVENT_REPORT_REQ structure which was received from the peer application.
eventResponse	Pointer to the structure in user's memory which will be used to create the N-EVENT-REPORT response.
eventCallback	Address of user callback function which is used to generate the N-EVENT-REPORT response.
eventCtx	User context information passed to user eventCallback function.
dirName	Name for directory where files may be created for large data sets.

Description

SRV_NEventReportResponse assists an application that wants to be an SCU of a number of SOP classes. When an application receives an N-EVENT-REPORT request message, it calls this function with the N-EVENT-REPORT request and other parameters. SRV_NEventReportResponse checks the caller's parameters and calls the user's callback function. In the callback function, the caller fills in the parameters of the N-EVENT-REPORT response message and then returns to the SRV function.

After the callback function returns, SRV_NEventReportResponse constructs a N-EVENT-REPORT Response message and sends it to the peer application which sent the request message.

The arguments to the callback function are:

```
MSG_N_EVENT_REPORT_REQ    *eventRequest
MSG_N_EVENT_REPORT_RESP   *eventResponse
void                      *ctx
DUL_PRESENTATIONCONTEXT    *pc
```

The first two arguments are MSG structures that contain the N-EVENT-REPORT Request and N-EVENT-REPORT Response messages respectively. The third argument is the caller's context variable that is passed to *SRV_NEventReportResponse*. The presentation context describes the SOP class.

The callback function should return SRV_NORMAL. Any other return value will cause the SRV facility to abort the Association.

Notes

The callback function is responsible for explicitly setting the following fields in the N-EVENT-REPORT request message.

```
type
messageIDRespondedTo
class UID
dataSetType
instanceUID
dataSet
```

The caller passes the address of a pointer to the MSG_N_EVENT_REPORT_REQ received by the application. *SRV_NEventReportResponse* frees the action request and writes NULL into the caller's pointer.

Return Values

```
SRV_NORMAL
SRV_NOCALLBACK
SRV_RESPONSEFAILED
SRV_ILLEGALPARAMETER
SRV_OBJECTBUILDFAILED
SRV_CALLBACKABORTEDSERVICE
```

SRV_NGetRequest

Name

SRV_NGetRequest - support the N-GET command as an SCU by providing network support.

Synopsis

CONDITION SRV_NGetRequest(DUL_ASSOCIATIONKEY **association,
DUL_ASSOCIATESERVICEPARAMETERS *params, char *SOPClass,
MSG_N_GET_REQ *getRequest, MSG_N_GET_RESP *getResponse,
CONDITION (*getCallback)(), void *getCtx, char *dirName)

association	The key used to transmit the N-GET request and to receive the N-GET response.
params	The structure which contains parameters which defines the association (and supported services).
SOPClass	UID of the SOP class used when the association was negotiated. Because this can be a meta class, it may not be the same as the class UID in the N-GET request.
getRequest	Pointer to structure containing the N-GET request to be transmitted to an SCP.
getResponse	Address of an MSG_N_GET_RESP structure allocated by the caller. This function will receive the get response from the peer. If this address is not NULL, a copy of the response message is stored at that address.
getCallback	Address of user callback function to be called with the N-GET response from SCP.
getCtx	Pointer to user context information which will be passed to the callback function. Caller uses this variable to contain any context required for the callback.
dirName	Name for directory where files may be created for large data sets.

Description

SRV_NGetRequest assists an application that wants to be an SCU of a number of SOP classes. This function constructs an N-GET Request message and sends it to the peer application which is acting as the SCP for a SOP class. This function waits for the response from the peer application and invokes the caller's callback function.

The arguments to the callback function are:

```
MSG_N_GET_REQ   *getRequest
MSG_N_GET_RESP  *getResponse
void            *ctx
```

The first two arguments are MSG structures that contain the N-GET Request and N-GET Response messages respectively.The final argument is the caller's context variable that is passed to SRV_NGetRequest.

The callback function should return SRV_NORMAL. Any other return value will cause the SRV facility to abort the Association.

Notes

The caller is responsible for explicitly setting the following fields in the N-GET request message:

```
type
messageID
classUID
ataSetType
attributeList
attributeCount
```

Return Values

```
SRV_NORMAL
SRV_NOCALLBACK
SRV_UNSUPPORTEDSERVICE
SRV_ILLEGALPARAMETER
SRV_OBJECTBUILDFAILED
SRV_UNEXPECTEDCOMMAND
SRV_CALLBACKABORTEDSERVICE
SRV_REQUESTFAILED
```

SRV_NGetResponse

Name

SRV_NGetResponse - support the N-GET command as an SCP by providing network support.

