Digital Broadband Delivery System (DBDS) APIs

Please read this entire guide

Important

Please read this entire guide before you install or operate this equipment. Give particular attention to all safety statements.

Scientific Atlanta

Digital Broadband Delivery System (DBDS) APIs

Notices

Trademarks

Disclaimer

Continued on next page

Documentation Copyright Notice

Software Use Notice

Firmware Use Notice

Contents

Continued on next page

Continued on next page

Warranty and Disclaimer

Statement

Scientific-Atlanta, Inc. ("S-A") warrants that software licensed by it ("Licensed Software"), as provided, shall conform in all material respects to its published specifications current at the time the Licensed Software was shipped, downloaded or otherwise delivered to the Customer. During the first ninety (90) days after the date of delivery of Licensed Software, S-A shall use reasonable commercial efforts to correct errors detected in Licensed Software after receiving notification of such errors from Customer.

Disclaimer

S-A MAKES NO OTHER WARRANTIES, WHETHER EXPRESS OR IMPLIED, WITH RESPECT TO ANY LICENSED SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. S-A DOES NOT WARRANT THAT THE FUNCTIONS CONTAINED IN LICENSED SOFTWARE WILL MEET THE CUSTOMER'S REQUIREMENTS OR THAT THE OPERATION OF THE LICENSED SOFTWARE WILL BE UNINTERRUPTED OR ERROR-FREE. S-A MAKES NO WARRANTY OF NON-INFRINGEMENT, EXPRESS OR IMPLIED. ANY THIRD PARTY SOFTWARE SUPPLIED WITH OR INCORPORATED IN LICENSED SOFTWARE IS PROVIDED "AS IS," WITHOUT WARRANTIES OF ANY KIND. IF ANY ADDITIONAL WARRANTIES ARE SUPPLIED BY A THIRD PARTY, SUCH WARRANTIES WILL BE OFFERED DIRECTLY BY SUCH THIRD PARTY TO CUSTOMER.

Customer Responsibility

Customer acknowledges its responsibility to use all reasonable methods to prove out and thoroughly test the operation of and output from Licensed Software prior to its use in Customer's operations.

Unless otherwise provided in a separate writing, and subject only to the warranty of this Section, S-A is under no obligation to provide Customer with any modifications, updates, additions or revisions to Licensed Software, nor to maintain Licensed Software in any manner.

In the event that any modifications are made to Licensed Software which have not been authorized by S-A, any and all warranty and other obligations of S-A shall immediately cease with respect to such software.

Continued on next page

Year 2000 Statement

Licensed Software may require upgrades to accept and correctly process dates before, on and after January 1, 2000 ("Year 2000 Ready"). In addition, S-A cannot warrant that any Licensed Software, even though itself Year 2000 Ready, will operate correctly when used in conjunction with hardware, software, data, applications or services of third parties or products previously purchased from S-A. S-A cannot be responsible for the interoperability of the Licensed Software with hardware, software, data, applications or other services of third parties and advises the Customer to contact all other third party suppliers and to perform its own testing adequately in advance.

Limitation of Liability

EXCEPT FOR CLAIMS FOR PERSONAL INJURY CAUSED BY LICENSED SOFTWARE FURNISHED BY SCIENTIFIC-ATLANTA, SCIENTIFIC-ATLANTA SHALL NOT BE LIABLE TO THE CUSTOMER OR ANY OTHER PERSON OR ENTITY FOR INDIRECT, SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE, OR EXEMPLARY DAMAGES ARISING OUT OF OR IN CONNECTION WITH THE TRANSACTION IN WHICH THE LICENSED SOFTWARE WAS FURNISHED OR ANY ACTS OR OMISSIONS ASSOCIATED THEREWITH OR RELATING TO THE SALE OR USE OF ANY LICENSED SOFTWARE FURNISHED, WHETHER SUCH CLAIM IS BASED ON BREACH OF WARRANTY, CONTRACT, TORT OR OTHER LEGAL THEORY AND REGARDLESS OF THE CAUSES OF SUCH LOSS OR DAMAGES OR WHETHER ANY OTHER REMEDY PROVIDED HEREIN FAILS. IN NO EVENT SHALL S-A BE LIABLE TO CUSTOMER FOR ANY DAMAGES, HOWEVER BASED, IN EXCESS OF THE LESSER OF TEN THOUSAND UNITED STATES DOLLARS (U.S. $10,000) OR THE LICENSE FEE PAID BY CUSTOMER TO S-A FOR THE LICENSED SOFTWARE.

Chapter 1
Application Interface System (AIS) APIs

Overview

Introduction

This chapter contains technical information on Application Interface System component APIs.

In this Chapter

This chapter contains the following topics.

Application Server Manager

Script File

A script file exists for the addition of X/Motif buttons to the Application Server tab of the Admin Console GUI. By adding an entry to this file, an application can add its X/Motif GUI to the Admin Console. This is completely optional.

Billing

Billing Transactions

Billing related transactions, primarily applicable for Impulse Pay-Per-View type applications, are available as part of the BOSS API. See the *BOSS Phase 1.0 Interface Specification* for details.

Emergency Alert System (EAS)

Emergency Alert Message (EAM)

When an Emergency Alert Message (EAM) is received through an EAS decoder/encoder, an Emergency Alert Controller (EAC) collects the text and audio of the message. It then formats and sends one or two messages containing the EAM to the Generic Emergency Alert Receiver Server (GEARS). The GEARS processes the EAM and then broadcasts the alert over the network to DHCTs as a PassThru message, via the Messaging Server. The format of the message is described in *Draft Revision of SCTE DVS/168, January 27, 1999.* Resident Navigator clients can register for the DSM-CC PassThru message ID 0x9111 using the DSM-CC Messaging Client. Other application clients do not need to interface with EAS.

Navigator API

Navigator Server API

No server APIs are available at this time.

Navigator Client API

The Navigator API client gives access to a few Navigator user settings and PIN entry applet. It is part of the SARA APIs as described in the *Scientific-Atlanta Resident Application (SARA) Developer's Reference.*

Service Access Manager (SAM)

SAM Server

The SAM Server API is available through the following BOSS transactions:

- DefineService
- QueryService
- RetireService
- DefineDisplayChannelMap
- QueryDisplayChannelMap
- RetireDisplayChannelMap

See the *BOSS Phase 1.0 Interface Specification* for details. SAM Service and Channel Map functionality is also available through the SAM Service and Channel Maps buttons on the Application Interface Modules tab of the Admin Console GUI.

URL Modifiers

The following table lists modifiers that can be added to the application client URL when a service is defined. Modifiers are of the form:

"`bfs://myapp/myapp.ptf;`*`parameter`*`=`*`value`*"

Parameter	**Values**	**Details**
Stay Resident	TRUE (default) FALSE	If TRUE, application client stays resident in DHCT DRAM after its services are suspended.
SAM Events	TRUE (default) FALSE	If TRUE, application client will be sent SAM Activate, Suspend, and Is Authroized events. If FALSE, application should abide by ResumeReq and SuspendReq sent by PowerTV to indicate service activational suspension.

Continued on next page

Parameter	Values	Details
EID	Decimal value	If a PowerKEY package has been created for conditional access to application, specify the package EID. This is required for applications in a production system.
EAID	Decimal Value (default 1)	If a package has been created for conditional access, specify the Entitlement Agent to be used by the CAM client. Specify 1 for internal PowerKey secure micro. **Note:** Applications in general use the default value.

SAM Client

SAM events can be registered for and received by application clients as described in the *Scientific-Atlanta Resident Application* (*SARA) Application Developer's Guide.*

Broadcast File System (BFS)

BFS Server

The BFS Server API is an RPC based API that allows applications to create, write, and delete files to the broadcast carousels. The BFS RPC header files are distributed as part of the Network Development Kit (NDK) Samples CD.

The server API to the BFS performs four distinct functions.

Function	Description
Register the application server with the BFS server	This API enables an application server to register (and cancel registration) with the BFS server. **Notes:** • An application server must register with the BFS server to receive its directory (published as part of the BFS), and to create directories and files in the file system. • Registration is also necessary for the server to receive requests from a client when the system is operating in two-way mode.
Create directories and files in the BFS directory structure	This API allows an application server to register file systems with the BFS.
Deliver files to the BFS server	This API provides a simple mechanism for delivering files from an application server into the BFS network. This includes copying files to the BFS server or creating links from the BFS server to the application server's file system.
Two-way messaging interface	This API provides the interface for a client to request a file from a registered application server and for the application server to reply to the request.

In general, the RPC tool generates client stub functions (.c file) and a header file (.h file), which defines the necessary data types and exposes the public functions for the BFS Server interface. The application's server program uses the client stubs to access the BFS Server.

The following sections document the data types and client stub functions used for the BFS Server interface.

