Business Operations Support System (BOSS) Phase 1.0

Interface Specification

Revision 3.4

NOTICE OF DISCLAIMER

This interface definition/system overview is published by Scientific-Atlanta, Inc. (S-A) to inform the industry of the requirements and operational characteristics for interaction with the S-A Digital Broadband Delivery System for the delivery of broadcast and/or interactive video services.

S-A reserves the right to revise this document at any time for any reason, including but not limited to, conformity with standards promulgated by various agencies or industry associations, utilization of advances in the state of the technical arts, or the reflection of changes in the design of any equipment, techniques, or procedures described or referred to herein.

S-A makes no representation or warranty, express or implied, of any kind with respect to this document or the any of the information contained herein. YOUR USE OF OR RELIANCE UPON THIS DOCUMENT AND THE INFORMATION CONTAINED HEREIN IS AT YOUR OWN RISK. S-A shall not be liable to you for any damage or injury incurred by any person arising out of the your use of these materials.

This document is not to be construed as conferring by implication, estoppel or otherwise any license or right under any copyright or patent whether or not the use of any information in this document necessarily employs an invention claimed in any existing or later issued patent.

Table of Contents

Revision 3.4

1. Introduction

1.1 Purpose

This document describes the Business and Operations Support System (BOSS) interface, which is the interface between the Digital Network Control System and the Administrative Gateways of Access Network Providers (ANPs) for both Analog Broadcast Services and Digital Broadcast.

The intended audience of this document consists of:

- Customers of the BOSS Interface both internal and external
- Developers of Digital Network Control System (DNCS) components
- Test Engineers, tasked with authoring the BOSS Interface Test Plan and Test Procedure, and with performing the tests
- Technical Writers, tasked with documenting the implementation and usage of DNCS components.

1.2 Scope

This document describes BOSS Interface transaction at the messaging level. It suggests an Applications Programmatic Interface (API) for use with the BOSS interface, but it places no requirements on that API.

1.3 Definitions, Acronyms, and Abbreviations

Terms	Description
AG	Administrative Gateway
BOSS	Business and Operations Support System
BRID	Bandwidth Reservation ID
CA	Conditional Access
CAT	Conditional Access Table
CMS	Customer Management System
CRL	Certificate Revocation List
CW	Control Word
DBDS	Digital Broadband Delivery System
DHCT	Digital Home Communications Terminal
DIS	Digital Interactive Service
DMS	Digital Multicast Service
DNCS	Digital Network Control System

Terms	Description
DSM-CC	Digital Storage Module Command and Control
ECM	Entitlement Control Message
EMM	Entitlement Management Message
EPG	Electronic Program Guide
ERM	Element Resource Manager. The Element Resource Manager is responsible for communications with the DHCTs. The ERM also is responsible for the managing all DBDS Digital Head Ends, including BIGs, QAMs, QPSK modulators and QPSK demodulators. The ERM allocates resources as the sessions request them.
IP	Internet Protocol
L1	Level 1
MAC Address	Media Access Control
MIB	Management Information Base
MPEG	Moving Pictures Experts Group
MSK	Multisession Key
NIT	Network Information Table
ONC	Open Network Computing
PAT	Program Association Table
PES	Packetized Elementary Stream
PIN	Personal Identification Number
PMT	Program Map Table
PSI	Program Specific Information
RPC	Remote Procedure Call
SBT	Standalone BOSS Transactor
SM	Session Manager. The Session Manager manages all sessions in a DBDS. The SM processes all session-related control system messages, and it communicates directly with service providers and DHCTs through EMs.
SM-20	System Manager 20. The Scientific-Atlanta product that controls S-A's analog network.
SMU	Server Management Unit
SNMP	Simple Network Management Protocol
SQL	Structured Query Language
USID	Universal Service ID
VIP	Video Information Provider
XDR	eXternal Data Representation

1.4 References

- Digital Broadband Delivery System Phase 1.0 System Description
- Digital Network Control System Architecture Product Family Product Specifications Revision 1.4
- RFC1050: RPC: Remote Procedure Call Protocol Specification
- RFC1057: RPC: Remote Procedure Call Protocol Specification: Version 2
- RFC1832: XDR: External Data Representation Standard
- ISO/IEC DIS 13818-6, MPEG-2 Digital Storage Media Command & Control

1.5 Apportioned Features

The purpose of this section is to delineate BOSS features which will not be available in the initial implementations of the BOSS interface.

1.5.1 Non-Responding File Upload

The capability to upload a file containing non-responding DHCTs will not be available in the initial implementation. The following transactions will not be available:

- InitializeNonRespondingUpload
- NonRespondingUploadComplete
- GetNonRespondingRecords

1.5.2 Fetch/Query Transactions

The capability to query the DNCS database in order to retrieve managed objects will not be available in the initial implementations. The following transactions will not be available:

- FetchDhct
- FetchDhctType
- FetchVaspProfile
- FetchBoottermPage
- QueryDhct
- QueryPackage
- QuerySourceSecurity
- QuerySource

- QuerySegment

1.5.3 Reservation PPV Upload

The capability to upload Reservation PPV (RPPV) orders to the Administrative Gateway objects will not be available in the initial implementation. The following transactions will not be available:

- AddRppvOrder
- AddRppvCancel

1.5.4 OpenCable Transactions

The following OpenCable transactions will not be available:

- RegisterHost
- DeregisterHost
- DefineCertificateRevocationList
- QueryCertificateRevocationList
- AddHostToList
- RemoveHostFromList

1.6 Added Features

Table 1-1 contains a list of BOSS features that have been implemented since the last revision of this document.

Transaction	System Release
BootDhct	1.2.5
DefineLineUpGroup	1.3
RetireLineUpGroup	1.3
DefineSubstitutionUnit	1.3
AcceptSubstitutionUnit	1.3
ExtendSubstitutionUnit	1.3
RetireSubstitutionUnit	1.3
GetDhctDiagnostics	1.3

1.7 Revision History

Table 1-2 identifies the revision history of this document.

Revision Number	Modifications
1.4.3	Added this revision history.
	Added a heading for 3.4.2.1 for formatting consistency.
	Fleshed out result descriptor.
	Added detail to the DHCT Administrative Status descriptor.
	Completed transition to Result descriptor for error returns.
	Fixed incorrect references to IP address in the Fetch VIP Profile and Delete VIP request structures.
	Added PrimaryVIPnsap descriptor. Included it in DHCT registration, fetch, and update descriptions.
	Added BootPage descriptor to the DHCT Type transactions in Table 3.1. This corrects an omission in version 1.4.2.
	Corrected transaction table references to read Table 1 [Now reads Table 3.1].
	Clarified description of DNCS component access in section 2.1.3.
	Extended description of descriptors.
	Added MacAddr descriptor to transaction table for SendDhctMessage transaction.
	Removed USID descriptor from the QueryPackage transaction.
	Modified package transactions to eliminate the Paid descriptor.
	Added packageName and packageMembership descriptors.
	Added descriptions of the fields in the MulticastService descriptor.
	Extended the description of the DhctInstantHit transaction.
	Extended the description of the ReplaceDhct transaction to indicate that the replacement DHCT must be provisioned previously in the IDM.
	Added a Mandatory/Optional table for BCAM transactions modeled after the IDM version. Removed the Required/Optional designation from the transaction table.
	Included more ECM-related fields in the MulticastService descriptor.
1.4.4	Corrected descriptor counts in example IDM transactions.
	Corrected Table 1 entries for Fetch and Delete VIP profile operations.
	Added definition for IVSN address in Definitions and Acronyms section.

Revision Number	Modifications
	Removed the reference to Secure RPCs.
	Moved purchase-related descriptors from defineService transaction to definePackage transaction, renamed multicastService descriptor to BasicConditionalAccess.
1.4.4	Changed the PPV descriptor to IPPV. Added cancelWindow and ippvCount fields. Moved to the definePackage transaction.
	Removed the NVOD descriptor, pending further analysis of conditional access support required for NVOD.
	Moved service and package descriptors next to other BCAM descriptors, and reorganized the BCAM descriptors.
	Added bandwidth identification descriptor section.
	Added some definitions and corrected some formatting in definitions.
	Changed PpvBuyRecord to IppvBuyRecord
	Reworked the DefineService transaction to eliminate references to service information.
	Removed the references in the definitions section to STB.
	Made many changes to BCAM sections to adopt a consistent coding style. A refinement of this style will be adopted in a future revision.
	Changed the noSecurityMode field to encryptionMode, and put it in a descriptor attached to services.
	Added a package sub-section to the Managed Objects section, and fleshed out the service sub-section in that section.
	Corrected the paragraph-fragment in section 2.1.4.
	Unified the naming of the DhctMacAddr descriptor.
	Brought the session usage record into compliance with v1.0 of the DNCS architecture spec.
	Added an IPPV purchase cancellation record.
	Added a new and improved BCAM transaction summary. This table replaces the BCAM portion of Table 3-1, and the old M/O table in section 3.4.2. It is presently located in section 3.4.2, but may replace Table 1 [now Table 3-1] in subsequent revisions of this document.
	Defined descriptors for the ReplaceDhct transaction.
	Removed BCAM DHCT and DHCT group message transactions.
	Added the DeleteDhct transaction for the BCAM.
	Added a section describing the Display Channel Table, and the transactions supplied by the BCAM to support it.

Revision Number	Modifications
	Removed the clock adjust record from the session usage data file and the IPPV purchase data file.
	Revised the usage data section.
1.4.5	Extensive editorial changes. No material change to the defined interface.
1.5	Returned enableDis to DhctState descriptor. Added enableAnalog and enableIppv, enablePin, and pin fields.
	Added description and resultCode that USIDs can't be reused until delete operation is fully complete.
	Removed freeTime concept.
	Added description and resultCode that recurrenceInterval is greater than duration.
	Removed months from recurrenceUnit.
	Added more description of Copy Protection.
	Removed purchasable and PES header states from copy protection.
	Changed BasicConditionalAccess descriptor to ServiceSecurity. Added some additional fields.
	Removed DeleteDhct transaction.
	Added a headEndCode to the error response for ReserveBandwidth transaction.
	Made HeadEndList optional on ReserveBandwidth transaction. If omitted indicates the reservation is intended for all headEnds.
	Changed CancelBandwidthReservation to RetireBandwidthReservation and defined it to fail if services are defined against the bandwidth.
	Changed DeleteService to RetireService and defined it to fail if service is member of a package.
	Added a QueryService transaction.
	Added a ChannelId descriptor for defining services on analog bandwidth.
	Added a descriptor summary table that includes field descriptions.
	Added a reference to ONC.
	In Recurring descriptor definition, inserted word "be" before word "greater."
	Corrected all enumerated fields to show enumeration in examples.
	Added narrative to PackageMembership descriptor to explain that packages with 0 services and 0 packages are legal.

Revision Number	Modifications
	Fleshed out IPPV descriptor.
	Added a PPV descriptor.
	Added a description of the standard request/response structures.
	Added AnalogEvent descriptor.
	Added ExtendAnalogEvent transaction.
	Added ModifyDhctState transaction.
`	Updated IPPV purchase report file record formats, included analog record formats.
	Deleted the old Table 3-2 -- billing record formats.
	Added new result codes.
	Added serverNsapAddress to ReserveBandwidth transaction
	Made the *Administrative Status* descriptor mandatory in DHCT registration and update transactions. (CR #8)
	Deleted DHCT group transactions. (CR #9)
	Clarified that DHCT types restrict DHCT registration. A DHCT can not be registered until its DHCT type has been registered. (CR #10)
	Clarified relationship between VIP IP address and VIP NSAP address. (CR #11)
	Defined values for VIP status in the vipInfo descriptor. (CR #13)
1.5.1	Deleted IP address from the list of legal descriptors listed in table 3-2 for the Deregister HCT transaction.
	Added reference for format of an NSAP address to the description of the NSAP descriptor.
	Eliminated all references to IVSN address.
	Added error count to Fetch DHCT response structure.
	Changed session to DSM-CC session
	Sorted the descriptor table.
	Added a reference to RFC1832.
	Corrected “rpcgen” comments.
	Eliminated comment on modeRightToCopy field in IPPV descriptor.
	Corrected usage of quotation marks on examples.
	Added unspecified failure result code to all transactions.
	Sorted result codes.