Synopsis

```
CONDITION SRV_NGetResponse(DUL_ASSOCIATIONKEY **association,
                DUL_PRESENTATIONCONTEXT *presentationCtx,
                MSG_N_GET_REQ **getRequest, MSG_N_GET_RESP *getResponse,
                CONDITION (*getCallback)(), void *getCtx, char *dirName)
```

association	The key used for the association which carried the N-GET request and will carry the N-GET response.
presentationCtx	Presentation context for this N-GET request.
getRequest	Address of a pointer to the MSG_N_GET_REQ structure which was received from the peer application.
getResponse	Pointer to structure in user's memory which will be used to create N-GET response
getCallback	Address of user callback function which is used to generate the Get response.
getCtx	User context information passed to user getCallback function.
dirName	Name for directory where files may be created for large data sets.

Description

SRV_NGetResponse assists an application that wants to be an SCP of a number of SOP classes. When an application receives an N-GET request message, it calls this function with the N-GET request and other parameters. SRV_NGetResponse checks the caller's parameters and calls the user's callback function. In the callback function, the caller fills in the parameters of the N-GET response message and then returns to the SRV function.

After the callback function returns, *SRV_NGetResponse* constructs a N-GET Response message and sends it to the peer application which sent the request message.

The arguments to the callback function are:

```
MSG_N_GET_REQ           *getRequest
MSG_N_GET_RESP          *getResponse
void                    *ctx
DUL_PRESENTATIONCONTEXT *pc
```

The first two arguments are MSG structures that contain the N-GET Request and N-GET Response messages respectively. The third argument is the caller's context variable that is passed to *SRV_NGetResponse*. The presentation context describes the SOP Class.

The callback function should return SRV_NORMAL. Any other return value will cause the SRV facility to abort the Association.

Notes

The callback function is responsible for explicitly setting the following fields in the N-GET response message.

```
type
messageIDRespondedTo
class UID
dataSetType
instanceUID
dataSet
```

The caller passes the address of a pointer to the MSG_N_GET_REQ received by the application. *SRV_NGetResponse* frees the action request and writes NULL into the caller's pointer.

Return Values

```
SRV_NORMAL
SRV_RESPONSEFAILED
SRV_ILLEGALPARAMETER
SRV_OBJECTBUILDFAILED
SRV_CALLBACKABORTEDSERVICE
```

SRV_NSetRequest

Name

SRV_NSetRequest - support the N-SET command as an SCU by providing network support.

Synopsis

```
CONDITION SRV_NSetRequest(DUL_ASSOCIATIONKEY **association,
        DUL_ASSOCIATESERVICEPARAMETERS *params, char *SOPClass,
        MSG_N_SET_REQ *setRequest, MSG_N_SET_RESP **setResponse,
        CONDITION (*setCallback)(), void *setCtx, char *dirName)
```

association	The key used to transmit the N-SET request and to receive the N-SET response.
params	The structure which contains parameters which defines the association (and supported services).
SOPClass	UID of the SOP class used when the association was negotiated. Because this can be a meta class, it may not be the same as the class UID in the N-SET request.
setRequest	Pointer to structure containing the N-SET request to be transmitted to an SCP.
setResponse	Address of an MSG_N_SET_RESP structure allocated by the caller. This function will receive the set response from the peer. If this address is not NULL, a copy of the response message is stored at that address.
setCallback	Address of user callback function to be called with the N-SET response from SCP.
setCtx	Pointer to user context information which will be passed to the callback function. Caller uses this variable to contain any context required for the callback function.
dirName	Name for directory where files may be created for large data sets.

Description

SRV_NSetRequest assists an application that wants to be an SCU of a number of SOP classes. This function constructs an N-SET-Request message and sends it to the peer application which is acting as an SCP for a SOP class. This function waits for the response from the peer application and invokes the caller's callback function.

The arguments to the callback function are:

```
MSG_N_SET_REQ   *setRequest
MSG_N_SET_RESP  *setResponse
void            *ctx
```

The first two arguments are MSG structures that contain the N-SET Request and N-SET Response messages respectively. The final argument is the caller's context variable that is passed to *SRV_NSetRequest*.

The callback function should return SRV_NORMAL. Any other return value will cause the SRV facility to abort the Association.