Continued on next page

Data Types

```
typedef unsigned long      BFSHandle;

typedef int          BFSLength;

typedef BFSHandle    BFSServerHandle;

typedef BFSHandle    BFSFileHandle;

typedef BFSHandle    BFSDirectoryHandle;

typedef BFSHandle    BFSLinkHandle;

typedef BFSHandle    BFSCursorHandle;

typedef unsigned char      BFSSourceId;

enum BFSOperationMode {
   BFSOperationModeOneWay = 0,
   BFSOperationModeTwoWay = 1
};
typedef enum BFSOperationMode BFSOperationMode;

enum BFSRegistrationType {
   BFSRegistrationTypeNormal = 0,
   BFSRegistrationTypeRestart = 1
};
typedef enum BFSRegistrationType BFSRegistrationType;

struct RegServ {
   char *BFSName;
   enum BFSOperationMode bfsOperationMode;
   enum BFSRegistrationType bfsRegistrationType;
   struct {
         u_int BFSSourceIdList_len;
         int *BFSSourceIdList_val;
   } BFSSourceIdList;
};
typedef struct RegServ RegServ;
```

Continued on next page

```
enum BFSError {
BFSInvalidServer = 0,
BFSInvalidName = 1,
BFSNoError = 2,
BFSInvalidOperationMode = 3,
BFSAlreadyRegistered = 4,
BFSInvalidPath = 5,
BFSInvalidHandle = 6,
BFSInvalidRequestHandle = 7,
BFSInvalidMode = 8,
BFSDuplicate = 9,
BFSExistingElement = 10,
BFSInvalidSourceId = 11,
BFSInternalError = 12,
BFSAlreadyOpen = 13,
BFSInvalidCursorHandle = 14,
BFSSystemServer = 15
};
typedef enum BFSError BFSError;

struct Servhandle {
   int         BFSServerHandle;
   BFSError    bfserror;
};
typedef struct Servhandle Servhandle;

enum BFSBoolean {
   BFSFalse = 0,
   BFSTrue = 0 + 1
};
typedef enum BFSBoolean BFSBoolean;

struct bfbool {
   enum BFSBoolean bfsboolean;
   enum BFSError bfserror;
};
typedef struct bfbool bfbool;


struct CreateDir {
   int BFSServerHandle;
   char *BFSPath;
};
typedef struct CreateDir CreateDir;
```

Continued on next page

```
struct dirhandle {
   int BFSDirectoryHandle;
   BFSError bfserror;
};
typedef struct dirhandle dirhandle;

struct ModDir {
   int BFSDirectoryHandle;
   char *BFSName;
};
typedef struct ModDir ModDir;

enum BFSMode {
   BFSModeLocalDelivery = 0,
   BFSModeBFSServerDelivery = 1
};
typedef enum BFSMode BFSMode;

struct CreateFile {
   int BFSHandle;
   char *BFSName;
   enum BFSMode bfsMode;
   int BFSSourceId;
};
typedef struct CreateFile CreateFile;

struct filehandle {
   int BFSFileHandle;
   BFSError bfserror;
};
typedef struct filehandle filehandle;

struct ModFile {
   int BFSFileHandle;
   char *BFSName;
};
typedef struct ModFile ModFile;

struct bfsCopy {
   int   BFSFileHandle;
   char  *BFSPath;
};
typedef struct bfsCopy bfsCopy;

struct CreateLnk {
   int   BFSHandle;
   char  *BFSPath;
   char  *BFSName;
   int   BFSSourceId;
};
typedef struct CreateLnk CreateLnk;
```

Continued on next page

```
struct linkhandle {
   int   BFSLinkHandle;
   BFSError bfserror;
};
typedef struct linkhandle linkhandle;

struct ModifyLnk {
   int   BFSLinkHandle;
   char  *BFSPath;
   char  *BFSName;
};
typedef struct ModifyLnk ModifyLnk;

enum BFSOpenMode {
   BFSOpenReadMode = 0,
   BFSOpenWriteMode = 1,
   BFSOpenModifyMode = 2,
   BFSOpenWriteTruncate = 3
};
typedef enum BFSOpenMode BFSOpenMode;

struct bfsOpen {
   int BFSFileHandle;
   enum BFSOpenMode bfsOpenMode;
};
typedef struct bfsOpen bfsOpen;

typedef struct {
   u_int Buffer_len;
   char *Buffer_val;
} Buffer;

struct bfsWrite {
   int BFSFileHandle;
   int BFSLength;
   Buffer Buffer;
};
typedef struct bfsWrite bfsWrite;

struct retServhandle {
   int   BFSServerHandle;
   enum  BFSOperationMode bfsOperationMode;
   enum  BFSError bfserror;
};
typedef struct retServhandle retServhandle;
```

Continued on next page

Generated Client Stub Functions

Note: All client stubs require a CLIENT type pointer that references the handle to the client returned from a clnt_create() call.

BFS Server Administrative Operations

An application server must register with the BFS server before it can participate as part of the BFS. When an application server registers with the BFS server, the BFS server creates a top-level directory entry for the server in the BFS directory structure.

Example: If application servers “GUIDE” and “WWW” registered with the BFS server, the BFS directory structure would contain two top level entries, /GUIDE and /WWW. These entries would then appear in the client’s view of the BFS. When an application server cancels registration with the BFS server, its corresponding entry is removed from the BFS directory.

Continued on next page

bfsqueryserver_1

Used to determine if BFS Server has been registered by an application.

Specification	Details
Syntax	`retServhandle *bfsqueryserver_1(Name *, CLIENT *);`
Parameters Name *	 Name of the server that appears in client's (DHCT) view of the BFS under the BFS directory. **Example:** If the server "GUIDE" had registered with the BFS, its directory entry would be **/GUIDE**.
Returns retServhandle * BFSServerHandle bfsOperationMode bfserror	 Pointer to retServhandle type structure containing the following elements. NULL pointer value or an error indication in the bfserror element (!= BFSNoError) indicates failure. Handle for BFS Server 'Name'. Indicates if the server will accept requests from the client for files that are not in currently in the BFS. Values possible for this field are: • BFSOperationModeOneWay • BFSOperationModeTwoWay enumerated error indication, see BFS Server Data Types.
Comments	None
Exceptions	None
Example	None
See Also	None

Continued on next page

bfsregisterserver_1

Registers an application server in the BFS directory hierarchy.

Specification	Details
Syntax	`Servhandle *bfsregisterserver_1(RegServ *, CLIENT *);`
Parameters	
RegServ *	pointer to a RegServ type structure which includes the following elements:
BFSName	Name of the server that will appear in the client's view of the BFS under the BFS directory. All directories and files created by this server will appear under this directory. **Example:** If the server "GUIDE" had registered with the BFS, its directory entry would be **/GUIDE**.
BfsOperationMode	Indicates if the server will accept requests from the client for files that are not in currently in the BFS. Values possible for this field are: • BFSOperationModeOneWay • BFSOperationModeTwoWay
bfsRegistrationType	Indicates if the server is requesting that any existing BFS operations for that server be terminated, essentially restarting that server. Values possible for this field are: • BFSRegistrationTypeNormal • BFSRegistrationTypeRestart **Note:** This operation is useful when a server does not know the state of the BFS server and wishes to completely rebuild the file system for that server. Alternately, the server may use the query commands to determine the state of the server on the BFS.

Continued on next page

Specification	Details
Parameters	
BFSSourceIDList	Array of source ID's defined in the DNCS. **Notes:** • In order for a server registration to be successful, the specified sourceId's must be defined in the DNCS.. • The sourceId is used by the BFS server to determine which transport stream will be used to deliver a particular file. In a distributed Data Carousel Server environment, the source Id is also used to identify which server will be used to deliver the file.
Returns	ServHandle type pointer which includes a BFS Server handle and an error code. See Data Types for details on the ServHandle type. A NULL pointer value or an error indication in the bfserror element (!= BFSNoError) indicates failure.
Comments	None
Exceptions	None
Example	None
See Also	None

Continued on next page

bfsderegisterserver_1

De-registers an application server from the BFS server. This cancels registration by instructing the BFS server to cease all operations for that application server and to remove the application server's directory from the BFS.

Specification	Details
Syntax	`bfbool *bfsderegisterserver_1(BFSServerHandle *, CLIENT *);`
Parameters BFSServerHandle *	Pointer to BFS Server handle returned from bfsregisterserver_1().
Returns	Pointer to bfbool type structure containing a pass/fail indication and a return error code. NULL pointer or BFSFalse value in the bfsboolean element indicates failure.
Comments	None
Exceptions	None
Example	None
See Also	None

Continued on next page

Directory and File Administration

After an application server registers with the BFS server, it can publish directory and file information. This section contains the API command set necessary to create directories and files in the BFS directory.

bfscreatedirectory_1

Creates a directory for the indicated server.

Specification	Details
Syntax	`dirhandle *bfscreatedirectory_1(CreateDir *, CLIENT *);`
Parameters CreateDir *	 pointer to CreateDir type structure, which includes the following elements:
BFSServerHandle	Indicates the element to which the directory will be connected. This field may be either: BFSServerHandle BFSDirectoryHandle
BFSPath	Indicates the location in the BFS hierarchy in which the file system will appear beyond the entry for the server. Examples: If application server "GUIDE" registers with the BFS server, its directory entry would be /GUIDE. Creating a directory on the BFS server handle with a path of "Plus24" would result in the directory entry /GUIDE/Plus24. Creating a directory on the directory handle that is returned from the above call with a path of "Listings/10AM" would result in the directory entry /GUIDE/Plus24/Listings/10AM.
Returns	Pointer to dirhandle type structure which contains new directory handle and a return error code. NULL pointer or BFSFalse value in the bfsboolean element indicates failure.
Comments	None
Exceptions	None
Example	None
See Also	None

Continued on next page

bfsmodifydirectory_1

Changes the location in the BFS directory structure of a particular directory handle. This is useful for "pruning and grafting" entire directory trees in the BFS.