Revision Number	Modifications
	Made PPV descriptor mandatory for analog.
	Removed specification of enum values.
	Included an apportioned section.
	Changed “purchased” to “purchasable”.
	Fleshed out description of handling of bandwidth descriptors on QueryService transaction.
	Added ChannelId descriptor to QueryService transaction.
	Added a description of processing of mandatory response descriptors when a query transaction fails.
	Removed invalid service name result code.
	Added description of errorCode processing.
	Added detail to description of instant hit.
	Changed QueryDhct transaction description to be consistent with current definition of the DhctState descriptor.
	Added description of update of managed objects.
	Corrected reference to BOSS descriptor definition.
	Added narrative of how SessionIds are obtained.
	Added missing PIN result code.
	Rationalized type usage in transaction table.
2.0	Changed wording of section 2.3.
	Added apportioned designation to bandwidth descriptor.
	Added description of difference between <> and [] notations.
	Added statement of padding property for fixed-length strings and opaque.
	Made KeyCertificate descriptor and PackageName descriptor sort in right order in descriptor summary table.
	Added result code for unspecified error, request parsed correctly.
	Eliminated TBD in section 3.2.2.32.1 VipInfo.
	Added apportioned statement to section 3.3.4.2 bandwidth reservation.
	Added new customer-suggested result codes.
	Corrected sessionId field type.
	Added blackoutRadius.
2.1	Changed wording of Section 1.1

Revision Number	Modifications
	Updated Section 1.3
	Updated references in Section 1.4
	Removed “Multiple Service IPPV Packages” from apportioned features (Section 1.5)
	Changed wording of Section 2.1.5
	Updated Figure 1
	Split Section 2.2.4 describing the BCAM into 2 sections: one to describe the BSM and another to describe the CAM
	Split Section 3 into 3 chapters
	Changed all references to BCAM to refer to BSM or CAM as appropriate
	Changed VIP to Value Added Service Provider (VASP)
	Added ippvEnableFlag, maxIppvEvents, fastRefreshFlag, locationX, locationY, to DhctState
	Removed Usid, replaced with SourceId
	Added SourceName descriptor
	Changed headEndCode to headEndName in HeadEnd
	Added freeCaStartTime, buyWindowStartTime to ImpulsePayPerView
	Changed usidCount to segmentCount in PackageMembership
	Changed usidList to segmentNameList
	Added rightToCopy to PayPerView
	Added DefineSource, QuerySource, RetireSource transactions
	Added EffectiveTimeDescriptor to DefineSource, QuerySource, DefineDct, QueryDct transactions
	Added AddAuthorizations, DeleteAuthorizations transactions
	Added IppvPoll transaction
	Changed ServiceName to SegmentName
	Changed ServiceSecurity to SegmentSecurity
	Added copyProtectionMode to SegmentSecurity
	Changed SessionId to DsmccSessionId
	Changed length of packageName to 21 bytes
	Added AnalogEvent descriptor to DefineSegment, QuerySegment
	Added channelType to DisplayChannelMap

Revision Number	Modifications
	Added HubId descriptor to RegisterDhct, QueryDhct transactions
	Added AnalogScramblingMode descriptor
	Added LoadAnalogScramblingMode transaction
	Removed HeadEndList descriptor
	Changed AdminStatus to DhctAdminStatus in Section 5
	Added RecordType descriptor
	Added IppvPurchase descriptor
	Added new result codes.
	Removed obsolete result codes.
	Moved SecurityMode, BandwidthReservationId, DsmccSessionId, ChannelId from DefineSegment to DefineSource transaction
	Added LimitedDuration to DefinePackage transaction
2.2	Changed wording of Section 2.1.3
	Updated Figure 1
	Added section to describe DNCS Administrative Console
	Added references to PowerKEY™ Control Suite and Broadcast Control Suite
	Corrected typos in Table 4-1
	Changed valid values for dhctAdminStatus field
	Changed resolution of freeCaInterval, butWindowInterfal, cancelWindowInterval to seconds
	Corrected typos in Section 4.2
	Corrected typos in Section 4.2.4
	Corrected typos in Section 4.2.4.1
	Corrected typos in Section 4.2.4.2
	Corrected typos in Section 4.2.4.3
	Changed wording of Section 4.2.6
	Corrected typos in Section 4.2.13
	Removed DisplayChannelMap descriptor
	Corrected typos in Section 4.2.19
	Changed wording of Section 4.2.19
	Changed wording of Section 4.2.29

Revision Number	Modifications
	Added result codes as required
	Changed wording of Section 4.2.36
	Added DhctseSerialNumber to RegisterDhct, FetchDhct, and UpdateDhct transactions
	Added DhctOpnStatus to response to FetchDhct transaction
	Made NsapAddress descriptor optional in RegisterDhct, UpdateDhct transactions
	Corrected typos in Table 5-1
	Removed DefineDct, QueryDct, and DeleteDct transactions
	Added responseRpcProgram field to transaction request format
	Corrected description of Complex Response Format
	Corrected title of Section 5.4.3.2
	Changed wording of Section 5.5.2.1
	Changed wording of Section 5.5.3.1
2.3	Updated Figure 1
	Updated References in Section 1.4
	Updated Section 2.2
	Updated Section 2.3
	Removed ownerVaspId from BoottermPageInfo descriptor
	Added organizationalUnitId field to DhctType descriptor
	Changed dhcttype field in DhctType descriptor to “unsigned short”
	Removed HeadEnd and HubId descriptors from RegisterDhct and UpdateDhct transactions
	Made DhctAdminStatus descriptor mandatory in RegisterDhct transaction
	Removed SourceName descriptor from DefineSource tranaction
	Added HubId descriptor to DefineSource transaction
	Added DefineSourceId, RetireSourceId transactions
	Added Result Codes as required
	Added EffectiveTime descriptor to DefineSegment transaction
	Added EffectiveTime and HubId descriptors to QuerySource and RetireSource transactions
	Removed examples in Descriptor Definitions section

Revision Number	Modifications
	Removed examples in Interface Transactions section
	Updated IPPV Purchase Data Interface section.
	Added valid value for blackoutActive in SegmentSecurity descriptor
	Added EntitlementId descriptor to DefinePackage and QueryPackage transactions
	Removed support for UDP/IP transport protocol in Section 2.1.2
	Renamed *Standalone BOSS Transactor* to *Administrative Gateway Emulator (AGE)*
	Updated Section 2.2.7
	Updated Section 2.3.3
	Added TimeOffset descriptor to RegisterDhct, UpdateDhct, and FetchDhct transactions
	Added SoftwareTOC descriptor to IDM transactions
	Removed allowExtension field from AnalogEvent descriptor
	Added AllowExtension descriptor to DefinePackage, QueryPackage transactions
	Changed ranges of unsigned short fields from 0..2^8 to 0..2^8-1
	Removed “blackout” from blackoutCentroidX, blackoutCentroidY, blackoutRadius field names in SegmentSecurity descriptor
	Updated Section 4.1
	Updated Section 4.2.13
	Marked LoadAnalogScramblingMode transaction as apportioned
	Updated Section 5.2.3
	Updated Section 5.4.1.2
	Added InitializeIppvUpload, IppvUploadComplete, GetIppvRecords transactions
	Added allowWindowCancel to ImplulsePayPerView descriptor
	Replaced ExtendAnalogEvent transaction with ExtendEvent transaction
2.4	Removed SecurityMode from DefineSource transaction
	Added DefineSourceSecurity, QuerySourceSecurity, RetireSourceSecurity transactions
	Added DhctMacAddr descriptor to GetIppvRecords transaction
	Added Result Codes as required
2.5	Marked AnalogScramblingMode descriptor as apportioned

Revision Number	Modifications
	Added ModifyDhctAdminStatus transaction
	Added InitializeNonRespondingUpload, NonRespondingUploadComplete, GetNonRespondingRecords transactions
	Added GetDhctDiagnostics transaction
	Added packageListType to PackageAuthorization descriptor
	Updated Section 1.2
	Updated Section 2.3.3
	Renamed organizationalUnitId field in DhctType descriptor to organizationalUniqueId
	Changed offsetMinutes field in TimeOffset descriptor to “signed long”
	Added EventExtension descriptor to ExtendEvent transaction
	Removed ChannelId and AnalogEvent descriptors from IppvPoll and GetIppvRecords transaction responses
	Removed Analog IPPV Purchase and Cancellation records from Usage Data Interface
2.6	Removed ReserveBandwidth, RetireBandwidthReservation transactions
	Marked FetchDhct, FetchDhctType, FetchVaspProfile, FetchBoottermPage transactions as apportioned.
	Marked QueryDhct, QueryPackage, QuerySourceSecurity, QuerySource, QuerySegment transactions as apportioned
	Changed length of all *name* fields in descriptors from 21 to 20
	Added/Modified Result Codes as required
	Added PollWindow descriptor to GetDhctDiagnostics transactions
	Added valid values for dhctAdminStatus in DhctAdminStatus descriptor
	Changed all *short* and *long* descriptor field types to *integer*
3.0	Added DefineService, QueryService, RetireService transactions
	Added DefineDisplayChannelMap, QueryDisplayChannelMap, RetireDisplayChannelMap transactions
	Added BillingId descriptor to RegisterDhct, UpdateDhct, FetchDhct, UpdateDhctAdminStatus, InitializeIppvUpload transactions
	Added billingId to IPPV purchase/cancellation records
	Updated Section 5.7.2
	Changed valid values for for dhctAdminStatus in DhctAdminStatus descriptor

Revision Number	Modifications
	Added AddRppvOrder, AddRppvCancel transactions (apportioned)
	Marked ReplaceDhct, ExtendEvent as apportioned
	Added/Modified Result Codes as required
3.1	Removed ExtendEvent transaction from list of apportioned features
	Changed upper range of billingId (in BillingId descriptor) to 2^16-1
	Changed upper range of creditLimit (in DhctState descriptor) to 2^16-1
	Changed upper range of modeCost (in ImpulsePayPerView descriptor) to 2^16-1
	Changed length of ShortDescription to 5 characters
	Added ServiceName descriptor to DefineService, QueryService, RetireService transactions
	Added IP Address descriptor to RegisterVasp, UpdateVaspProfile transactions
	Added BillingId descriptor to response to GetIppvRecords
	Changed length of LongDescription descriptor to 32 characters
	Removed DisplayChannelMapName descriptor from QueryDisplayChannelMap, RetireDisplayChannelMap transactions
	Added Service Id descriptor to response to QueryService transaction
3.2	Changed length of DhctMacAddr, DhctMacAddrRep, DhctSerialNumber descriptors to 17 characters
	Modified GetDhctDiagnostics transaction
	Removed AnalogEvent descriptor
	Added AuthorizeExclusiveSession, DeauthorizeExclusiveSession transactions
	Added DefineLineUpGroup, RetireLineUpGroup, DefineSubstitutionUnit, AcceptSubstitutionUnit, ExtendSubstitutionUnit, RetireSubstitutionUnit transactions (apportioned)
	Removed ReplaceDhct transaction from list of apportioned features
	Added/Modified Result Codes as required
3.3	Added NonRespondingArchivePath to response to NonRespondingUploadComplete transaction.
	Removed IpAddress descriptor from RegisterVasp, UpdateVaspInfo transactions
	Added BootDhct transaction (apportioned)

Revision Number	Modifications
	Replaced HubId descriptor with LineUpGroupId in DefineDisplayChannelMap, QueryDisplayChannelMap, RetireDisplayChannelMap transactions
	Added/Modified Result Codes as required
3.4	Changed upper range of maxIppvEvents (in ImpulsePayPerView descriptor) to 72
	Added billingId to Non-Responding DHCT Record
	Added LineUpGroupName descriptor to DefineLineUpGroup transaction
	Added optional BillingID descriptor to InitializeNonRespondingUpload transaction
	Added ModifyDhctHubId transaction (apportioned)
	Removed Exclusive Session transactions
	Removed LoadAnalogScramblingMode transaction
	Removed BootDhct, GetDhctDiagnostics transactions from list of apportioned features
	Removed Channel Substitution transactions from list of apportioned features
	Added RegisterHost, DeregisterHost DefineCertificateRevocationList, AddHostToList, RemoveHostFromList, QueryCertificateRevocationList
	Added RefreshAll descriptor to ModifyDhctConfiguration transaction

2. BOSS Interface Principles of Operations

2.1 Protocols

2.1.1 Message Protocol

The BOSS interface uses Open Network Computing (ONC) Remote Procedure Call (RPC) protocol for sending transaction requests and responses. This protocol is specified with the eXternal Data Representation (XDR) language. The following sections provide the format of the data elements used in the transactions and present the suggested RPC implementation for each transaction.