Notes

The caller is responsible for explicitly setting the following fields in the N-SET request message:

```
type
messageID
classUID
dataSetType
instanceUID
dataSet
```

Return Values

```
SRV_NORMAL
SRV_UNSUPPORTEDSERVICE
SRV_ILLEGALPARAMETER
SRV_OBJECTBUILDFAILED
SRV_UNEXPECTEDCOMMAND
SRV_CALLBACKABORTEDSERVICE
SRV_REQUESTFAILED
```

SRV_NSetResponse

Name

SRV_NSetResponse - Support the N-SET command as an SCP by providing network support.

Synopsis

CONDITION SRV_NSetProvide(DUL_ASSOCIATIONKEY **association,
DUL_PRESENTATIONCONTEXT *presentationCtx, MSG_N_SET_REQ **setRequest,
MSG_N_SET_RESP *setResponse, CONDITION (*setCallback)(), void *setCtx,
char *dirName)

association	The key used for the association which carried the N-SET request and will carry the N-SET response.
presentationCtx	Presentation context for this N-SET request.
setRequest	Address of a pointer to the MSG_N_SET_REQ structure which was received from the peer application.
setResponse	Pointer to structure in user's memory which will be used to create N-SET response.
setCallback	Address of user callback function which is used to generate a Set response.
setCtx	User context information passed to user setCallback function.
dirName	Name for directory where files may be created for large data sets.

Description

SRV_NSetResponse assists an application that wants to be an SCP of a number of SOP classes. When an application receives an N-SET request message, it calls this function with the N-SET request and other parameters. *SRV_NSetResponse* checks the caller's parameters and calls the user's callback function. In the callback function, the caller fills in the parameters of the N-SET response message and then returns to the SRV function.

After the callback function returns, *SRV_NSetResponse* constructs a N-SET Response message and sends it to the peer application which sent the request message.

The arguments to the callback function are:

```
MSG_N_SET_REQ   *SetRequest
MSG_N_SET_RESP  *SetResponse
void            *ctx
DUL_PRESENTATIONCONTEXT *pc
```

The first two arguments are MSG structures that contain the N-SET Request and N-SET Response messages respectively. The third argument is the caller's context variable that is passed to SRV_NSetResponse. The presentation context describes the SOP class.

The callback function should return SRV_NORMAL. Any other return value will cause the SRV facility to abort the Association.

Notes

The callback function is responsible for explicitly setting the following fields in the N-SET Response message.

```
type
messageIDRespondedTo
class UID
dataSetType
instanceUID
dataSet
```

The caller passes the address of a pointer to the MSG_N_SET_REQ received by the application. *SRV_NSetResponse* frees the action request and writes NULL into the caller's pointer.

Return Values

```
SRV_NORMAL
SRV_NOCALLBACK
SRV_RESPONSEFAILED
SRV_ILLEGALPARAMETER
SRV_OBJECTBUILDFAILED
SRV_CALLBACKABORTEDSERVICE
```

SRV_ReceiveCommand

Name

SRV_ReceiveCommand - check the status of the network and receive the next command on the network.

Synopsis

CONDITION SRV_ReceiveCommand(DUL_ASSOCIATIONKEY **association,
DUL_ASSOCIATESERVICEPARAMETERS *params, DUL_BLOCKOPTIONS block,J
int timeout, DUL_PRESENTATIONCONTEXTID *ctxID, unsigned short *command,
MSG_TYPE *messageType, void **messageArg)

association	The key which describes which association to use to check the network for a command. params The set of service parameters which describe the services (and associated presentation contexts) which are valid for this association.
block	A flag indicating if the caller wishes to block waiting for a command (DUL_BLOCK) or return immediately if there is no command present (DUL_NOBLOCK).
timeout	If the application chooses to block and wait for a command, the amount of time to wait before returning to the caller (in seconds).
ctxID	Pointer to a caller variable where this function will store the presentation context ID for the command received from the network.
command	Pointer to a caller variable where this function will store the command value from the COMMAND group which was read from the network.
messageType	Pointer to a caller variable where this function will store one of the enumerated message types from the MSG facility. There should be a one to one correspondence between the COMMAND received from the network and this message type.
messageArg	Address of a pointer in the caller's space. This function allocates a MSG structure and writes the address of the allocated structure in the caller's messageArg pointer.

Description

SRV_ReceiveCommand is used to poll an Association and read the next available COMMAND (as defined by data in the COMMAND group). The caller provides association information and the *block* and *timeout* parameters which are used to control the DUL routines. These parameters instruct the DUL routines to block while waiting for data from the network or to return after a timeout.

If a COMMAND is successfully read from the network, this function calls an MSG routine to parse the COMMAND and translate it into one of the fixed MSG structures. Memory for the structure is allocated and the address of the structure is returned to the caller via the *messageArg* parameter. This function also writes the Command Field value from the COMMAND into the variable pointed at by *command*.