Note: The hierarchy of elements below the modified directory remains intact.

Specification	Details
Syntax	`bfbool *bfsmodifydirectory_1(ModDir *, CLIENT *);`
Parameters ModDir *	 Pointer to ModDir type structure, which includes the following elements:
BFSDirectoryHandle	Indicates the directory whose name is being changed.
BFSPath	Indicates the new location and name for the handle in the BFS directory structure. **Example:** • If the server "GUIDE" has already created the directory "Plus24/Listings/10AM," the BFS directory structure would contain the directory entry: /GUIDE/Plus24/Listings/10AM. • The server may modify where the directory appears in the BFS directory structure by querying for the BFSHandle using the BFSPath "Plus24." Using the handle returned by this query, and a BFSPath of "Current," the directory will now appear in the BFS directory structure as: /GUIDE/Current/Listings/10AM.
Returns	Pointer to bfbool type structure containing a pass/fail indication and a return error code. NULL pointer or BFSFalse value in the bfsboolean element indicates failure.
Comments	None
Exceptions	None
Example	None
See Also	None

Continued on next page

bfsdeletedirectory_1

Deletes a directory that appears in the Broadcast Files System.

Note: Deleting a directory also deletes all files and directories that are attached under that directory.

Specification	Details
Syntax	`bfbool * bfsdeletedirectory_1(BFSDirectoryHandle *, CLIENT *);`
Parameters BFSDirectoryHandle *	 Indicates the directory whose name is being deleted.
Returns	Pointer to bfbool type structure containing a pass/fail indication and a return error code. NULL pointer or BFSFalse value in the bfsboolean element indicates failure.
Comments	None
Exceptions	None
Example	None
See Also	None

Continued on next page

bfscreatefile_1

Creates a file entry in the indicated directory path.

Specification	Details
Syntax	`filehandle *bfscreatefile_1(CreateFile *, CLIENT *);`
Parameters	
CreateFile *	pointer to CreateFile type structure with the following elements:
BFSHandle	The BFSHandle field may be either: • BFSServerHandle • BFSDirectoryHandle **Notes:** • If the handle is a server handle, the file entry will be created under the root directory for the server. • If the handle is a directory handle, the file entry will be created under the indicated directory path.
BFSName	Indicates the name that will appear in the BFS for that file. **Example:** • If server "WWW" had registered with the BFS server, its directory entry would be: /WWW. Creating a file on this server handle with a name of "toc.html" would result in a directory entry of: /WWW/toc.html.
BFSMode	Indicates how the file will be sent. This field must be set to BFSModeBFSServerDelivery.
BFSSourceId	Identifies the transport stream over which the file will be delivered. **Notes:** • This sourceId must have been one that was identified when the application server registers with the BFS server. • In a distributed Data Carousel server environment, the sourceId is also used to identify the Data Carousel server that will be used to deliver the file.

Continued on next page

Specification	Details
Returns	Pointer to filehandle type structure which contains new file handle and a return error code. NULL pointer or BFSFalse value in the bfsboolean element indicates failure.
Comments	None
Exceptions	None
Example	None
See Also	None

Continued on next page

bfsmodifyfile_1

Changes the name of a file that appears in the BFS.

Specification	Details
Syntax	`bfbool *bfsmodifyfile_1(ModFile *, CLIENT *);`
Parameters	
ModFile *	Pointer to ModDir type structure, which includes the following elements:
BFSFileHandle	Indicates the file whose name is being changed.
BFSName	Indicates the new name for the file that will appear in the BFS directory structure.
Returns	Pointer to bfbool type structure containing a pass / fail indication and a return error code. NULL pointer or BFSFalse value in the bfsboolean element indicates failure.
Comments	None
Exceptions	None
Example	None
See Also	None

Continued on next page

bfsdeletefile_1

Deletes a file that appears in the BFS. Deleting a file will cause the BFS server to remove that file from the BFS directory structure and stop sending the file.

Specification	Details
Syntax	`bfbool *bfsdeletefile_1(BFSFileHandle *, CLIENT *);`
Parameters	
BFSFileHandle *	Indicates the file whose name is being deleted.
Returns	Pointer to bfbool type structure containing a pass/fail indication and a return error code. NULL pointer or BFSFalse value in the bfsboolean element indicates failure.
Comments	None
Exceptions	None
Example	None
See Also	None

Continued on next page

bfscopy_1

Copies a file from the application server's file system into the BFS server's file system.

Note: This will overwrite any existing file on the BFS server.

Specification	Details
Syntax	`bfbool *bfscopy_1(bfsCopy *, CLIENT *);`
Parameters	
bfsCopy *	Pointer to bfsCopy type structure, which includes the following elements:
BFSFileHandle	This is the handle of a file in the BFS that has been created using the BFSCreateFile command.
BFSPath	Indicates the location in the application server's file system of the point that will be copied into the BFS directory structure. **Note:** If this file system is on another physical file system, that file system must be mounted on the file system that is being used by the BFS.
Returns	Pointer to bfbool type structure containing a pass/fail indication and a return error code. NULL pointer or BFSFalse value in the bfsboolean element indicates failure.
Comments	None
Exceptions	None
Example	None
See Also	None

Continued on next page

bfscreatelink_1

Creates a link from the BFS server to an application server's file system. The link allows the application to maintain the structure and content of the linked portion of the file system. The file system will be mirrored in the BFS. The BFS does not maintain a copy of the linked file structure. If the linked file structure is unavailable to the BFS server, then that portion of the file system will not be delivered over the BFS.

Specification	Details
Syntax	`linkhandle *bfscreatelink_1(CreateLnk *, CLIENT *);`
Parameters	
CreateLnk *	pointer to CreateLnk type structure, which includes the following elements:
BFSHandle	The handle below which the link will appear. This Handle field may be either: • BFSServerHandle • BFSDirectoryHandle
BFSPath	Indicates the location in the servers file system of the point that will be linked under the BFSHandle. **Note:** If this file system is on another physical file system, that file system must be mounted on the file system that is being used by the BFS.
BFSName	Indicates the name that will be given to the link point in the BFS. **Note:** This name may be different from the actual name in the application servers file system.
BFSSourceId	Identifies the transport stream over which all files that appear below the linked point will be delivered. **Note:** This sourceId must have been one that was identified when the application server registers with the BFS server.

Continued on next page

Specification	**Details**
Returns	Pointer to linkhandle type structure that contains the new link handle and a return error code. NULL pointer indicates failure. **Note:** If the link already exists in the BFS, this is not an error. The existing BFSLinkHandle will be returned and the BFSErrorCode field will be set to BFSExistingElement.
Comments	None
Exceptions	None
Example	None
See Also	None

Continued on next page

bfsmodifylink_1

Changes the name or path of a link that exists in the BFS.

Specification	Details
Syntax	`bfbool *bfsmodifylink_1(ModifyLnk *, CLIENT *);`
Parameters	
ModifyLnk *	pointer to ModifyLnk type structure, which includes the following elements:
BFSLinkHandle	Indicates the link whose name is being changed.
BFSPath	Indicates the new path to be linked at this point.
BFSName	Indicates the new name for the link.
Returns	Pointer to bfbool type structure containing a pass/fail indication and a return error code. NULL pointer or BFSFalse value in the bfsboolean element indicates failure.
Comments	None
Exceptions	None
Example	None
See Also	None

Continued on next page

bfsdeletelink_1

Deletes a link that exists in the BFS. Deleting a link causes the BFS server to stop sending all directories and files below the link point, and to remove the directories, files, and the link point from the BFS directory.

Specification	**Details**
Syntax	`bfbool * bfsdeletelink_1(BFSLinkHandle *, CLIENT *);`
Parameters BFSLinkHandle *	 Indicates the link that is being deleted.
Returns	Pointer to bfbool type structure containing a pass/fail indication and a return error code. NULL pointer or BFSFalse value in the bfsboolean element indicates failure.
Comments	None
Exceptions	None
Example	None
See Also	None

Continued on next page

BFS Files Access and Manipulation

The BFSCreateFile command returns a BFSFileHandle that is used to access the data delivered over the BFS. This section contains the BFS API set that performs BFS file access operations.

bfsopen_1

Opens a file for reading or writing. A file must be open before reading or writing can take place.

Specification	Details
Syntax	`bfbool *bfsopen_1(bfsOpen *, CLIENT *);`
Parameters bfsOpen * BFSFileHandle BFSOpenMode	 pointer to bfsOpen type structure with the following elements: Indicates the file that is being opened. Indicates the operation that will be performed on the file. This field may have one of the following values: • BFSOpenModeRead • BFSOpenModeWrite
Returns	Pointer to bfbool type structure containing a pass/fail indication and a return error code. NULL pointer or BFSFalse value in the bfsboolean element indicates failure.
Comments	None
Exceptions	None
Example	None
See Also	None

Continued on next page

bfswrite_1

Writes a file to the BFS. A file must be opened with the correct mode before writing can take place.