The BOSS interface uses *asynchronous* remote calls. A BOSS server does not return results with the return from a BOSS request. BOSS clients must register with their local portmapper processes to receive the result from any call to a server. That result will arrive an indeterminate time after the original request is made of the server. Thus, the client must make provisions to pair responses with its requests. This is facilitated with the *messageId* field in each transaction/response.

2.1.2 Transport Protocol

The BOSS interface utilizes TCP/IP transport protocol on an Ethernet network.

2.1.3 DNCS Component Access

The functionality of the BOSS Interface is partitioned into groups, and each functional group is implemented by one DNCS component. However, all BOSS requests are processed by the BOSS server and routed to the proper DNCS component. A client requires only the host address of BOSS server. The port number of the request is discovered via the portmapper, which is a component of the ONC RPC protocol.

2.1.4 Digital Storage Module Command and Control

Digital Storage Module Command and Control (DSM-CC) represents the single avenue by which broadband connections, or sessions, may be created and manipulated. Any client of the network requiring a session through the network must request that session by communicating with the DNCS using DSM-CC.

2.1.5 Descriptors

Many of the following transactions make use of the *Descriptor* concept. The descriptor is a structure containing fields that describe some aspect of a BOSS interface managed object. A single transaction may carry several descriptors to completely describe the desired state of a single managed object.

For example, to define a package that may be accessed only with permission, the following descriptors must be included in the DefinePackage transaction:

- PackageName
- PackageMembership

If it is desired that permission to access such a package may be gained by an impulse purchase at the DHCT, the preceding list of descriptors would include the LimitedDuration and ImpulsePayPerView descriptors.

As this example shows, a descriptor may be optional. An optional descriptor is one whose inclusion in a transaction results in a valid definition of a more expanded form of its associated managed object. A valid package may exist indefinitely or be of limited duration and may include pay-per-view or impulse pay-per-view purchasability. DNCS components do not supply default values for missing required descriptors.

Descriptors are passed in transactions using descriptor loops. Each loop contains some combination of descriptors to describe the desired state of a single managed object. Many BOSS transactions allow multiple loops, each of which specifies the state of a separate managed object. Each descriptor carries an individually unique combination of fields. Descriptors are positionally interchangeable within a descriptor loop.

The BOSS interface descriptors are defined in ***Section 4, BOSS Descriptors***.

2.2 Reference Architecture

The following diagram illustrates the system-context in which the BOSS interface is defined to operate. Items to the right of the dotted line are Scientific-Atlanta supplied components. Items to the left of the dotted line are non-Scientific-Atlanta products required for proper operation of the BOSS interface.

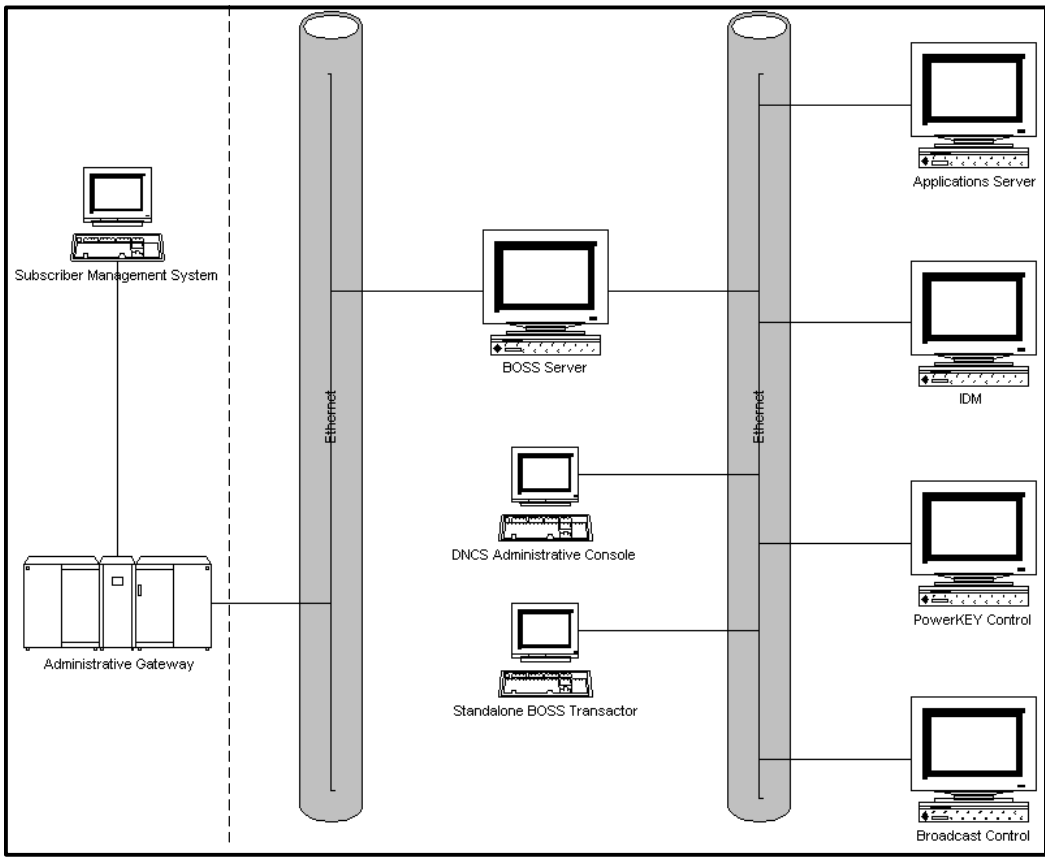


Figure 1-1 -- Reference Architecture

2.2.1 Subscriber Management System

The Subscriber Management System (SMS) is the system by which a service provider defines services to the Administrative Gateway (AG), and manages subscriber authorizations to those services. The interface between the SMS and the AG is not specified in this document.

Note: SMSs are provided by other manufacturers.

2.2.2 Administrative Gateway

The Administrative Gateway (AG) controls access to and modifies data held by DNCS component systems. The AG implements the access network's policies concerning service and authorization provisioning. As an example of such a policy, a given access network may determine that only the VASP that defined a given service may grant or revoke access to it.

When the AG has validated policy compliance, it transmits a BOSS request to the appropriate DNCS component to execute the request.

DNCS components will execute operations received from the Administrative Gateway without further access checks.

Note: Administrative Gateways are provided by other manufacturers.

2.2.3 DNCS Administrative Console

The DNCS Administrative Console provides a Graphical User Interface (GUI) allowing a DNCS operator to provision, control, and monitor DNCS components and other components of the Scientific-Atlanta Digital Broadband Delivery System (DBDS). These components may include digital headend equipment, PowerKEY™ equipment, and digital home communications terminals (DHCTs). Additional, the Administrative Console allows for the configuration of communications paths to external equipment.

2.2.4 Administrator Gateway Emulator

The Administrative Gateway Emulator (AGE) provides a Graphical User Interface (GUI) allowing a DNCS operator to directly enter BOSS transactions and view the results. The AGE is used in installations lacking an Administrative Gateway system. One example of such an installation would be one in which the Administrative Gateway is a legacy system and has not yet been upgraded to process the BOSS interface.

2.2.5 BOSS Server

The BOSS Server provides a mechanism for the routing of a BOSS request to the appropriate DNCS component and BOSS response to the BOSS client that initiated the request.

2.2.6 Broadcast Control Suite

The Broadcast Control Suite (BCS) works in conjunction with the Administrative Gateway for the definition of broadcast-based services supplied via the Scientific-Atlanta Digital Broadband Delivery System.

The Broadcast Control Suite is responsible for the definition of bandwidth segments. The bandwidth segment is an abstraction of a session over some particular time, which may be unlimited. The BCS manages the higher-level segment, but not the underlying session carrying the segment. Management of sessions is the responsibility of other DNCS components. The segment provides an interface for scheduling events, an interface for scheduling encryption status of a segment, and the basis for forming segment offerings.

2.2.7 PowerKEY™ Control Suite

The PowerKEY™ Control Suite is the component of the DNCS responsible for the encryption, key management, and other conditional access functions of the PowerKEY™ system.

The Conditional Access Manager (CAM) is a component of the PowerKEY™ Control Suite. The CAM manages end-user access to services supplied via the DBDS through the use of packages.

The package is the unit of authorization on which the CAM grants permission to use the defined segments. A given package may contain any number of segments. When a user is granted authorization (permission) for the package, that user has the right to use any of the segments contained in the package.

The CAM, in conjunction with the BCS, provides conditional access services for digital broadcast networks, and manages bandwidth, provisioning, and encryption keys for the service encryptor.

2.2.8 Inventory & Directory Manager

The Inventory & Directory Manager (IDM) stores widely useful information about entities in the Digital Network Control System (DNCS). It makes this information available to entities throughout the DNCS. As examples of its contents, the IDM will hold DHCT MAC address to IP address mappings, distinguished name to IP address mappings, and public cryptographic key certificates for Content Providers and DHCTs.

2.2.9 Applications Control Suite

The Application Control Suite (ACS) is the component of the DNCS responsible for management of *services* and *display channel maps* in the DNCS.

2.2.10 Applications Server

One or more Applications Servers will be present in a typical network to manage and transmit data required by client applications running on DHCTs. For example, an Impulse Pay-Per-View (IPPV) Applications Server might be used to provide IPPV event schedule information to allow a user of a DHCT to select an IPPV event for purchase.

Transactions from the SMS to Applications Servers are not specified by the BOSS Interface. However, the BOSS Server can route these transactions to Applications Servers supplied by Scientific-Atlanta.

2.3 BOSS Interface Managed Objects

The BOSS Interface provides remote access to several categories of objects managed by DNCS components. In some cases these objects are managed by a single such

component, and in others management responsibility is divided over several of the components. In either case, it is correct to think of these managed objects as single distributed entities.

In general, the Inventory & Directory Manager is tasked with basic management of Service Providers, DHCTs, and DHCT types, including network addresses and public encryption key certificates. The Broadcast Control Suite is responsible for maintaining the definitions of the Digital Broadcast Services provided by those Value Added Service Providers, or VASPs, and for bandwidth allocation for these services. The PowerKEY™ Control Suite (specifically the CAM) is responsible for maintaining DHCT package authorizations for services provided by VASPs and managed by the BCS.

Also, when no update transaction is specifically noted for a managed object, attributes of previously defined managed objects are modified using the same transaction which was used to originally define object. For example, a DefineSegment transaction is used to modify the attributes of a service previously defined to the BCS with a DefineSegment transaction.

2.3.1 VASPs

Each VASP object represents the DNCS' store of knowledge concerning a single Value Added Service Provider.

The Inventory & Directory Manager is considered the authority concerning the existence of a VASP. In addition to the VASP's name, network addresses, and public key certificate, the Inventory & Directory Manager maintains a status for the VASP.

2.3.2 DHCTs, DHCT Types

Each DHCT object represents the DNCS' store of knowledge concerning a single DHCT.

The Inventory & Directory Manager is considered the authority concerning the existence of a DHCT. In addition to the DHCT's name, network addresses, and public key certificate, the Inventory & Directory Manager maintains the DHCT's Bootterm page assignments and DHCT Type. DHCT types control the ability of DHCTs to gain access to the DNCS modules. A DHCT registered with an invalid DHCT type will be rejected. A DHCT attempting to boot with an unregistered DHCT type will not be allowed to boot.

The CAM maintains status indications of DHCT digital multicast service enable, interactive service enable, analog service enable, and IPPV purchase authorization parameters, as well the list of package authorizations that have been granted to the DHCT.