Notes

If the function reads a COMMAND but is not able to parse it, the user should examine the value returned through command. All legal values in the Command Field should be supported.

Return Values

SRV_NORMAL	SRV_RECEIVEFAILED
SRV_UNSUPPORTEDCOMMAND	SRV_ILLEGALASSOCIATION
SRV_PEERREQUESTEDRELEASE	SRV_PEERABORTEDASSOCIATION
SRV_READPDVFAILED	SRV_NETWORKTIMEOUT

SRV_ReceiveDataSet

Name

SRV_ReceiveDataSet - poll an Association and read what is expected to be a data set.

Synopsis

CONDITION SRV_ReceiveDataSet(DUL_ASSOCIATIONKEY **association,
DUL_PRESENTATIONCONTEXT *presentationCtx, DUL_BLOCKOPTIONS block,
int timeout, char *dirName, DCM_OBJECT **dataSet)

association	The key used to receive the database from the network
presentationCtx	Presentation context on which we expect to receive the dataset.
block	Flag to be passed to network routines for blocking or non blocking I/O.
timeout	Timeout passed to network routines.
dirName	Name for directory where files may be created for (possibly large) data sets.
dataSet	Address of DICOM object variable which will be created when the dataset is received.

Description

SRV_ReceiveDataSet is used to poll an Association and read the next available data set. The caller provides association information and the block and timeout parameters which are used to control the DUL routines. These parameters instruct the DUL routines to block while waiting for data from the network or to return after a timeout.

If the data set is successfully read from the network, a DICOM Information Object is created and the handle stored at the address specified by dataSet.

Notes

dirName is an optional parameter. Some datasets are too large to store directly in memory. When the function determines that a large data set is being read, it will write it to a file. This file will be stored in the directory indicated by dirName. If dirName is empty (““) or NULL, the current working directory is assumed.

Return Values

```
SRV_NORMAL
SRV_RECEIVEFAILED
SRV_UNEXPECTEDPRESENTATIONCONTEXTID
SRV_UNEXPECTEDPDVTYPE
SRV_ILLEGALASSOCIATION
SRV_PEERREQUESTEDRELEASE
SRV_PEERABORTEDASSOCIATION
SRV_READPDVFAILED
SRV_NETWORKTIMEOUT
```

SRV_RejectServiceClass

Name

SRV_RejectServiceClass - reject an SOP class proposed by a calling application.

Synopsis

CONDITION SRV_RejectServiceClass(DUL_PRESENTATIONCONTEXT *requestedCtx,
unsigned short result, DUL_ASSOCIATESERVICEPARAMETERS *params)

requestedCtx	Pointer to requested Presentation Context which user is rejecting.
result	One of the defined DUL results which provide reasons for rejecting a Presentation Context.
params	The structure which contains parameters which defines the association (and supported services).

Description

SRV_RejectServiceClass is called by an application which is accepting requests for Associations. This function should be called one time for each SOP Class that is proposed by the requesting application that the accepting application wishes to reject.

This function allocates a DUL_PRESENTATIONCONTEXTITEM with a failed code placed in the result field. This item is added to the caller's list of accepted presentation contexts (but with the failed result) so the presentation context can be returned in the DUL accept PDU and notifiy the requestor why the SOP class was rejected.

Notes

Return Values

```
SRV_NORMAL
SRV_LISTFAILURE
SRV_PRESENTATIONCONTEXTERROR
```

SRV_RequestServiceClass

Name

SRV_RequestServiceClass - request a service class as an Association initiator and build a presentation context to be transmitted to an Association acceptor.

Synopsis

CONDITION SRV_RequestServiceClass(char *SOPClass, DUL_SC_ROLE role,
DUL_ASSOCIATESERVICEPARAMETERS *params)

SOPClass	The UID of the SOP Class that is being requested by the user.
role	The role the user is proposing for the service class for this Association. The user can propose to be an SCU, an SCP, either an SCU or an SCP, or the default.
params	Pointer to a user structure which contains the parameters which define the Association. If the SRV facility supports the requested service, a presentation context is created and added to the list of presentation contexts which will be proposed when the Association is requested.

Description

This function is called by an application which is proposing an Association and wishes to request a service class. The application can request the Service Class as an SCU or as an SCP or as both or as default. If the application requests the service class in one of these four roles, the Association negotiation includes information which proposes the Presentation Context according to the mode specified by the caller. If the caller uses the default mode, the SRV (and DUL) facility will assume the default SCU mode.