Note: The file will be copied into the BFS server's file system and delivered on the Data Carousel, as defined by the BFSMode options in the BFSCreateFile command.

Specification	Details
Syntax	`bfbool *bfswrite_1(bfsWrite *, CLIENT *);`
Parameters	
bfsWrite *	Pointer to bfsWrite type structure with the following elements:
BFSFileHandle	Indicates the file that is being written.
BFSLength	Indicates the size of the BFSBuffer.
Buffer	Indicates the buffer into which the file will be placed. This buffer must be at least BFSLength in size.
Returns	Pointer to bfbool type structure containing a pass/fail indication and a return error code. NULL pointer or BFSFalse value in the bfsboolean element indicates failure.
Comments	None
Exceptions	None
Example	None
See Also	None

Continued on next page

bfsclose_1

Closes a file. If the file was created for BFS server delivery, the file must be closed before the BFS server will begin sending the file.

Specification	Details
Syntax	`bfbool *bfsclose_1(BFSFileHandle *, CLIENT *);`
Parameters BFSFileHandle *	 Indicates the file that is being closed.
Returns	Pointer to bfbool type structure containing a pass/fail indication and a return error code. NULL pointer or BFSFalse value in the bfsboolean element indicates failure.
Comments	None
Exceptions	None
Example	None
See Also	None

BFS Client

The BFS Client API is available via the PowerTV Stream Manager, using the "bfs" protocol; see the *PowerTV OS 1.5 API* documentation.

DSM-CC Messaging Manager

Messaging Server

A PassThru Messaging library that allows access to the Messaging Server is distributed with the S-A DBDS Network Development Kit in source code form. It is not necessary to use this API library to send PassThru messages; a developer can directly follow DSM-CC specification for U-N PassThru messaging and send messages to the Messaging Server. However, the library provides convenience functions for making DSM-CC headers, etc. It also hides the fact that the FetchDhct BOSS transaction is not implemented currently, so applications have no way of knowing the NSAP address of a DHCT. The library "knows" how the DNCS makes an NSAP from a DHCT MAC address.

The following sections document the API.

Continued on next page

ptapi_createConnection

A routine to establish a TCP connection to the Messaging Server PassThru port and returns a handle that is used in subsequent calls.

The following table summarizes the general transaction format of the Messaging Server API routine ptapi_createConnection.

Specification	Details
Syntax	`int ptapi_createConnection(const char *dncsHost, PTAPI_Handle *retHandle);`
Parameters → dncsHost ← retHandle	 An IP address string of form aa.bb.cc.dd Filled with a handle to be used in future ptapi calls
Returns	PTAPI_OK - retHandle contains a valid file descriptor PTAPI_INVALID_PARAM – dncsHost or retHandle not specified properly PTAPI_ERROR - error establishing the connection
Comments	None
Exceptions	None
Example	None
See Also	None

Continued on next page

ptapi_createConnectionFromSocket

A routine to create a handle from existing TCP socket connection to the Messaging Server PassThru port established outside of the scope of this API.

Specification	**Details**
Syntax	`int ptapi_createConnectionFromSocket (PTAPI_socket sock, PTAPI_Handle *retHandle);`
Parameters	
$\rightarrow$ sock	Specifies TCP connection to PassThru port (already assumed to be connected by the caller) (on Unix systems, this should specify an open file descriptor)
$\leftarrow$ retHandle	Filled with a handle to be used in future ptapi calls
Returns	PTAPI_OK - retHandle contains a valid handle PTAPI_INVALID_PARAM - sock not a valid socket
Comments	None
Exceptions	None
Example	None
See Also	None

Continued on next page

ptapi_deleteConnection

A routine to delete connection created by ptapi_createConnectionXXX.

Specification	**Details**
Syntax	`int ptapi_deleteConnection(PTAPI_Handle h);`
Parameters → h	 A handle created using ptapi_createConnectionXXX routine
Returns	PTAPI_OK - handle was deleted and if applicable, socket was closed properly. PTAPI_INVALID_PARAM - h does not specify a valid handle PTAPI_ERROR - there was an error closing socket.
Comments	None
Exceptions	None
Example	None
See Also	None

Continued on next page

ptapi_freePassThruMessage

This routine frees a message allocation by ptapi_readPassThruMessage.

Specification	Details
Syntax	`int ptapi_freePassThruMessage (PTAPI_DsmccHeader *hdr);`
Parameters → hdr	 Pointer to message header previously allocated by ptapi_readPassThruMessage
Returns	PTAPI_OK - message header freed successfully PTAPI_INVALID_PARAM - hdr does not point to a valid DSM-CC header
Comments	None
Exceptions	None
Example	None
See Also	None

Continued on next page

ptapi_getAdaptationHeader

A routine to return adaptation bytes given a DSM-CC PassThru header.

Specification	Details
Syntax	`int ptapi_getAdaptationHeader (PTAPI_DsmccHeader *hdr, unsigned char **retAdaptationBytes);`
Parameters	
→ hdr	DSM-CC PassThru header (constructed by user or one returned from ptapi_readPassThruMessage)
← retAdaptationBytes	A pointer to a pointer which will be filled with address of start of adaptation bytes.
Returns	PTAPI_OK - a valid adaptation byte address was returned. The retAdaptationBytes can still be zero (NULL) if there was no adaptation data. PTAPI_INVALID_PARAM - hdr does not point to a valid DSM-CC header
Comments	Caller should not free data at the returned address.
Exceptions	None
Example	None
See Also	`ptapi_freePassThruMessage`

Continued on next page

ptapi_getHCTNsapAddr

A routine to return NSAP address given a DHCT MAC address. Currently, it follows DNCS conventions to make an NSAP address. However, in the future, this will be enhanced to use DNCS to do a name lookup.

Specification	**Details**

Syntax	`int ptapi_getHCTNsapAddr(const unsigned char *hctMac,`
	`                      unsigned char *hctNsap);`

Parameters → hctMac ← hctNsap	 HCT MAC address A pointer to 20 bytes of data that is filled with NSAP address of a DHCT
Returns	PTAPI_OK - Success, hctNsap was filled in with proper value PTAPI_INVALID_PARAM - hctMac or hctNsap are not specified properly PTAPI_ERROR - An error occurred trying to get HCT NSAP
Comments	None
Exceptions	None
Example	None
See Also	None

Continued on next page

ptapi_getPassThruMessagePayload

A routine to return PassThru message payload given a DSM-CC PassThru header.

Specification	Details

Syntax	`int ptapi_getPassThruMessagePayload`
	`(PTAPI_DsmccHeader *hdr,`
	`void**retPayload);`

Parameters	
→ hdr	DSM-CC PassThru header (constructed by user or one returned from ptapi_readPassThruMessage)
← retPayload	A pointer to a pointer that is filled with the address of start of payload bytes.
Returns	PTAPI_OK - a pointer to payload was returned in retPayload. PTAPI_INVALID_PARAM - hdr does not point to a valid DSM-CC PassThru header

Comments	Caller should not free the data at the returned address.
	
	Depending on value of hdr->messagdId value in return payload may be cast as follows:
	`**      hdr->messageId                                   *retPayload`
	`**`
	`**   0x0001 (PTAPI_DSMCC_UN_PassThruReq)      PTAPI_UN_PassThruReq*`
	`**   0x0002 (PTAPI_DSMCC_UN_PassThruInd)      PTAPI_UN_PassThruInd*`
	`**   0x0003 (PTAPI_DSMCC_UN_PassThruRecReq)   PTAPI_UN_PassThruRecReq*`
	`**   0x0004 (PTAPI_DSMCC_UN_PassThruRecCnf)   PTAPI_UN_PassThruRecCnf*`
	`**   0x0005 (PTAPI_DSMCC_UN_PassThruRecInd)   PTAPI_UN_PassThruRecInd*`
	`**   0x0006 (PTAPI_DSMCC_UN_PassThruRecRsp)   PTAPI_UN_PassThruRecRsp*`

Exceptions	None
Example	None
See Also	`ptapi_freePassThruMessage`

Continued on next page

ptapi_getSocket

A routine to return a socket from a PTAPI_Handle. On a Unix system, this returns a file descriptor. On other systems, an appropriate socket handle is returned.

Specification	Details
Syntax	`int ptapi_getSocket(PTAPI_Handle h, PTAPI_socket *retSocket);`
Parameters → h ← retSocket	 A handle created using ptapi_createConnectionXXX routine A pointer that is filled with return socket (for Unix systems, this is a file descriptor (int))
Returns	PTAPI_OK - a valid socket was returned PTAPI_INVALID_PARAM - h does not specify a valid handle
Comments	None
Exceptions	None
Example	None
See Also	None

Continued on next page

ptapi_initConnection

A routine to establish the identity of the server using this TCP connection to the Messaging Server. It sends an empty PassThruRequest with a DSM-CC adaptation header containing the NSAP address of the server.