DHCT Types are defined and used through the Inventory & Directory Manager. They provide a convenient way to specify Bootterm pages according to different models of DHCTs.

2.3.3 Source IDs and Sources

Every broadcast service to be delivered over the network, whether secure or non-secure, must be associated with a Source ID that uniquely identifies the source, or originator, of the service. For example, HBO/E has a different Source ID than HBO/W. The BCS maintains a table of all valid Source IDs.

The analog or digital bandwidth over which every segment will be delivered is described by a source definition. The BCS maintains a table of all defined sources. Note that multiple sources (with the same Source ID) may be defined when the same service is carried over bandwidth allocated uniquely for different hubs within the same network. In this case, the combination of Source ID, effective time, and Hub ID will uniquely identify a source.

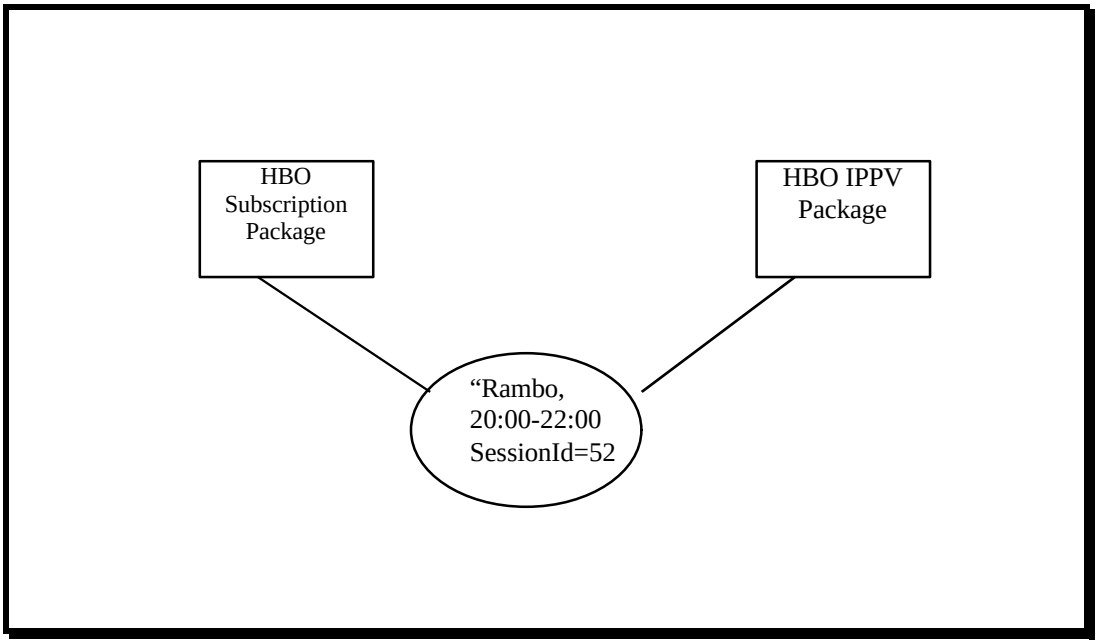
2.3.4 Segments

Secure delivery of services requires that some CA information be imbedded in the bandwidth segments for those services. In order to provide this capability, the BCS must know which bandwidth each service will be carried over, and during what time interval each service will be valid. Segments are defined in terms of bandwidth over an interval of time.

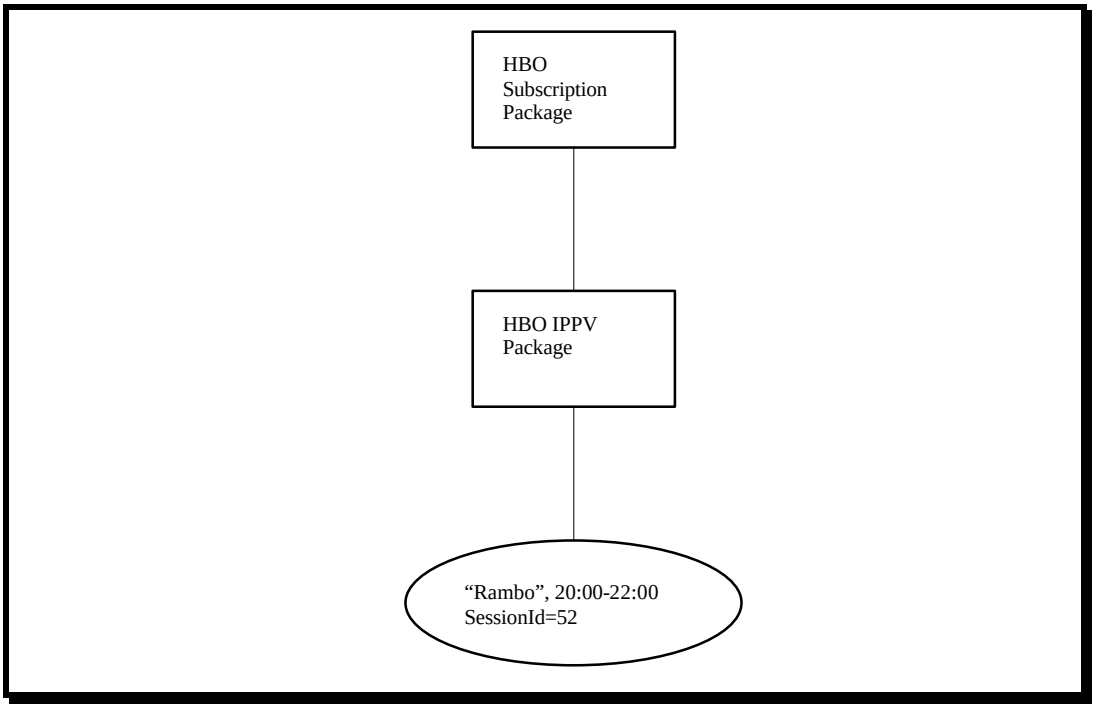
2.3.5 Packages

The CAM component of the PowerKEY™ Control Suite provides CA for bandwidth segments via purchasable units called packages. A package contains one or more segments, and each segment may belong to one or more packages. The CAM grants permission to use segments on a per-package basis. If a package contains multiple segments, and the CAM grants a DHCT permission for the package, then the DHCT may access any of the segments in the package.

Packages and segments are organized as a tree structure in which segments represent leaves and packages represent all other nodes. Packages may contain other packages. This has the effect of adding segments from the contained package to the membership of the containing package. When a user attempts to use a segment, the CA attributes for the package via which the user gains permission to use the segment apply. For example, the following diagram shows a segment belonging to two packages:



The next diagram shows the same segment, belonging to a package which itself belongs to another package:



Conditional access information is specified for each package, and this CA information is used in the construction of Entitlement Control Messages (ECMs) which are distributed to the network within the bandwidth of the defined segments.

In either of the two scenarios, the segment is considered a member of two packages, and appropriate conditional access information for each of the two packages will be included in the ECM stream for the segment. A user who has purchased either of the packages will be able to access the segment, subject to the CA information defined for the package.

The CAM is considered the authority on the existence and definition of packages.

2.3.6 Authorizations

The CAM is also considered the authority on the existence and definition of package authorizations. The CAM maintains authorizations on a per-DHCT basis. For each DHCT, a list of packages for which the DHCT is authorized is maintained.

3. BOSS Interface Coding Standards

BOSS transactions, descriptors, and fields are specified in this document using the following coding standards:

- All names of transactions, descriptors, or fields shall consist of single words composed from the following character set: upper case letters A-Z, lower case letters a-z, digits 0-9, and underscore.
- Names of transactions and descriptors shall begin with an upper case letter A-Z.
- Names of unitary data fields shall begin with a lower case letter a-z.
- Succeeding words of all names shall begin with an upper case letter A-Z.
- Succeeding letters in all words of all names shall be lower case a-z.
- In examples, literal character strings are enclosed in quotation marks.

Arrays in BOSS transactions are denoted using one of the following notations:

- The <> notation indicates that the array is variable in size, up to the maximum specified in the length field.
- The [] notation indicates that the array is fixed in size, and must be sent at the size specified in the length field.

In either case, a string field is assumed to be terminated with an ASCII null character (0x00). In the case of a fixed-length string array, the characters after the null may be padded as either nulls or spaces. Fixed-length opaque arrays are not processed by DNCS, and the rules for padding of such arrays are left to the AG.

4. BOSS Descriptors

In the following sections, each BOSS operation includes example request parameters and response values.

4.1 BOSS Interface Descriptor Summary

The table in this section (Table 4-1) provides a summary of all valid BOSS descriptors. It includes all fields in each descriptor, the types of the fields, an indication of the valid range of the field, a length for character strings, and, in some cases, an explanatory comment.

Note that the null terminator is NOT included in the lengths specified in this table.

A cross-index of valid descriptors per BOSS transaction can be found in Table 5-1.

Table 4-1

AllowExtension		Sent in the DefinePackage transaction to indicate that the authorization for a PPV or IPPV associated with the package may be extended in length.		
Field	*Type*	*Range*	*Length*	*Comments*
allowExtension	enum	AE_True AE_False		If true, allows the ExtendEvent transaction to extend the authorization for the event.
ApplicationUrl		Sent in the DefineService transaction to indicate the resource location at which the samClient running on the DHCT can locate the application client code for the service.		
Field	*Type*	*Range*	*Length*	*Comments*
applicationUrl	string<>		100	Location of the applicationClient executable.
BillingId		Sent to the IDM to identify the billing system that owns a DHCT.		
Field	*Type*	*Range*	*Length*	*Comments*
billingId	unsigned integer	0..2^16-1		
BootPage		Carries the name of a bootterm page.		
Field	*Type*	*Range*	*Length*	*Comments*
boottermPageName	string<>		20	
BoottermPageInfo		Sent to the Inventory & Directory Manager to inform it of Bootterm Page profile information to be associated with a given Bootterm Page.		
Field	*Type*	*Range*	*Length*	*Comments*
pageContents	opaque<>		512	
ChannelId		Sent to identify the analog bandwidth over which the segment described by the current descriptor loop is being carried.		
Field	*Type*	*Range*	*Length*	*Comments*

channelId	unsigned integer	0..255		The SM-20 presently supports a 10-bit channelId, but restricts the number represented to be less than 256.
DatabaseOperation		Sent in the DefineService transaction to indicate whether a strict insertion is desired or a strict update. If this descriptor is omitted from a transaction, the appropriate database operation will be determined by the server from the state of the server's database.		
Field	*Type*	*Range*	*Length*	*Comments*
databaseOperation	enum	do_InsertOnly do_UpdateOnly		
DhctAdminStatus		Communicates a DHCT's administrative status.		
Field	*Type*	*Range*	*Length*	*Comments*
dhctAdminStatus	enum	DAS_Deployed DAS_InService OneWay DAS_InService TwoWay DAS_OutOfService		
DhctMacAddr		Sent to identify the DHCT being described by the current descriptor loop.		
Field	*Type*	*Range*	*Length*	*Comments*
dhctMacAddr	string[]		17	
DhctMacAddrRep		Sent in the ReplaceDhct transaction to indicate the mac address of the DHCT that is replacing an existing DHCT.		
Field	*Type*	*Range*	*Length*	*Comments*
dhctMacAddr	string[]		17	
DhctOpnStatus		Communicates a DHCT's operational status. This read-only field is included in the response to a FetchDhct transaction.		
Field	*Type*	*Range*	*Length*	*Comments*
dhctOpnStatus	enum	DOS_Active DOS_DsmccBootFailed DOS_Initialized DOS_InitializationFailed DOS_Unknown		
DhctseSerialNumber		Sent in the RegisterDhct transaction to identify the internal secure element (DHCTSE) installed in the DHCT.		
Field	*Type*	*Range*	*Length*	*Comments*
dhctseSerialNumber	string[]		17	
DhctState		Sent to the CAM to provision various conditional access related state parameters for a given DHCT.		
Field	*Type*	*Range*	*Length*	*Comments*