SRV_RequestServiceClass determines if the SRV facility supports the service class via a table lookup. If the class is supported, it builds a Presentation Context for the service and adds it to the list of Presentation Contexts for the Association which is to be requested.

Notes

This function creates a Presentation Context which uses the DICOM implicit Little Endian transfer syntax.

Return Values

```
SRV_NORMAL
SRV_LISTFAILURE
SRV_MALLOCFAILURE
SRV_PRESENTATIONCONTEXTERROR
SRV_UNSUPPORTEDSERVICE
```

SRV_SendCommand

Name

SRV_SendCommand - send a DICOM command to a peer using an established Association.

Synopsis

CONDITION SRV_SendCommand(DUL_ASSOCIATIONKEY **association,
DUL_PRESENTATIONCONTEXT *context, DCM_OBJECT **object)

association	Key which describes the Association used for transmitting the command.
context	Presentation context to be used when sending the command.
object	The DICOM object which contains the attributes of the command to be transmitted.

Description

SRV_SendCommand sends a DICOM command to a peer application using an established association. The user specifies the command to be sent by providing a DCM_OBJECT containing the proper attributes. *SRV_SendCommand* uses the object passed by the caller and constructs the proper PDVs and PDUs for transmission.

Notes

This function is normally called by higher level SRV functions (e.g., *SRV_CStoreRequest*).

Return Values

```
SRV_NORMAL
SRV_NOTRANSFERSYNTAX
SRV_SENDFAILED
```

SRV_SendDataSet

Name

CONDITION SRV_SendDataSet - send a DICOM data set to a peer using an established Association.

Synopsis

```
CONDITION SRV_SendDataSet( DUL_ASSOCIATIONKEY **association,
                DUL_PRESENTATIONCONTEXT *context, DCM_OBJECT **object,
                CONDITION (*callback)(), void *callbackCtx, unsigned long length)
```

association	Key which describes the Association used for transmitting the command.
context	Presentation context to be used when sending the dataset.
object	The DICOM object which contains the attributes of the dataset to be transmitted.
callback	User callback function which is called periodically while the data set is sent across the network. This allows the application to monitor the rate of transmission and cancel the transmission.
callbackCtx	User context information which is passed to the callback function as a parameter.
length	Length in bytes of the amount of data to transmit over the network before calling the user's callback function.

Description

SRV_SendDataSet transmits a single data set across the network using an existing Association. The user specifies the data set to be sent by providing a DCM_OBJECT containing the proper attributes. *SRV_SendDataSet* uses the object passed by the caller and constructs the proper PDVs and PDUs for transmission.

Notes

SRV_SendDataSet is normally called by higher level SRV functions (e.g., SRV_CStoreRequest).

Return Values

```
SRV_NORMAL
SRV_OBJECTACCESSFAILED
SRV_NOTRANSFERSYNTAX
SRV_SENDFAILED
```

SRV_TestForCancel

Name

SRV_TestForCancel - test the network for a cancel command and read it, if present

Synopsis

```
CONDITION SRV_TestForCancel(DUL_ASSOCIATIONKEY **association,
                DUL_BLOCKOPTIONS block, int timeout,
                DUL_PRESENTATIONCONTEXTID ctxID, unsigned short *command,
                MSG_TYPE *messageType, void **messageArg)
```

association	Key which describes the Association used for receiving the command.
block	Flag indicating if the operation should be performed in blocking mode.
timeout	Indicates length of time to wait if non-blocking.
messageType	Address of structure in user's memory to hold message type (if one is present).
messageArg	Address of a pointer in the user's memory. If a command is found on the network, SRV_TestForCancel will allocate a new MSG structure and place the address at messageArg.

Description

SRV_TestForCancel is used by functions that wish to test for cancel commands during operations (like a find or move). *SRV_TestForCancel* first inspects a queue of commands that are maintained by the SRV facility. If there is an entry on the queue, this command is removed and returned to the caller. Note that this may not be a cancel command. If that queue is empty, *SRV_TestForCancel* tries to read the next command from the network. If a command is received, it is parsed. If the command is a cancel command, that command will be returned to the caller. Other commands are placed on the queue.

Notes

The queueing mechanism and mode for returning commands that are not cancels make the behavior inconsistent. The user needs to be aware that the function may return a command that is not a cancel. This function is normally called by the SRV functions for handling move and find commands (and not the general public).

Return Values

```
SRV_NORMAL
SRV_RECEIVEFAILED
SRV_PARSEFAILED
SRV_UNSUPPORTEDCOMMAND
```