Specification	Details
Syntax	`int ptapi_initConnection(PTAPI_Handle h, unsigned char *serverNsap);`
Parameters	
→ h	A handle created using ptapi_createConnectionXXX routine
→ serverNsap	NSAP address of the server application. This is established when the server is registered as VASP. Currently the QueryVasp BOSS transaction is not supported so the application must provide an alternate means of registering this value.
Returns	PTAPI_OK - initialization successful PTAPI_INVALID_PARAM - serverNsap not valid PTAPI_ERROR - Error sending message to DNCS
Comments	The use of this routine is mandatory unless the server application includes an adaptation header in the messages it sends. Also, the server application must send a message with such an adaptation header in order to receive messages.
Exceptions	None
Example	None
See Also	None

Continued on next page

ptapi_readPassThruMessage

A routine to read PassThru message. Typically, an application server uses a select() call to wait until data is available on this descriptor, then calls this routine to read the message. The returned message must be freed using ptapi_freePassThruMessage().

Specification	Details

Syntax	`int ptapi_readPassThruMessage(PTAPI_Handle h,`
	`                    PTAPI_DsmccHeader **rethdr);`

Parameters → h ← rethdr	 A handle created using ptapi_createConnectionXXX routine A pointer to a pointer to PTAPI_DsmccHeader
Returns	If successful, rethdr is filled with a pointer to memory containing the DSM-CC PassThru header followed by any optional adaptation header and the PassThru data (there are convenience routines to access the latter). The value returned by this routine must be freed using ptapi_freePassThruMessage().

Comments	The following is the format of the returned message:
	`**`
	`**        field                 length          value`
	`**`
	`**  protocolDiscriminator    1       0x11 = DSM-CC`
	`**  dsmccType                1       0x05 = UN-PassThru`
	`**  messageId                2       Depends on dsmccType`
	`**                                   for dsmccType=UN-PassThru`
	`**                                   0x0001 PassThruReqest`
	`**                                   0x0002 PassThruInd`
	`**                                   0x0003 PassThruReceiptRequest`
	`**                                   0x0004 PassThruReceiptConfirm`
	`**                                   0x0005 PassThruReceiptInd`
	`**                                   0x0006 PassThruReceipt`
	`**`
	`**  transactionId            4       user assigned (see DSM-CC docs)`
	`**  reserved                 1       0xff`
	`**  adaptationLength         1       Adaptation length`
	`**  messageLength            2       Length of message from here on`
	`**   if (adaptationLength > 0) {`
	`**      adaptationHeader()`
	`**   }`
	`**   MessagePayload()        x   depends on messageId - see DSM-CC`
	docs

Exceptions	None

See Also	`ptapi_getPassThruAdaptationHeader,`
	`ptapi_getPassThruMessagePayload`

Continued on next page

ptapi_sendPassThruRecieptRequest

A routine to send a PassThruReceiptRequest message to the Messaging Server addressed to a DHCT NSAP address.

Specification	Details
Syntax	`int ptapi_sendPassThruRequest(PTAPI_Handle h, unsigned char *srcNsap, unsigned char *dstNsap, unsigned short passThruType, unsigned short dataLength, const unsigned char *data, unsigned long transactionId);`
Parameters	
→ h	A handle created using ptapi_createConnectionXXX routine
→ srcNsap	NSAP address of the message source
→ destNsap	NSAP address of the destination DHCT
→ passThruType	PassThru message type for this application (PowerTV keeps a registry of application defined message IDs)
→ dataLength	Length of PassThru data
→ data	PassThru data (payload) to be sent
→ transactionId	Transaction Id of the message. Transaction Ids are truly meaningful when requesting a reply, but they can be useful for debugging - specify zero if not needed.
Returns	PTAPI_OK PTAPI_INVALID_PARAM PTAPI_ERROR
Comments	None
Exceptions	None
Example	None
See Also	`ptapi_sendPassThruRequestMAC, ptapi_getHCTNsapAddr`

Continued on next page

ptapi_sendPassThruReceiptRequestMac

A routine to send a PassThruReceiptRequest message to the Messaging Server addressed to a DHCT MAC address. This is a convenience routine that translates MAC address into NSAP address, then calls ptapi_sendPassThruRequest.

Specification	Details
Syntax	`int ptapi_sendPassThruRequestMac(PTAPI_Handle h, unsigned char *srcNsap, unsigned char *hctMac, unsigned short passThruType, unsigned short dataLength, const unsigned char *data, unsigned long transactionId);`
Parameters	
→ h	A handle created using ptapi_createConnectionXXX routine
→ srcNsap	NSAP address of the message source
→ hctNsap	MAC address of the destination HCT
→ passThruType	PassThru message type for this application (PowerTV keeps a registry of application defined message IDs)
→ dataLength	Length of PassThru data
→ data	PassThru data (payload) to be sent
→ transactionId	Transaction Id of the message. Transaction Ids are meaningful when requesting a reply, but they can be useful for debugging - specify zero if not needed.
Returns	PTAPI_OK PTAPI_INVALID_PARAM PTAPI_ERROR
Comments	None
Exceptions	None
Example	None
See Also	`ptapi_sendPassThruRequest, ptapi_getHCTNsapAddr`

Continued on next page

ptapi_sendPassThruReceiptResponse

A routine to send a PassThruReceiptResponse message to DNCS.

Specification	Details

Syntax	`int ptapi_sendPassThruReceiptResponse (PTAPI_Handle h,`
	`unsigned short response,`
	`unsigned short dataLength,`
	`const unsigned char *data,`
	`unsigned long transactionId);`

Parameters	
→ h	A handle created using ptapi_createConnectionXXX routine
→ response	The response code for the reply (user defined)
→ dataLength	Length of PassThru data
→ data	PassThru data (payload) to be sent
→ transactionId	Transaction Id of the message. This must be the same value that was specified in a PassThruReceiptInd message asking for this response.
Returns	PTAPI_OK PTAPI_INVALID_PARAM PTAPI_ERROR
Comments	None
Exceptions	None
Example	None
See Also	None

Continued on next page

ptapi_sendPassThruRequest

A routine to send a PassThruRequest message to the Messaging Server addressed to a DHCT NSAP address.

Specification	Details

Syntax	`int ptapi_sendPassThruRequest(PTAPI_Handle h,`
	`unsigned char *dstNsap,`
	`unsigned short passThruType,`
	`unsigned short dataLength,`
	`const unsigned char *data,`
	`unsigned long transactionId);`

Parameters	
→ h	A handle created using ptapi_createConnectionXXX routine
→ destNsap	NSAP address of the destination DHCT
→ passThruType	PassThru message type for this application (PowerTV, Inc. keeps a registry of application defined message IDs)
→ dataLength	Length of PassThru data
→ data	PassThru data (payload) to be sent
→ transactionId	Transaction Id of the message. Transaction Ids are meaningful when requesting a reply, but they can be useful for debugging - specify zero if not needed.
Returns	PTAPI_OK PTAPI_INVALID_PARAM PTAPI_ERROR
Comments	None
Exceptions	None
Example	None
See Also	`ptapi_sendPassThruRequestMac, ptapi_getHCTNsapAddr`

Continued on next page

ptapi_sendPassThruRequestMac

A routine to send a PassThruRequest message to the Messaging Server addressed to a DHCT MAC address. This is a convenience routine that translates MAC address into NSAP address and calls ptapi_sendPassThruRequest.

Specification	Details
Syntax	`int ptapi_sendPassThruRequestMac PTAPI_Handle h, unsigned char *hctNsap, unsigned short passThruType, unsigned short dataLength, const unsigned char *data, unsigned long transactionId);`
Parameters	
→ h	A handle created using ptapi_createConnectionXXX routine
→ destNsap	NSAP address of the destination DHCT
→ passThruType	PassThru message type for this application (PowerTV, Inc. keeps a registry of application defined message IDs)
→ dataLength	Length of PassThru data
→ data	PassThru data (payload) to be sent
→ transactionId	Transaction Id of the message. Transaction Ids are meaningful when requesting a reply, but they can be useful for debugging - specify zero if not needed.
Returns	PTAPI_OK PTAPI_INVALID_PARAM PTAPI_ERROR
Comments	None
Exceptions	None
Example	None
See Also	`ptapi_sendPassThruRequest, ptapi_getHCTNsapAddr`

Continued on next page

Messaging Client

The PowerTV Session Manager API provides a mechanism for an application client to register for and receive DSM-CC PassThru messages; see the *PowerTV OS 1.5 API* documentation.

Window Manager

Window Manager Server

The Window Manager on the server depends on the hardware platform upon which the application server is running. It could be X/Motif, HTML, or Java.

Window Manager Client

Application clients can optionally utilize the SARA Window Manager as described in the *SARA Application Developer's Guide*. A future release will include the Window Manager client as part of the PowerTV Operating System.

Chapter 2
Digital Network Control System (DNCS) APIs

Overview

Introduction

This chapter contains technical information on the Digital Network Control System (DNCS) component APIs.

In this Chapter

This chapter contains the following topics.