dmsEnableFlag	enum	DMS_Enabled DMS_Disabled		
disEnableFlag	enum	DIS_Enabled DIS_Disabled		
analogEnableFlag	enum	AEF_Enabled AEF_Disabled		
ippvEnableFlag	enum	IEF_Enabled IEF_Disabled		
maxIppvEvents	unsigned integer	0..~~255~~72		
creditLimit	unsigned integer	0..2^16-1		The units for this field must be consistent with those used to specify the modeCost field in the ImpulsePayPerView descriptor.
pinEnable	enum	PE_Enabled PE_Disabled		
pin	string<>		8	The pin may be any character string up to 8 characters in length.
fastRefreshFlag	enum	FRF_Enabled FRF_Disabled		
locationX	unsigned integer	0..2^8-1		
locationY	unsigned integer	0..2^8-1		
DhctType		Sent to the Inventory & Directory Manager to communicate the type of a DHCT.		
Field	*Type*	*Range*	*Length*	*Comments*
dhctType	unsigned integer	0..2^16-1		
dhctTypeRevision	unsigned integer	0..2^16-1		
organizationalUniqueId	opaque[]		3	
DhctTypeInfo		Sent to the Inventory & Directory Manager to inform it of DHCT Type profile information to be associated with a given DHCT Type.		
Field	*Type*	*Range*	*Length*	*Comments*
dhctTypeDescription	string<>		80	
dhctVendor	string<>		80	
DisplayChannelMap		Sent in the DefineDisplayChannelMap transaction to associate a group of services with identification numbers that are presented to home DHCT users as channel numbers. The map is in the form of an array of serviceIds such that the array is indexed by the Display Channel Number.		
Field	*Type*	*Range*	*Length*	*Comments*
displayChannelCount	unsigned integer	1..1999		Number of entries in the serviceIdList.
serviceIdList	unsigned integer<>			The list of servicedIds implementing the DisplayChannelMap. The index into this array is assumed to be the DisplayChannelNumber.
DisplayChannelNumberList ~~(apportioned)~~		Sent in the AcceptSubstitutionUnit transaction to specify acceptance of a subset of a Substitution Unit.		
Field	*Type*	*Range*	*Length*	*Comments*
displayChannelNumberCount	unsigned integer			

displayChannelNumberList	unsigned integer<>			This is an XDR array.
DsmccSessionId		Used in segment definitions to identify digital service bandwidth.		
Field	*Type*	*Range*	*Length*	*Comments*
dsmccSessionId	opaque[]		10	Defined by DSM-CC to consist of a mac address followed by a 4-byte unsigned integer. No check for this format is performed on this field by DNCS.
EffectiveTime		Sent in the DefineSource and DefineSegment transactions to determine when the definitions are to be applied.		
Field	*Type*	*Range*	*Length*	*Comments*
effectiveTime	unsigned integer	0..2^16-1		UNIX time
EntitlementId		Provided in response to a DefinePackage transaction to represent the authorization token given to a package		
Field	*Type*	*Range*	*Length*	*Comments*
entitlementId	unsigned integer			
EventExtension		Sent in the *ExtendEvent* transaction to specify the length of time (in minutes) that the event should be extended.		
Field	*Type*	*Range*	*Length*	*Comments*
eventExtension	unsigned integer			Minutes
ExpirationTime ~~(apportioned)~~		Sent in the AcceptSubstitutionUnit transaction and the ExtendSubstitutionUnit transaction to specify the end time of an accepted Substitution Unit.		
Field	*Type*	*Range*	*Length*	*Comments*
expirationTime	Unsigned integer			UNIX time.
HeadEnd		Provides a mechanism for assigning a DHCT to a head-end.		
Field	*Type*	*Range*	*Length*	*Comments*
headEndName	string<>		20	
HostId		Sent to identify the Host device being described by the current descriptor loop.		
Field	*Type*	*Range*	*Length*	*Comments*
hostId	string[]		10	
HostIdList		Sent to identify the set of Host devices being described by the current descriptor loop.		
Field	*Type*	*Range*	*Length*	*Comments*
hostIdCount	unsigned integer			
hostId<>	string[]		10	This is an XDR array of hostIds.
HubId		Provides a mechanism for identifying hubs within headends.		
Field	*Type*	*Range*	*Length*	*Comments*
hubId	unsigned integer	0..2^32-1		
HubIdList ~~(apportioned)~~		Sent in the DefineLineUpGroup transaction to enumerate the set of hubs having the same DisplayChannelMap.		

Field	*Type*	*Range*	*Length*	*Comments*
hubIdCount	unsigned integer			
hubIdList	unsigned integer<>			This is an XDR array of hubIds.
ImpulsePayPerView		Sent in the DefinePackage transaction to provide additional definition for packages that are to be purchasable via the DHCT on a per-use basis.		
Field	*Type*	*Range*	*Length*	*Comments*
freeCaStartTime	unsigned integer			UNIX time
freeCaInterval	unsigned integer			Seconds
buyWindowStartTime	unsigned integer			UNIX time
buyWindowInterval	unsigned integer			Seconds
allowWindowCancel	enum	AWC_True AWC_False		
cancelWindowInterval	unsigned integer			Seconds from start of package.
allowImmediateCancel	enum	AIC_True AIC_False		
modeCount	unsigned integer	1..5		Number of repetitions of modeLength, modeCost, and modeRightToCopy
modeLength	unsigned integer			Minutes
modeCost	unsigned integer	0..2^16-1		Units are determined during installation configuration, by the CAM customer. These units must be consistent with the creditLimit in the DhctState descriptor.
modeRightToCopy	enum	MRTC_CopyPermitted MRTC_CopyProhibited		
IpAddress		Sent to the Inventory & Directory Manager to provide the IP address for either DHCT or Service Provider.		
Field	*Type*	*Range*	*Length*	*Comments*
ipAddress	string<>		16	IP addresses shall be represented in Internet-traditional dotted decimal format.
IppvArchivePath		Sent in response to an IppvUploadComplete transaction to specify the full path name of the IPPV Archive file created..		
Field	*Type*	*Range*	*Length*	*Comments*
ippvArchivePath	string<>		256	Full path name of IPPV Archive file.
IppvPurchase		Sent in the response to an IppvPoll transaction to describe the characteristics of a purchased IPPV event.		
Field	*Type*	*Range*	*Length*	*Comments*
purchaseTime	unsigned integer			UNIX time
purchaseLength	unsigned integer			minutes
purchaseCost	unsigned integer	0..255		Units are determined during installation configuration, by the CAM customer. These units must be consistent with the creditLimit in the DhctState descriptor.

purchaseRightToCopy	enum	PRTC_CopyPermitted PRTC_CopyProhibited		
IppvUploadFile		Sent in an InitializeIppvUpload transaction to specify the name of the IPPV Upload file to be created..		
Field	*Type*	*Range*	*Length*	*Comments*
ippvUploadFile	string<>		80	IPPV Upload file name.
IppvUploadPath		Sent in response to an InitializeIppvUpload transaction to specify the full path name of the IPPV Upload file created..		
Field	*Type*	*Range*	*Length*	*Comments*
ippvUploadPath	string<>		256	Full path name of IPPV Upload file.
KeyCertificate		Sent to specify a public key certificate assigned to a network entity.		
Field	*Type*	*Range*	*Length*	*Comments*
keyCertificate	opaque<>		1024	
LimitedDuration		Sent in the DefineSegment transaction to specify the interval over which a segment is active and in the DefinePackage transaction to specify the interval over which a package is to be authorized.		
Field	*Type*	*Range*	*Length*	*Comments*
startTime	unsigned integer			UNIX time.
duration	unsigned integer			Minutes.
LineUpGroupId ~~(apportioned)~~		Sent in many transactions to identify the Line Up Group to which the transaction applies.		
Field	*Type*	*Range*	*Length*	*Comments*
lineUpGroupId	unsigned integer	0…2^16-1		
LineUpGroupIdList ~~(apportioned)~~		Sent in many transactions to identify the Line Up Groups to which the transaction applies.		
Field	*Type*	*Range*	*Length*	*Comments*
lineUpGroupIdCount	unsigned integer			
lineUpGroupIdList	unsigned integer<>			
LineUpGroupName ~~(apportioned)~~		Sent in the DefineLineUpGroup transaction to provide an optional name.		
Field	*Type*	*Range*	*Length*	*Comments*
lineUpGroupName	string<>		20	
Logo		Sent in the DefineService transaction to provide a graphical icon for the service being defined.		
Field	*Type*	*Range*	*Length*	*Comments*
logo	opaque<>		1024	
LongDescription		Sent in the DefineService transaction to provide a longer description of the service being defined.		
Field	*Type*	*Range*	*Length*	*Comments*

longDescription	string<>		32	
MibCommunityString ~~(apportioned)~~		Sent in the GetDhctDiagnostics transaction to allow access to MIB data.		
Field	*Type*	*Range*	*Length*	*Comments*
mibCommunityString	string<>			
MibObjectId ~~(apportioned)~~		Globally unique for all space and time, OIDs are a sequence of non-negative integers organized heirarchically like UNIX or PC-DOS file system names.		
Field	*Type*	*Range*	*Length*	*Comments*
mibObjectId	string<>		127	Example OID: "1.2.6.3.1429.1"
MibObjectIdList ~~(apportioned)~~				
Field	*Type*	*Range*	*Length*	*Comments*
mibObjectIdCount	unsigned integer	1..8		
mibObjectIdList	string<>			This is an XDR array of MibObjectIds
MibObjectList ~~(apportioned)~~				
Field	*Type*	*Range*	*Length*	*Comments*
mibObjectCount	unsigned integer			
mibObjectId	string<>		127	
mibObjectValue	string<>			
MibQuery Mode ~~(apportioned)~~		Sent in the GetDhctDiagnostics transaction to indicate whether the data at the specified mibObjectId or the data at the mibObjectId after the specified mibObjectId is to be retrieved.		
Field	*Type*	*Range*	*Length*	*Comments*
mibQueryMode	enum	MQM_Current MQM_Next		
NonRespondingArchivePath (apportioned)		Sent in response to an NonRespondingComplete transaction to specify the full path name of the non-responding archive file created..		
Field	*Type*	*Range*	*Length*	*Comments*
nonRespondingArchivePath	string<>		256	Full path name of non-responding archive file.
NonRespondingDuration (apportioned)		Sent in the InitializeNonRespondingUpload transaction to specify the criteria to be used to select records to be included in the non-responding upload file. DHCTs that have not transmitted a reverse path message in NonRespondingDuration hours will be included.		
Field	*Type*	*Range*	*Length*	*Comments*
nonRespondingDuration	unsigned integer			Hours.
NonRespondingFile (apportioned)		Sent in an InitializeNonRespondingUpload transaction to specify the name of the non-responding upload file to be created..		
Field	*Type*	*Range*	*Length*	*Comments*

nonRespondingFile	string<>		80	Non-responding upload file name.
NonRespondingPath (apportioned)		Sent in response to an InitializeNonRespondingUpload transaction to specify the full path name of the non-responding upload file created..		
Field	*Type*	*Range*	*Length*	*Comments*
nonRespondingPath	string<>		256	Full path name of non-responding upload file.
NsapAddress		Carries an OSI NSAP address for use in either DHCT or Service Provider definition.		
Field	*Type*	*Range*	*Length*	*Comments*
nsapAddr	opaque<>		20	
OrderSource (apportioned)		Sent in the AddRppvOrder and AddRppvCancel transactions to identify the originator of a RPPV authorization.		
Field	*Type*	*Range*	*Length*	*Comments*
orderSource	unsigned integer			
OrderTime (apportioned)		Sent in the AddRppvOrder and AddRppvCancel transactions to identify the time at which the order was added or removed.		
Field	*Type*	*Range*	*Length*	*Comments*
orderTimee	unsigned integer			
PackageAuthorization		Used to inform the CAM of authorizations for given packages.		
Field	*Type*	*Range*	*Length*	*Comments*
packageListType	enum	PLT_AllPackages PLT_SubscriptionOnly PLT_PpvOnly		
packageCount	unsigned integer			
packageNameList	string<>		20	This is an XDR array of packageNames.
PackageMembership		Sent to identify the constituents of the package being described by the current descriptor loop.		
Field	*Type*	*Range*	*Length*	*Comments*
segmentCount	unsigned integer			
segmentNameList	string<>		20	This is an XDR array of segment names .
packageCount	unsigned integer			
packageNameList	string<>		20	This is an XDR array of packageNames.
PackageName		Sent to identify the package being described by the current descriptor loop.		
Field	*Type*	*Range*	*Length*	*Comments*
packageName	string<>		20	
ParameterNumber		Sent in the DefineService transaction to provide a numerical parameter to the service's application client.		

Field	*Type*	*Range*	*Length*	*Comments*
parameterNumber	unsigned integer			
ParameterString		Sent in the DefineService transaction to provide a string parameter to the service's application client. Application clients may be written to use this parameter directly, or as a URL providing the resource location of the client's parameters.		
Field	*Type*	*Range*	*Length*	*Comments*
parameterString	string<>		100	
PayPerView		Sent in the DefinePackage transaction to indicate that a package is authorized on a reservation pay per view basis.		
Field	*Type*	*Range*	*Length*	*Comments*
rightToCopy	enum	RTC_CopyPermitted RTC_CopyProhibited		
PrimaryVaspNsap		Carries the OSI NSAP address used by the DHCT to contact its primary or owning VASP for "level 2" boot or application download.		
Field	*Type*	*Range*	*Length*	*Comments*
nsapAddr	opaque<>		20	
RecordCount		Sent in the *GetIppvRecords* and *GetNonResponsingRecords* transaction to identify the number of to be retrieved.		
Field	*Type*	*Range*	*Length*	*Comments*
recordCount	unsigned integer	1..50		
RecordType		Sent in the response to an IppvPoll transaction to identify the type of IPPV purchase record being reported.		
Field	*Type*	*Range*	*Length*	*Comments*
recordType	enum	RT_DigitalIppvPurchase RT_DigitalIppvCancel RT_AnalogIppvPurchase RT_AnalogIppvCancel		
Recurring		Sent in the DefineSegment transaction, with the LimitedDuration transaction, to specify a limited duration segment that may run repetitively.		
Field	*Type*	*Range*	*Length*	*Comments*
recurrenceUnit	enum	RU_Seconds RU_Minutes RU_Hours RU_Days RU_Weeks		
recurrenceInterval	unsigned integer			Specified in the units indicated in the recurrenceUnit field.
RefreshAll (apportioned)		Sent in the ModifyDhctConfiguration transaction to indicate that all EMMs are to be refreshed		

Field	*Type*	*Range*	*Length*	*Comments*
refreshAll	enum	RA_True RA_False		!
ResponseTime (apportioned)		Sent in response to a *GetNonRespondingRecords* transaction to indictate the last time a reverse path message was received from a DHCT.		
Field	*Type*	*Range*	*Length*	*Comments*
responseTime	unsigned integer			UNIX time
Result		Sent to encapsulate the result of the corresponding descriptor loop in the transaction.		
Field	*Type*	*Range*	*Length*	*Comments*
resultCode	enum	See text		A complete list of resultCodes can be found in section 4.2.65.1 Valid Result Codes.
SecurityMode		Sent in the DefineSource transaction to indicate whether a particular service is to be encrypted.		
Field	*Type*	*Range*	*Length*	*Comments*
securityMode	enum	SM_Clear SM_Encrypted		
SegmentName		Sent to identify the segment being described by the current descriptor loop.		
Field	*Type*	*Range*	*Length*	*Comments*
segmentName	string<>		20	
SegmentSecurity		Sent to the BCS to define the security control attributes of a new segment, or to modify the security control attributes of an existing segment.		
Field	*Type*	*Range*	*Length*	*Comments*
blackoutActive	enum	BA_True, BA_False BA_Spotlight		
centroidX	unsigned integer	0..2^8-1		Ignored if blackoutActive == BA_False.
centroidY	unsigned integer	0..2^8-1		Ignored if blackoutActive == BA_False.
radius	unsigned integer	0..2^8-1		Ignored if blackoutActive == BA_False.
fingerprintEnable	enum	FPE_Disable FPE_Visible FPE_Invisible		
copyProtectionLevel	enum	CPL_CopyPermitted CPL_CopyProhibited CPL_Copy_Purchasable		
copyProtectionMode	unsigned integer			
ServiceId		Returned from the DefineService transaction as a unique identifier for a service. The serviceId may be used to select the parameters for a given service when an application is written to send the parameters for all its		

		supported services in a single file. The ServiceId can also be used in the QueryService and RetireService transactions.		
Field	*Type*	*Range*	*Length*	*Comments*
serviceId	unsigned integer			
ServiceName		Sent in the DefineService transaction to provide a unique identification of the service being defined.		
Field	*Type*	*Range*	*Length*	*Comments*
serviceName	string<>		20	
ShortDescription		Sent in the DefineService transaction to provide a short description of the service being defined.		
Field	*Type*	*Range*	*Length*	*Comments*
shortDescription	string<>		5	
SoftwareTOC		Sent to the Inventory & Directory Manager to define the Software Table-of-Contents for a DHCTor DHCT Type.		
Field	*Type*	*Range*	*Length*	*Comments*
softwareTOC	string<>		80	
SourceId		Sent in the DefineSegment transaction to identify the originating source of the segment being defined.		
Field	*Type*	*Range*	*Length*	*Comments*
sourceId	unsigned integer	1..2^16-1		
SourceName		Sent to provide a human-readable name for the source being defined.		
Field	*Type*	*Range*	*Length*	*Comments*
sourceName	string<>		20	
SplitChannelMap		Sent in the DefineDisplayChannelMap transaction to provide a list of channels which can carry different services depending on the time of day.		
Field	*Type*	*Range*	*Length*	*Comments*
splitChannelCount	unsigned integer	0..8		Defines the number of iterations of the remaining fields in this descriptor.
displayChannelNumber	unsigned integer	1..1999		Identifies the entry in the Display Channel Map.
serviceIdA	unsigned integer			Service for first time period. This service runs from serviceAStartTime until serviceBStartTime each day.
serviceAStartTime	unsigned integer	0..86400		Start time of service A, specified as an offset in seconds from midnight.
serviceIdB	unsigned integer			Service for second time period. This service runs from serviceBStartTime until serviceAStartTime each day.
serviceBStartTime	unsigned integer	0..86400		Start time of service B, specified as an offset in seconds from midnight.
StartRecordNumber		Sent in the GetIppvRecords transaction to identify the first IPPV purchase records to be retrieved.		
Field	*Type*	*Range*	*Length*	*Comments*
startRecordNumber	unsigned integer			

StartTime ~~(apportioned)~~		Sent in the DefineSubstitutionUnit transaction to indicate the time at which the substitution unit will take effect, if accepted.		
Field	*Type*	*Range*	*Length*	*Comments*
startTime	unsigned integer			UNIX time.
SubstitutionUnit ~~(apportioned)~~		Sent in the DefineSubstitutionUnit transaction to provide a set of old/new channel substitutions to be effected as a unit.		
Field	*Type*	*Range*	*Length*	*Comments*
substitutionCount	unsigned integer			
oldDcnList	unsigned integer<>	Each DCN must be in the range 1..1999		This is an XDR array, with *substitutionCount* elements.
newDcnList	unsigned integer<>	Each DCN must be in the range 1..1999		This is an XDR array, with *substitutionCount* elements. The elements of this array are assumed to match positionally with those in oldDcnList.
SubstitutionUnitId ~~(apportioned)~~		Sent in many transactions to identify the Substitution Unit to which the transaction applies.		
Field	*Type*	*Range*	*Length*	*Comments*
substitutionUnitId	unsigned integer			
TimeOffset		Sent to the Inventory & Directory Manager to indicate the time-of-day information for a DHCT..		
Field	*Type*	*Range*	*Length*	*Comments*
offsetMinutes	signed integer			
daylightSaving	Enum	DS_True DS_False		
UploadFileSize		Sent in response to an *InitializeIppvUpload* or *InitializeNonRespondingUpload* transaction to specify the number of records in the upload file created..		
Field	*Type*	*Range*	*Length*	*Comments*
uploadFileSize	unsigned integer			number or records
VaspId		Carries a Service Provider identifier.		
Field	*Type*	*Range*	*Length*	*Comments*
vaspId	unsigned integer	0..2^32-1		
VaspInfo		Sent to the Inventory & Directory Manager to inform it of VASP profile information to be associated with a given VASP.		
Field	*Type*	*Range*	*Length*	*Comments*
vaspName	string<>		80	
vaspIpAddress	string<>		16	
vaspStatus	enum	VS_InService VS_OutOfService		

4.2 BOSS Interface Descriptor Definitions

In definitions for the following descriptors, these rules generally apply:

In the remote procedure calls, a key certificate is represented as a variable-length array of untyped bytes. A zero-length array represents a null or absent certificate.

An IP address is represented as a null-terminated character string containing the dotted-decimal rendition of the address. A null (zero-length) character string represents a null or missing address.

DHCT type and revision are represented as unsigned short integers. Zero values indicate null or missing values for either type or revision. In the face of a null or unknown DHCT type, the DHCT type revision shall be ignored.

An NSAP address is represented as a variable-length array of untyped data bytes. A zero-length array represents a null or missing NSAP address.

4.2.1 AllowExtension

The AllowExtension is sent to indicate to the CAM that the authorization for a package associated with a PPV or IPPV event may be extended in length. Because live events, such as sporting events, are not strictly time controlled, it is necessary to implement a mechanism to extend the authorization for those events. The AllowExtension descriptor provides a method to allow or restrict such extension.

4.2.2 ApplicationUrl

The ApplicationUrl descriptor is sent in the DefineService transaction to indicate the resource location at which the samClient running on the DHCT can locate the application client code for the service.

4.2.3 Bandwidth Identification

Because ECMs are constrained to be sent in the bandwidth carrying the packages' member segments, the BCS requires a mechanism for identification of that bandwidth. It is possible to identify bandwidth in three ways. BOSS specifies a separate descriptor for each mechanism, and each source definition received by the BCS (in a DefineSource transaction) is required to carry one and only one of the three descriptors.

The DsmccSessionId descriptor may be sent to define sources against bandwidth in the digital network for which bandwidth was not reserved in advance. The dsmccSessionId field refers to a DSM-CC session. DsmccSessionIds for DSM-CC sessions are provided to the BCS from the AG. The coordination of DsmccSessionIds between servers providing content and the AG is a requirement for the AG.

The ChannelId descriptor may be sent to define sources against bandwidth in the analog network.

4.2.3.1 DsmccSessionId

Another mechanism for identification of digital bandwidth is the DSM-CC Session ID (SID), which is required by DSM-CC for any bandwidth source. The DsmccSessionId

descriptor is used in source definitions to identify digital service bandwidth only in the case where DNCS bandwidth reservation is not being used.

4.2.3.2 ChannelId

It is possible to define a source to the BCS which is carried over analog bandwidth. The ChannelId descriptor is sent to identify the analog bandwidth over which the source described by the current descriptor loop is being carried. Bandwidth segments that originate from such a source may be included in packages in the same manner as digital segments.

While the CAM will properly authorize the analog sources, it relies on the Scientific-Atlanta System Manager 20 (SM-20) to control the bandwidth and security for the analog network. Therefore, the channelId specified in this descriptor must be coordinated with the channelIds known to the SM-20. The responsibility for this coordination is assigned to the AG.

4.2.4 BillingId

The BillingId is used to identify a DHCT's "owning" billing system in a configuration where multiple billing systems are being used with a single DNCS. The BillingId can be used to limit the contents of an IPPV upload file to only puchases that were made by DHCTs "owned" by a particular billing system.

4.2.5 BootPage

The BootPage carries the name of a bootterm page. One descriptor of this type may be present in DHCT or DHCT Type definition, indicating a bootterm page to be used with the DHCT(s) in question.

4.2.6 BoottermPageInfo

The BoottermPageInfo descriptor is sent to the Inventory & Directory Manager to inform it of Bootterm Page profile information to be associated with a given Bootterm Page. It is also sent by the Inventory & Directory Manager to report the current values of the profile information.

4.2.7 ChannelId

See Section 4.2.3.2 for a description of the ChannelId descriptor.