Broadcast Control Suite (BCS)

BCS Server

The BCS Server API is available through the following BOSS transactions:

- DefineSourceId
- RetireSourceId
- DefineSourceSecurity
- QuerySourceSecurity
- RetireSourceSecurity
- DefineSource
- QuerySource
- RetireSource
- DefineSegment
- QuerySegment
- RetireSegment

See the *BOSS Phase 1.0 Interface Specification* for details. Source and segment functionality is also available through the Source and Segment buttons on the DNCS System Provisioning tab of the Admin Console GUI.

BCS Client

On the client side, the PowerTV TV Manager API provides access to tune analog and digital video content. PowerTV MPEG Transport allows access to MPEG transport streams directly, as well as the PowerTV Stream Manager API using the “mpegt” protocol. See the *PowerTV OS 1.5 API* documentation.

Conditional Access Manager (CAM)

CAM Server

The CAM Server API is available through the following BOSS transactions:

- DefinePackage
- QueryPackage
- DeletePackage
- ExtendEvent

See the *BOSS Phase 1.0 Interface Specification* for details. Package functionality is also available through the Package button on the DNCS System Provisioning tab of the Admin Console GUI.

CAM Client

On the client side, the PowerTV TV Manager `tv_IsAuthorized()` API provides limited access to the CAM Client. See the *PowerTV OS 1.5 API* documentation.

DSM-CC Session Manager

Session Manager Server

Interface Requirements

DSM-CC Message Format

The Session Manager signaling interface supports a subset of the User to Network session messages. These messages are defined by the Digital Storage Media Command and Control (DSM-CC) specification (ISO / IEC DIS 13818-6) as well as extensions specified in this document. The subset includes messages required for services that are supported by the DNCS.

The Session Manager accepts DSM-CC user Network Session Messages. These messages consist of two composite parts, which are the DSM-CC Message Header and the Message payload. The following tables detail the use of different fields in a DSM-CC User to Network message. The sizes are specified in bytes, unless otherwise specified.

DSM-CC Message Header

Field Name	Size	Description
protocolDiscriminator	1	As defined by DSM-CC.
DsmccType	1	0x02 for U-N messages.
MessageId	2	As defined by DSM-CC.
TransactionId	4	Set by the client or server, which is initiating a command sequence.

Continued on next page

Field Name	Size	Description
(reserved)	1	Not Used.
AdaptationLength	1	If non-zero, it must be a multiple of 4 and the adaptation data must be padded to the specified length. All messages originating from a Server must include the UserId adaptation. All other adaptations are ignored.
MessageLength	2	Length of the message including dsmccAdaptationHeaderData.
DsmccAdaptationHeaderData	0	All messages originating from a Server must include the UserId adaptation. All other adaptations are ignored.

Supported DSM-CC Message Id sub fields

messageDiscriminator	messageScenario	messageType
2 bits	10 bits	4 bits
1 for Client<->SM messages 2 for Server<->SM messages	Supported values include: 0x01 - SessionSetup 0x02 - SessionRelease 0x03 - AddResource 0x05 - CFSessionSetup 0x06 - StatusMessage 0x08 - Proceeding 0x09 - Connect 0x0b - SessionInProgress	0 - Request Message: unsolicited User to SM message 1 - Confirm Message: SM to User related to a Request message 2 - Indication Message: unsolicited SM to User message. 3 - Response Message: User to SM related to an Indication message.

Continued on next page

Client Communications Interface

The communication interface between the SRM and clients uses UDP / IP protocols. The SRM listens to UDP port 13819 for messages from a client. The SRM sends messages to a client at UDP port 13819. The maximum size for SRM / client signaling messages is defaulted to 4096 bytes for the payload.

Server Communications Interface

The communications interface between the SRM and servers uses TCP / IP protocols. The SRM listens to TCP port 13819 for connect requests from servers. It is the responsibility of the Server to establish a connection to the SRM. The SRM does not initiate connection to Servers. The maximum size for SRM / server signaling messages is defaulted to 64 Kbytes. All messages from the Server must use the UserId adaptation.

Message Specification

This section outlines the Server and Signaling Interfaces. Details for signaling scenarios are provided where additional clarification is needed or implementation specific information is required. The following tables show the abbreviations of DSM-CC and DNCS messages used in subsequent message flows.

DSM-CC Message Abbreviations for Exclusive Session Setup

Abbreviation	Message Name
CSSUReq	ClientSessionSetupRequest
SSSUInd	ServerSessionSetupIndication
CSPInd	ClentSessionProceedingIndication
SARReq	ServerAddResourceRequest
SARCnf	ServerAddResourceConfirm
SSSURsp	ServerSessionSetupResponse
CSSUCnf	ClientSessionSetupConfirm
CCReq	ClientConnectRequest
SCInd	ServerConnectIndication

Continued on next page

DSM-CC Message Abbreviations for Continuous Feed Session Setup

8Abbreviation	Message Name
SCFSReq	ServerContinuousFeedSessionRequest
SCFSCnf	ServerContinuousFeedSessionConfirm

DSM-CC Message Abbreviations for Status

Abbreviation	Message Name
SSReq	ServerStatusRequest
SSCnf	ServerStatusConfirm
CSReq	ClientStatusRequest
CSCnf	ClientStatusConfirm

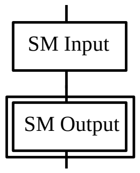
DSM-CC Message Abbreviations for Session Release

Abbreviation	Message Name
CRReq	ClientReleaseRequest
CRCnf	ClientReleaseConfirm
CRInd	ClientReleaseIndication
CRRsp	ClientReleaseResponse
SRReq	ServerReleaseRequest
SRCnf	ServerReleaseConfirm
SRInd	ServerReleaseIndication
SRRsp	ServerReleaseResponse

Continued on next page

The following diagram shows how Session Manager input and output messages are represented in message flows:

SM Message Flow Indicators

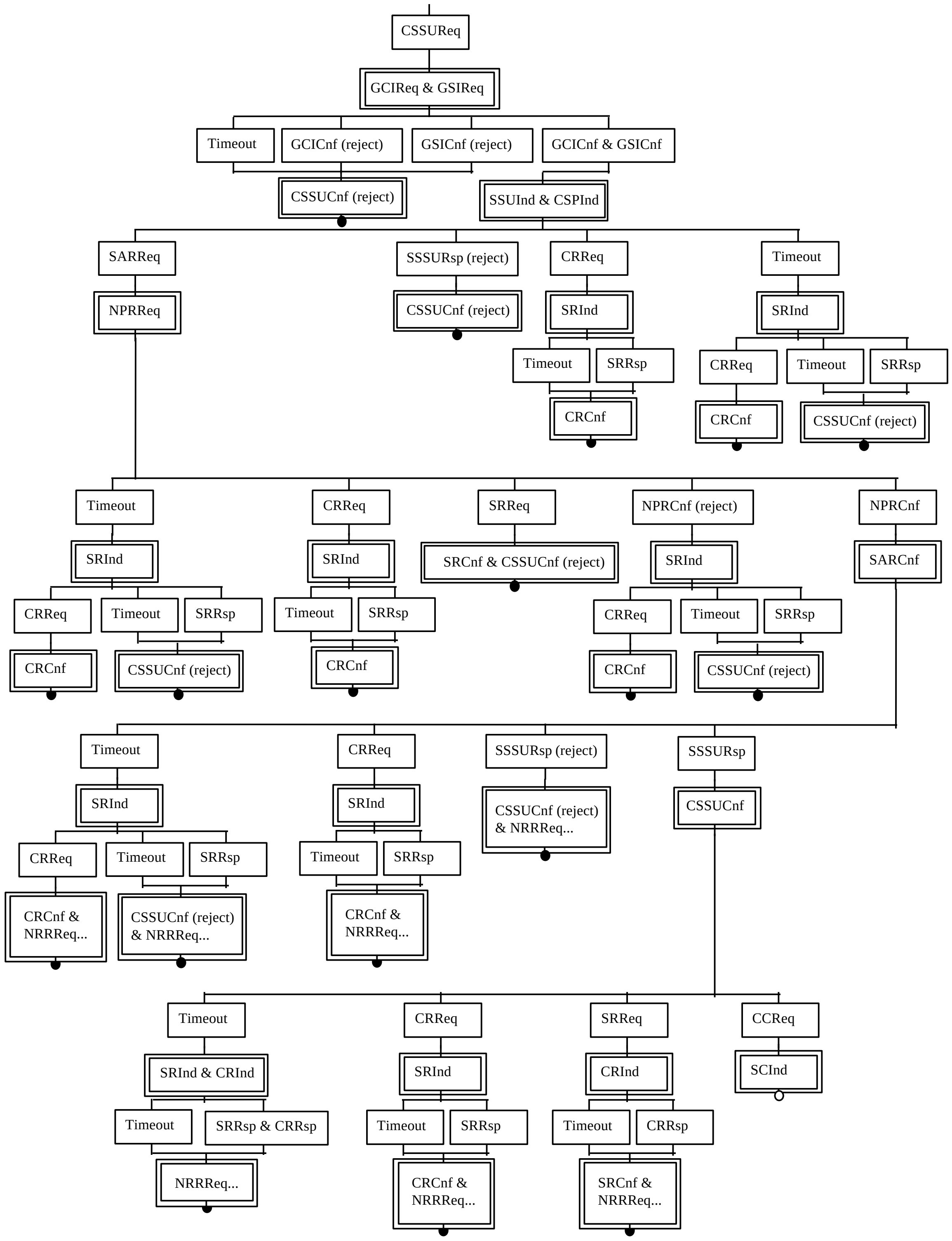


Client Session Setup (Exclusive Sessions)

Client Session Setup is used to establish exclusive sessions between a server and a client. The following diagram represents the possible message flows for Client Session Setup. This does not include services where content from a single server is shared among many clients.