4.2.8 DatabaseOperation

The DatabaseOperation is sent in the DefineService transaction to indicate whether a strict insertion is desired or a strict update. If this descriptor is omitted from a transaction, the appropriate database operation will be determined by the server from the state of the server's database.

4.2.9 DhctAdminStatus

The DhctAdminStatus descriptor communicates a DHCT's administrative status. The administrative status can take one of three values:

Status	Description
1	*Out of Service*, indicating that the DHCT shall not be booted. The DNCS will fail MAC-level DHCT verification and it will not honor DSM-CC UNConfig requests involving DHCTs in the *Out of Service* state.
2	*In Service-Two Way*, indicating that, administratively, the DHCT is in service and operational. Note that this does not imply that the physical DHCT is active on the network. It does indicate that the DNCS DNCS will generate conditional access messages to the DHCT and will honor MAC-level verification and DSM-CC UNConfig operations on the DHCT.
3	*Deployment*. The DHCT is being deployed or has recently been deployed into the network. The DNCS will honor MAC-level verification and DSM-CC UNConfig operations on the DHCT.
4	*In Service-One Way*, indicating that, administratively, the DHCT is in service and operational. Note that this does not imply that the physical DHCT is active on the network. It does indicate that the DNCS will generate conditional access messages to the DHCT.

Note that a change in DHCT administrative status from *In Service-Two Way* to *Out of Service* will not terminate any active sessions involving that DHCT. The administrative status is examined only when a DHCT initiates either MAC-level or DSM-CC initialization.

See also the *DhctOpnStatus* descriptor.

4.2.10 DhctMacAddr

The DhctMacAddr descriptor is sent to identify the DHCT being described by the current descriptor loop.

4.2.11 DhctMacAddrRep

The DhctMacAddrRep descriptor is sent in the ReplaceDhct transaction to indicate the MAC address of the DHCT which is replacing an existing DHCT. Its purpose is to distinguish the replacing DHCT from the DHCT being replaced.

4.2.12 DhctOpnStatus

The DhctOpnStatus descriptor communicates a DHCT's operational status. This status is assigned by the DNCS during the normal course of DNCS operation. It can only be read via the BOSS interface. It can not be written via the BOSS interface.

The administrative status can take one of five values:

Status	Description
1	*Unknown*, indicating that the DNCS has no record of contact with the DHCT, or that the DNCS has lost contact with the DHCT. This is the initial operational state for any DHCT added to the IDM.
2	*MAC initialization failed*, indicating that the DNCS has failed to properly initialize the DHCT during MAC-level DHCT verification.
3	*MAC initialized*, indicating that the DHCT has passed MAC-level verification and configuration but has not yet completed DSM-CC configuration.
4	*DSMCC Boot failed*, indicating that the DHCT has passed MAC-level verification and configuration, but the DSM-CC UNConfig exchange has failed.
5	*Active*, indicating that, from the perspective of the DNCS, the DHCT is properly initialized and is operational.

Note that the DNCS will not adjust the operational state of a DHCT that is administratively *Out of Service.* If an *In Service* DHCT is moved to *Out of Service* using a BOSS transaction, the DHCT's operational status will retain its most recent value.

4.2.13 DhctseSerialNumber

The DhctseSerialNumber is sent in the RegisterDhct and UpdateDhct transactions to identify the internal secure element (DHCTSE) installed in the DHCT or the secure element on a smart card plugged into the DHCT.

4.2.14 DhctState

The DhctState descriptor is sent to the CAM to provision various conditional access related state parameters for a given DHCT. It is also sent by the CAM to report the state of those parameters.

The dmsEnableFlag field is set to False to cause the CAM to deprovision all package authorizations on a given DHCT. Unless deleted by inclusion of a PackageAuthorization descriptor, the authorizations are retained in the CAM database, such that resetting the dmsEnableFlag causes the CAM to re-provision the authorizations to the DHCT.

The disEnableFlag field is set to False to cause the CAM to disable the ability of the specified DHCT to request interactive sessions.

The analogEnableFlag field is set to False to cause the CAM to disable the ability of the specified DHCT to utilize analog services.

The ippvEnableFlag is set to False to cause the CAM to disable the ability of the specified DHCT to utilize IPPV services.

The maxIppvEvents field specifies the maximum number of IPPV purchases allowed without being reported to and acknowledged by the CAM. The ability of the specified DHCT to make IPPV purchases can be revoked by setting this field to 0.

The creditLimit field specifies the maximum total cost of stored IPPV purchases. The cost of a given IPPV purchase is figured against this maximum only until the purchase is successfully reported to and acknowledged by the CAM. The creditLimit field is specified without units, but is required to be unit-consistent with the modeCost field in the ImpulsePayPerView descriptor. This allows the owner of a CAM to specify its own units for use with the creditLimit function. The ability of the specified DHCT to make IPPV purchases can be revoked by setting this field to 0.

The pinEnable field is used to manage the capability to require a Personal Identification Number (PIN) to make IPPV purchases. If this field is disabled, the pin field is ignored.

If the pinEnable field is set, the pin field must contain an 8 character PIN.

The fastRefreshFlag is used to cause the CAM to increase the rate at which EMMs are transmitted for the specified DHCT.

The locationX and locationY fields are used to specify the geographic location of the specified DHCT within the network.

4.2.15 DhctType

The DhctType descriptor is sent to the Inventory & Directory Manager to communicate the type of a DHCT. It is also sent by the Inventory & Directory Manager to report the current values of the profile information.

4.2.16 DhctTypeInfo

The DhctTypeInfo descriptor is sent to the Inventory & Directory Manager to inform it of DHCT Type profile information to be associated with a given DHCT Type. It is also sent by the Inventory & Directory Manager to report the current values of the profile information.

4.2.17 DisplayChannelMap

The DisplayChannelMap descriptor is sent in the DefineDisplayChannelMap transaction to associate a group of services with identification numbers that are presented to home DHCT users as channel numbers. The map is in the form of an array of serviceIds such that the array is indexed by the Display Channel Number.

4.2.18 DisplayChannelNumberList

The DisplayChannelNumberList descriptor is sent in the AcceptSubstitutionUnit transaction to specify acceptance of a subset of a Substitution Unit

4.2.19 DsmccSessionId

See Section 4.2.3.1 for a description of the DsmccSessionId descriptor.

4.2.20 EffectiveTime

The EffectiveTime descriptor, specified in UNIX time format, is sent in the DefineSource and DefineSegment transactions to specify the time at which the definition is to be applied.

4.2.21 EntitlementId

The EntitlementId is provided in response to a DefinePackage transaction and is used to allow BOSS clients (such as Applications Servers) to determine the number used by the DNCS to identify packages.

4.2.22 EventExtension

This descriptor is sent in the *ExtendEvent* transaction to specify the length of time (in minutes) that the event is to be extended.

4.2.23 ExpirationTime

This descriptor is sent in the AcceptSubstitutionUnit transaction and the ExtendSubstitutionUnit transaction to specify the end time of an accepted Substitution Unit.

4.2.24 HeadEnd

This descriptor provides a mechanism for naming a single head-end.

4.2.25 HostId

The DhctMacAddr descriptor is sent to identify the Host being described by the current descriptor loop.

4.2.26 HostIdList

This descriptor is sent in the DefineCertificateRevocationList transaction to enumerate the Host devices to be included in the CRL.

4.2.27 HubId

This descriptor provides a mechanism for identifying one of the multiple hubs within a head-end.

4.2.28 HubIdList

This descriptor is sent in the DefineLineUpGroup transaction to enumerate the set of hubs having the same DisplayChannelMap.

4.2.29 ImpulsePayPerView

The ImpulsePayPerView descriptor is sent in the DefinePackage transaction to provide additional definition for packages that are to be purchasable via the DHCT on a per-use basis.

The ImpulsePayPerView descriptor is optional, such that when it is not included in the definition of a package, authorization to view services contained in that package must be provided to the CAM via the ModifyDhctConfiguration transaction.

The freeCaInterval field allows a free preview window to be specified during which services belonging to the package are available to anyone on the network. This field is specified as a duration, in seconds, which runs from the freeCaStartTime, specified in UNIX time format.

The buyWindowInterval field specifies a duration during which time the package may be purchased. This field only applies to package member-services defined with the LimitedDuration descriptor. This field is specified as a duration, in seconds, which runs from the buyWindowStartTime, specified in UNIX time format.

The allowWindowCancel field allows a *cancelWindowInterval* to be specified.

The cancelWindowInterval field specifies a duration, in seconds, during which an IPPV purchase may be canceled. The interval runs from the buyWindowStartTime.

The allowImmediateCancel field allows the possibility that a IPPV purchase may be canceled within a short, PowerKEY-defined duration beginning with the purchase. This cancellation duration is operative even in the event that the purchase was made after the cancelWindowInterval closed.

IPPV Modes represent purchase options for the IPPV package. The modeCount field specifies the number of modes available by which the package may be impulse purchased.

The modeLength specifies an amount of time for which the authorization will be granted, in minutes.

The modeCost represents the charge against the DHCT's creditLimit that will be applied until the IPPV purchase is successfully reported to the CAM and successfully acknowledged by the CAM. The total of the modeCosts of all outstanding IPPV purchases may not exceed the credit limit. The DHCT security element enforces this by refusing to authorize additional IPPV purchases in such situations until valid CAM acknowledgments are received. When the acknowledgments are received, the outstanding purchases are retired by the DHCT security element, lowering the total of the modeCosts below the creditLimit.

The modeRightToCopy field indicates whether, for a given purchase mode, the right to copy the package member services has been purchased. Because copy protection mechanisms are merely managed and not implemented by PowerKEY, if a given installation does not include a copy protection mechanism the enforceable value for this fields MRTC_CopyPermitted.

4.2.30 IpAddress

The IpAddress descriptor is sent to the Inventory & Directory Manager to provide the IP address for either DHCT or Service Provider. It is also sent by the Inventory & Directory Manager to report the current IP address of either DHCT or VASP.

4.2.31 IppvArchivePath

The IppvArchivePath is sent in response to an IppvUploadComplete transaction to specify the full path name of the IPPV Archive file created.

4.2.32 IppvPurchase

The IppvPurchase descriptor is sent by the CAM in response to an IppvPoll transaction to specify the characteristics of a purchased IPPV event..

4.2.33 IppvUploadFile

The IppvUploadFile is sent in an InitializeIppvUpload transaction to specify the name of the IPPV Upload file to be created.

4.2.34 IppvUploadPath

The IppvUploadPath is sent in response to an InitializeIppvUpload transaction to specify the full path name of the IPPV Upload file created.

4.2.35 KeyCertificate

The KeyCertificate descriptor is sent to specify the encryption key certificate assigned to the network entity (such as a DHCT) being described by the current descriptor loop.

4.2.36 LimitedDuration

The LimitedDuration descriptor is sent in the DefineSegment transaction to specify the interval over which a segment is active and in the DefinePackage transaction to indicate the interval over which a package is authorized. The startTime field is specified in UNIX time format, and the duration is specified in minutes.

The LimitedDuration descriptor is optional, such that if it is not included in a segment definition, that segment is assumed to be active as soon as the bandwidth (DSM-CC session) for that segment is established, and will continue to be active until a RetireSegment transaction is successfully processed by the BSM.

4.2.37 LineUpGroupId

The LineUpGroupId descriptor is sent to identify the Line Up Group to which the transaction applies.

4.2.38 LineUpGroupIdList

The LineUpGroupIdList is sent in many transactions to identify the Line Up Groups to which the transaction applies

4.2.39 LineUpGroupName

The LineUpGroupName is sent in the DefineLineUpGroup transaction to provide an optional name field.

4.2.40 Logo

The Logo descriptor is sent in the DefineService transaction to provide a graphical icon for the service being defined.

4.2.41 LongDescription

This descriptor is sent in the DefineService transaction to provide a longer description of the service being defined.

4.2.42 MibCommunityString

The MibCommunityString descriptor is sent in the GetDhctDiagnostics transactions as a password to authorize access to MIB data.

4.2.43 MibObjectId

The MibObjectId descriptor is globally unique for all space and time, MIB OIDs are a sequence of non-negative integers organized heirarchically like UNIX or PC-DOS file system names.