Continued on next page

Client Session Setup Message Flow



Continued on next page

Successful Client Session Setup Scenario

1.	A ***ClientSessionSetupRequest*** is received to initiate the session setup. For a Client's first session, the SRM validates the client with the DNCS database. The source IP Address of the request is validated as belonging to a Client that is In Service and has an associated QPSK modem. The *ClientId* contained in the request is validated independently using the same criteria. The *SessionId* is set by the Client and must be unique among all pending, active and expiring sessions. If a connection exists to the server identified by *ServerId*, the SRM proceeds with session setup. The SRM does not initiate connections to servers.
2.	A ***ClientSessionProceedingIndication*** message is sent to the client to acknowledge receipt of the session request. The SRM sets the CSPItimer. If the CSPItimer expires before the SRM sends the ***ClientSessionSetUpConfirm***, the SRM sends another ***ClientSessionProceedingIndication*** message and the CSPItimer is reset.
3.	A ***ServerSessionIndication*** is sent to the specified server. The *SessionId, ClientId, ServerId* and *userData* of this message are the same as the ***ClientSessionSetupRequest***. The *forwardServerList* is empty (i.e., forwardCount = 0). Session "forwarding" is not supported by this release of DNCS. The SRM sets the SSSUItimer. If the SSSUItimer expires, the SRM begins release procedures.
4.	Upon receipt of a ***ServerAddResourceRequest***, the SSSUItimer is canceled. The *userData* of field this message is expected to be empty. Any values in that field will be ignored. A "resource set" is extracted from the *ResourceList* by the SRM.
5.	The SRM validates the "resource set" and issues a ***NetworkProvisionResourcesRequest*** (not a DSM-CC UN message). The SRM sets the NPRRtimer. If this timer expires, the SRM begins release procedures.
6.	Upon receipt of a ***NetworkProvisionResourcesConfirm*** (not a DSM-CC UN message) the SRM cancels the NPRRtimer. This message contains the resource set that represents the network resources allocated for the session.
7.	A ***ServerAddResourceConfirm*** is sent to the server. The *ResourceList* is populated with the server's view of the allocated Resource Descriptors. The *userData* structure is empty. The SRM sets the SARCtimer. If the this timer expires, the SRM begins release procedures.

Continued on next page

8.	Upon receipt of the ***ServerSessionSetUpResponse***, the SARCtimer is canceled.
9.	A ***ClientSessionSetUpConfirm*** is sent to the client. The *userData* structure is taken from the ***ServerSessionSetUpResponse***. The *ResourceList* is populated to the client's view by the Resource Descriptor API. SessionId and ServerId are the same values as the initial session request. At this point the session becomes active.
10.	If a ***ClientConnectRequest*** is received, the SRM will send a ***ServerConnectIndication*** containing the SessionId and userData from the ***ClientConnectRequest***.

Client Session Setup Failure Descriptions

Condition	Action
Server ID in request is not valid.	A ***ClientSessionSetupResponse*** is sent with response code set to *rspNeInvalidServer*.
ClientId in request is not valid.	A ***ClientSessionSetupResponse*** is sent with response code set to *rspNeInvalidClient*
Client is not authorized for the service.	A ***ClientSessionSetupResponse*** is sent with response code set to *rspNeInvalidClient*
SessionId is not unique.	A ***ClientSessionSetupResponse*** is sent with response code set to *rspNeNoSession*
The Server does not have a connection to the SRM.	A ***ClientSessionSetupResponse*** is sent with response code set to ***rspSeNoCalls.***
Server rejects ServerSessionSetupIndication with a ServerSessionSetupResponse (response != rspOK).	A ***ClientSessionSetupResponse*** is sent with response code set to the value from the ***ServerSessionSetupResponse***

Continued on next page

Server does not respond to ServerSessionSetupIndication.	A ***ClientSessionSetupResponse*** is sent with response code set to *rspSeNoCalls*.
Client initiates early release.	If the server has not been sent a ***ServerSessionSetupIndication***, only a ***ClientReleaseResponse*** is sent. Otherwise, the Client Initiated release procedures are initiated.
Network Resources cannot be allocated.	A ***ServerReleaseIndication*** and a ***ClientSessionSetupResponse*** are sent with response code set to *rspNeResourceFailed*.
Server initiates early release.	A ***ClientSessionSetupResponse*** is sent with response code set to the value from the ***ServerReleaseRequest***.
Server rejects network resources allocated for the session.	A ***ClientSessionSetupResponse*** is sent with response code set to from the ***ServerSessionSetupResponse***.
Server does not respond to ServerAddResourcesConfirm.	A ***ClientSessionSetupResponse*** is sent with response code set to *rspSeNoCalls*.

Client Session In Progress

Client Session In Progress is required at periodic intervals as a session "keep alive" for exclusive sessions. When an exclusive session becomes active, a CSIPtimer is set. When a Client Session in Progress message is received the CSIPtimer is reset. If this timer expires twice in successive intervals, the SRM initiates session release procedures.

Server Session In Progress

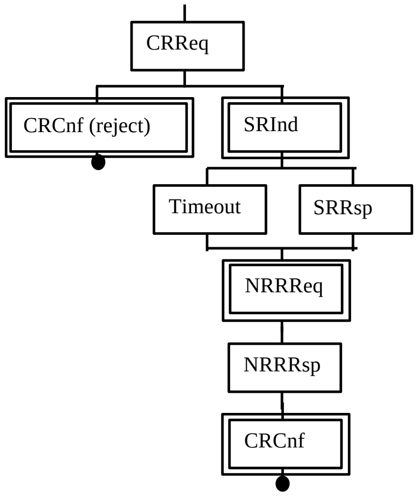
This message is optional for a server. The SRM processes this message, but this processing does not affect the state of sessions or resources associated with the Server.

Client Session Release (Exclusive Sessions)

The Client Session Release sequence provides the mechanism for a client to end a session.

Continued on next page

Message Flow for Client Session Release



Client Session Release Error Conditions

Since the release sequence is clearly defined by DSM-CC, only error conditions are presented.

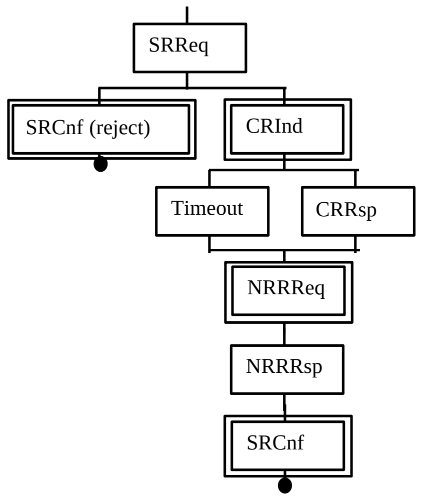
Condition	Action
The client making the request is not the Client of the session.	A ***ClientReleaseConfirm*** is sent with the response code set to *rspNeInvalidClient*

Server Session Release (Exclusive Sessions).

The Server Session Release is the sequence where the server initiates an exclusive session tear down. This tear down sequence involves the server, the network and a client.

Continued on next page

Server Session Release Message Flow



Server EX Session Release Error Conditions

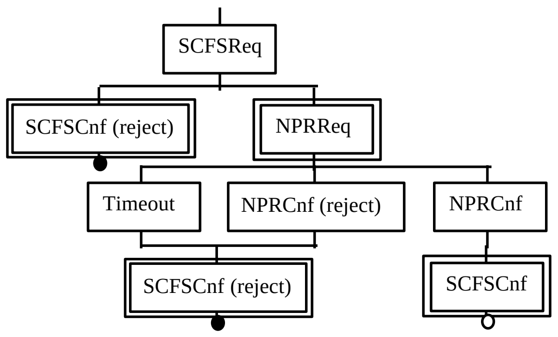
Since the release sequence is clearly defined by DSM-CC, only error conditions are presented.

Condition	Action
The server making the request is not the server of the session.	A ***ServerReleaseConfirm*** is sent with the response code set to *rspNeInvalidServer*

Server Continuous Feed Session Setup

Server CFSession Setup is used to establish sessions where the resources can be shared by many clients. Applications of this functionality include Digital Broadcast, PPV, IPPV, NVOD and EPG software and data download.

Server Continuous Feed Session Setup Message Flow



Continued on next page

Successful Server CFS Setup Scenario

1.	A ***ServerContinuousFeedSessionRequest*** is received to initiate session setup. The *SessionId* is set by the Server and must be unique among all pending, active and expiring sessions. The SRM extracts the “resource set” from the *ResourceList*.
2.	The SRM validates the “resource set” and issues a ***NetworkProvisionResourcesRequest*** (not a DSM-CC UN message). The SRM sets the NPRRtimer. If this timer expires, the SRM fails the session setup.
3.	Upon receipt of a ***NetworkProvisionResourcesConfirm*** (not a DSM-CC UN message) the SRM cancels the NPRRtimer. This message contains the resource set that represents the network resources allocated for the session.
4.	A ***ServerSessionSetupConfirm*** is sent to the Server with the *ResourceList* that is populated with the server’s view of allocated Resource Descriptors. At this point the session becomes active.

CFS Setup Failure Descriptions

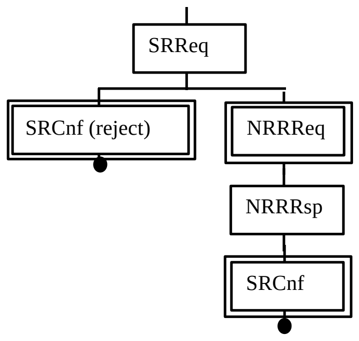
Condition	Action
Server ID in request is not vaild.	A ***ServerSessionSetupResponse*** is sent with response code set to *rspNeInvalidServer*.
SessionId is not unique.	A ***ServerSessionSetupResponse*** is sent with response code set to *rspNeNoSession*
Network Resources cannot be allocated.	A ***ServerSessionSetupResponse*** is sent with response code set to *rspNeResourceFailed*.

Server Continuous Feed Session Release.

The Server Session Release sequence allows the server to delete a continuous feed session. This tear down sequence involves only the server and the network. The network will begin release sequences for all shared exclusive sessions that are bound to a CF session when it is released.

Continued on next page

Server CF Session Release Message Flow



Server CF Session Release Error Conditions

Since the release sequence is clearly defined by DSM-CC, only error conditions are presented.

Condition	Action
The server making the request is not the server of the session.	A ***ServerReleaseConfirm*** is sent with the response code set to *rspNeInvalidServert*

Server Status

Server Initiated Session Status is used by Servers needing to synchronize their view of sessions with that of the Network. This sequence is performed after the server make its TCP connection to the network. The supported status types are SessionList (0x0001) and SessionStatus (0x0002). The SessionList status provides the server with a list of SessionIds for active sessions associated with the server. The SessionStatus status provides the server with the ResourceList for a particular session.

Client Status

Client Initiated Session Status is used by Clients needing to synchronize their view of sessions with that of the Network. The supported status types are SessionList (0x0001) and SessionStatus (0x0002). The SessionList status provides the client with a list of SessionIds for active sessions associated with the client. The SessionStatus status provides the client with the ResourceList for a particular session.

Continued on next page

Session Resource Management

This section describes resource management performed by the SRM. DSM-CC Resource descriptors describe the network connection parameters that are needed to deliver a service associated with a session. A DSM-CC Server is the requester of resources for both exclusive and continuous feed sessions. The resource descriptors used in the Server resource request are the Server View resource set. As session setup proceeds, the SRM processes the initial resource set and generates additional resource descriptors that correspond to the Client View of the resource set. The Client View portion of the resource set describes the network parameters required by a Client to access the service associated with a session.

Resource Descriptors

The SRM supports a subset of the resource descriptors defined by DSM-CC, as well as 4 user defined resource descriptors. Except where noted, only "single value type" is supported by the SRM for resource fields that are variable. Following is a list of supported resource descriptors and a brief description of their use.

DSM-CC Resource Descriptors

Resource Name	Description
AtmConnection	Describes ATM connection parameters NSAP address, VPI and VCI. When requesting this resource, a Server provides only the NSAP address that represents the Server's point of connection point to the ATM network. The Network populates the VPI and VCI fields in the response.
MpegProgram	Provides MPEG Program and Packet Identifier information for a MPEG Program stream. The context of the resource request defines what fields are used. The Conditional Access PID is not used.
PhysicalChannel	Describes the channel used for tuning sessions. All fields are set by the Network. The channelId field is the QAM frequency in Hertz. The direction field is "Downstream".
TSDownstreamBandwidth	Describes connection parameters from the Server to the Network in the Server View and from the Network to the Client in the Client View. In the resource request, the Server sets the bandwidth field and, where applicable, the transport stream Id field.

Continued on next page

Resource Groups

The SRM uses the association tag of resource descriptors to break resource sets into resource groups. Combinations of resource groups are used by the SRM to allocate network resources for sessions. The following list identifies the supported resource groups.

- **Downstream MPEG Group** is used to identify the connection parameters for a CFSession that is submitted to the network as a MPEG Program or Transport stream. (IRT and RTE sessions).
- **Downstream ATM MPEG Group** is used to identify the connection parameters for a CFSession that is introduced to the network as a MPEG Program stream which is carried over an ATM connection.

The following sections outline the Server view resource descriptors for each resource group. Client view resources are presented in the Resource Group Combination section.

Downstream MPEG Group

This resource descriptor group is used for sessions where the content originates from a source utilizing MPEG Transport or MPEG elementary streams carryied over a non-switched transport. This group describes the parameters for a Server to obtain a uni-directional connection to the network that may be distributed to one or more downstream MPEG transport streams.

Downstream MPEG Group

Server View Resource Descriptor	Description
TSDownstreamBandwidth	The Server provides both the bandwidth and the transport stream Id in the TSDownstreamBandwidth descriptor. The transport stream Id represents the output of the source device. Single value transport stream Id is supported for all resource requests. List value transport stream Id is supported for exclusive sessions only.
MPEGProgram	For source devices that output MPEG elementary streams without MPEG PSI, the elementary stream table and the PCR PID index must be set by the Server. For MPEG Transport output types only the MPEG program number is required in the Server View.

Continued on next page

Downstream ATM MPEG Group

Sessions where the content originates from a source delivering MPEG / ATM use the Downstream ATM MPEG Group resource descriptors. This group describes the parameters for a Server to obtain a unidirectional connection to a BIG through an ATM network.

Downstream ATM MPEG Group Descriptors

Server View Resource Descriptors	Description
TSDownstreamBandwidth	The bandwidth must be set by the Server and the transport stream Id not used.
AtmConnection	In the resource request, the Server only populates the NSAP address portion of the AtmConnection descriptor. The SRM populates the VPI and VCI in the resource response for the ATMConnection descriptor.
MPEGProgram	The elementary stream table and the PCR PID index must be set by the Server. The other fields of the MPEGProgram descriptor are not used.

Resource Group Combinations

Only specific combinations of resource groups are supported by the SRM. The combinations are composed of the groups needed to support the services provided by sessions. This section outlines the supported Resource Group Combinations and presents the generated client view resource sets.

Supported Resource Group Combinations

Group Combination Name	Groups	Group Cardinality
MPEG Point to Point	Downstream MPEG	1
ATM MPEG Point to Point	Downstream ATM MPEG	1

MPEG and ATM MPEG Point to Point Resource Group Combinations

These resource group combinations are used for Exclusive Sessions, where the content is unidirectional MPEG traffic. The SRM generates the Client view resource descriptors that allow the Client to access session traffic. The Client view resource descriptors use the association tag of the MPEG or ATM MPEG Resource Group. The following table describes the Client View Resource Descriptors.

Continued on next page

Client View Resource Descriptors for MPEG and ATM MPEG Point to Point

Resource Descriptor	Description
TSDownstreamBandwidth	The SRM sets both the bandwidth and the transport stream Id in the TSDownstreamBandwidth descriptor. The transport stream Id represents the output a QAM.
MPEGProgram	Only the MPEG program number is required for the Client View MPEGProgram descriptor.
PhysicalChannel	The SRM sets the channelId to the frequency in Hz of the QAM carrying the session traffic. The direction field is not used.
AtscModulationMode	The SRM sets the fields of this descriptor to the values that correspond to the modulation mode of the QAM carrying the session traffic.
ClientConditionalAccess	The SRM sets the fields of this descriptor to the values that allow the client access to the session content. This descriptor is used only if the Server resource request contains the ServerConditionalAccess descriptor.

References

Document Title	Document Number
MPEG-2 Digital Storage Media Command and Control	ISO/IEC DIS 13818-6 DSM-CC
Business Operations Support System Interface	S-A Drawing No. 541692, Rev. 1.4.3

Session Manager Client

The only Session Manager Client API in this phase is for DSM-CC User-to-Network Configuration via the PowerTV Session Manager API. See the *PowerTV 1.5 OS API* documentation for details.

Index

United States: Scientific-Atlanta, Inc., 4261 Communications Drive, P. O. Box 6850, Norcross, GA 30091-6850; Tel: 770-903-5000; TWX: 810-799-4912; Telex: 0542898
Europe: Scientific-Atlanta Europe, Ltd., Home Park Estate, Kings Langley, Herts WD4 8LZ, England; Tel: +44-1-923-266133; Fax: +44-1-923-269018
Asia-Pacific: Scientific-Atlanta (HK), Ltd., Suite 56-57, 5/F New Henry House, 10 Ice House Street, Central, Hong Kong; Tel: 852-2522-5059; Fax: 852-2522-5624

Part Number 716392 Rev 001 August 1999