4.2.44 MibObjectIdList

The MibObjectId descriptor is sent in the *GetDhctDiagnostics* transaction to specify the diagnostic variables to be retrieved.

4.2.45 MibObjectList

The MidObjectList descriptor is sent in response to a *GetDhctDiagnostics* transaction and contains an XDR array of MIB Objects.

4.2.46 MibQueryMode

The MibQueryMode descriptor is sent in the GetDhctDiagnostics transaction to indicate whether the data at the specified mibObjectId or the data at the mibObjectId after the specified mibObjectId is to be retrieved.

4.2.47 NonRespondingArchivePath

The NonRespondingArchivePath is sent in response to an NonRespondingUploadComplete transaction to specify the full path name of the non-responding archive file created.

4.2.48 NonRespondingDuration

The NonRespondingDuration descriptor is sent in the InitializeNonRespondingUpload transaction to specify the criteria to be used to select records to be included in the non-responding upload file. DHCTs that have not transmitted a reverse path message in NonRespondingDuration hours will be included.

4.2.49 NonRespondingFile

The NonRespondingFile is sent in an InitializeNonRespondingUpload transaction to specify the name of the non-responding upload file to be created.

4.2.50 NonRespondingPath

The NonRespondingPath is sent in response to an InitializeNonRespondingUpload transaction to specify the full path name of the non-responding upload file created.

4.2.51 NsapAddress

The NsapAddress descriptor carries an OSI NSAP address for use in either DHCT or Service Provider definition. The format and content of the NSAP address is described in Annex J of the DSM-CC specification (ISO/IEC DIS 13818-6).

4.2.52 OrderSource

The OrderSource descriptor is sent in the AddRppvOrder and AddRppvCancel transactions to identify the originator of a RPPV authorization.

4.2.53 OrderTime

The OrderTime descriptor is sent in the AddRppvOrder and AddRppvCancel transactions to identify the time at which the order was added or removed.

4.2.54 PackageAuthorization

The PackageAuthorization descriptor is used to inform the CAM of authorizations for given packages. When used for this purpose, the PackageAuthorization descriptor specifies a replacement string of authorizations.

The packageListType field is used to indicate whether the PackageAuthorization contains only subscription packages, only PPV packages, or both.

The PackageAuthorization descriptor is also sent by the CAM to report the packages that have been authorized.

4.2.55 PackageMembership

The PackageMembership descriptor is sent to identify the constituents of the package being described by the current descriptor loop. Since packages may contain either segments or other packages, either may be specified in this descriptor. Either of the counts may be 0, indicating no constituents of the associated type are present. Both the segmentCount field and packageCount field may simultaneously be 0, representing a package with no membership

In this description, packageName and segmentName represent unitary data items, not descriptors.

4.2.56 PackageName

The PackageName descriptor is sent to identify the package being described by the current descriptor loop.

4.2.57 ParameterNumber

The ParameterNumber descriptor is sent in the DefineService transaction to provide a numerical parameter to the service's application client.

4.2.58 ParameterString

This descriptor is sent in the DefineService transaction to provide a string parameter to the service's application client. Application clients may be written to use this parameter directly, or as a URL providing the resource location of the client's parameters.

4.2.59 PayPerView

The PayPerView descriptor is sent to indicate to the CAM that a given package is used to authorize segments in a reservation-style pay per view mode.

The PayPerView descriptor is optional, such that if it is not included the package authorization is assumed to be either a normal, continuing authorization, or, in the presence of the ImpulsePayPerView descriptor, a DHCT-originated authorization.

4.2.60 PrimaryVaspNsap

The PrimaryVaspNsap descriptor carries an OSI NSAP address for use in DHCT definition. This NSAP address will be used by the DHCT to contact its primary or owning VASP for "level 2" boot or application download.

4.2.61 RecordType

The RecordType descriptor is sent by the CAM in response to an IppvPoll transaction to identify the type of IPPV purchase record being reported.

4.2.62 Recurring

The Recurring descriptor is sent in the DefineSegment transaction, with the LimitedDuration transaction, to specify a limited duration segment that may run repetitively. A recurring segment is assumed to retain its original definition between occurrences unless that definition is modified by a new DefineSegment transaction.

The recurrenceUnit field specifies the units in which the recurrenceInterval is measured. The following table lists the valid values for the recurrenceUnit field:

Unit
Seconds
Minutes
Hours
Days
Weeks

The recurrenceInterval field in this descriptor measures, in the units specified in the recurrenceUnit field, the interval between segment starts. For example, if the segment is to begin twice per hour, the recurrenceUnit field could be set to Minutes, and the recurrenceInterval field would be set to 30. If it is to begin once per week the recurrenceUnit field could be set to Weeks, and the recurrenceInterval field would be set to 1.

The LimitedDuration descriptor specifies the startTime of the first occurrence of the segment, and subsequent recurrences of the segment will begin as near that time as the

recurrenceInterval allows. For example, a segment that begins at 8:00pm on Friday and repeats twice per hour will recur at 8:30pm Friday.

The recurrenceInterval is required to be greater than the duration field specified for the segment in the LimitedDuration descriptor.

4.2.63 RefreshAll

The RefreshAll descriptor is sent in the *ModifyDhctConfiguration* transaction to indicate that all EMMs for the named DHCT are to be transmitted.

This descriptor is an apportioned feature and is not currently implemented in the DNCS,

4.2.64 ResponseTime

The ResponseTime descriptor is sent in response to a *GetNonRespondingRecords* transaction to indictate the last time a reverse path message was received from a DHCT

4.2.65 Result

The Result descriptor is sent to encapsulate the result of the corresponding descriptor loop in the transaction.

4.2.65.1 Valid Result Codes

The following table is an exhaustive list of valid resultCodes:

Result Code	Description	Applicable Transactions
0	No Error	All transactions
3	Bootterm page already exists	RegisterBoottermPage
4	Bootterm page not found	RegisterDhct RegisterDhctType FetchBoottermPage UpdateBoottermPage
5	DHCT not registered	FetchDhct UpdateDhct ModifyDhctAdminStatus ModifyDhctConfiguration ModifyDhctState ReplaceDhct DhctInstantHit QueryDhct AddAuthorizations RemoveAuthorizations IppvPoll ~~AuthorizeExclusiveSession~~ ~~DeauthorizeExclusiveSession~~ GetDhctDiagnostics BootDhct RegisterHost DeregisterHost
6	DHCT previously registered	RegisterDhct

Result Code	Description	Applicable Transactions
7	DHCT Type not found	FetchDhctType UpdateDhctType RegisterDhct UpdateDhct
8	DHCT Type previously registered	RegisterDhctType
10	Effective Time has passed	DefineSource DefineSegment DefineSourceSecurity
11	Event Already Extended	ExtendEvent
12	Event Not Extendible	ExtendEvent
17	Invalid BootPage	RegisterDhct UpdateDhct RegisterDhctType UpdateDhctType RegisterBoottermPage
18	Invalid ChannelId	DefineSource
19	Invalid Credit Limit	ModifyDhctConfiguration ModifyDhctState
20	Invalid DhctAdminStatus	RegisterDhct UpdateDhct ModifyDhctAdminStatus
21	Invalid DHCTSE Serial Number	RegisterDhct UpdateDhct
22	Invalid DhctState	ModifyDhctConfiguration ModifyDhctState
25	Invalid HubId	DefineSource ModifyDhctHubId
26	Invalid ImpulsePayPerView	DefinePackage
27	Invalid IP Address	DHCT and VASP register and update transactions
28	Invalid KeyCertificate	RegisterDhct UpdateDhct RegisterVasp UpdateVaspProfile ModifyDhctConfig ModifyDhctState ReplaceDhct AddAuthorizations RemoveAuthorizations
29	Invalid LimitedDuration	DefinePackage DefineSegment

Result Code	Description	Applicable Transactions
30	Invalid MAC Address	ModifyDhctConfiguration ModifyDhctState ReplaceDhct DhctInstantHit QueryDhct AddAuthorizations RemoveAuthorizations IppvPoll RegisterDhct DeregisterDhct UpdateDhct ModifyDhctAdminStatus FetchDhct ~~AuthorizeExclusiveSession~~ ~~DeauthorizeExclusiveSession~~ GetDhctDiagnostics BootDhct RegisterHost DeregisterHost
31	Invalid Max IPPV Events	ModifyDhctConfiguration ModifyDhctState
32	Invalid NSAPAddress	DHCT and VASP register, fetch, and update transactions
33	Invalid Package Authorization	ModifyDhctConfiguration
34	Invalid Package Membership	DefinePackage
35	Invalid Package Name	DefinePackage QueryPackage DeletePackage
36	Invalid PayPerView	DefinePackage
37	Invalid Recurring	DefineSegment
39	Invalid SegmentSecurity	DefineSegment
40	Invalid Segment Name	DefineSegment QuerySegment RetireSegment
41	Invalid Source Id	DefineSourceId RetireSourceId DefineSourceSecurity QuerySourceSecurity RetireSourceSecurity DefineSource QuerySource RetireSource DefineSegment
42	Invalid Source Name	DefineSourceId
44	Missing PIN	ModifyDhctConfiguration ModifyDhctState
45	Missing required descriptor	All transactions
46	Package not defined	ModifyDhctConfiguration QueryPackage DeletePackage
47	Recurs prior to end	DefineSegment

Result Code	Description	Applicable Transactions
50	Segment not defined	RetireSegment QuerySegment DefinePackage
51	Source Id not defined	RetireSourceId DefineSource QuerySource RetireSource DefineSourceSecurity QuerySourceSecurity RetireSourceSecurity
52	Start time has passed	DefineSegment DefineSubstitutionUnit
54	VASP not registered	FetchVasp UpdateVasp
55	VASP previously registered	RegisterVasp
56	Extraneous descriptor present	All transactions
58	Segment definitions exist	RetireSourceId
60	File already exists	InitializeIppvUpload InitializeNonRespondingUpload
61	File not found	IppvUploadComplete GetIppvRecords NonRespondingUploadComplete GetNonRespondingRecords
62	Invalid file name	InitializeIppvUpload GetIppvRecords IppvUploadComplete
63	Invalid record number	GetIppvRecords
64	Invalid record count	GetIppvRecords
65	End of file	GetIppvRecords
66	Duplicate DHCTSE Serial Number	RegisterDhct UpdateDhct
67	File read failure	GetIppvRecords GetNonRespondingRecords
68	Duplicate IP Address	RegisterDhct UpdateDhct
69	PPV table full	ModifyDhctConfiguration AddAuthorizations
71	DHCT Type in use	DeleteDhctType
72	SwToc not found	RegisterDhct UpdateDhct RegisterDhctType UpdateDhctType
73	Invalid BillingId	RegisterDhct UpdateDhct ModifyDhctAdminStatus InitializeIppvUpload
74	Invalid ServiceName	DefineService QueryService RetireService
75	Invalid ShortDescription	DefineService
76	Invalid LongDescription	DefineService

Result Code	Description	Applicable Transactions
77	Invalid ApplicationUrl	DefineService
78	Invalid Logo	DefineService
79	Invalid ParameterString	DefineService
80	Invalid ParameterNumber	DefineService
81	Invalid ServiceId	QueryService RetireService
82	ServiceId not defined	QueryService RetireService
83	DisplayChannelMap not defined	QueryDisplayChannelMap RetireDisplayChannelMap
84	Invalid DisplayChannelMap	DefineDisplayChannelMap
85	Invalid SplitChannelMap	DefineDisplayChannelMap
86	Invalid Package Type modification	DefinePackage
87	No Purchases Reported	IppvPoll
88	No Response from DHCT	IppvPoll GetDhctDiagnostics
89	Invalid MibObjectId	GetDhctDiagnostics
90	Invalid MibQueryMode	GetDhctDiagnostics
91	Too many oids requested	GetDhctDiagnostics
92	No read access	GetDhctDiagnostics
93	Response buffer overflow	GetDhctDiagnostics
95	Hub already assigned to LUG	DefineLineUpGroup
96	Service has events defined	RetireService
97	PPV event window error	DefinePackage
98	LineUpGroup not defined	RetireLineUpGroup DefineSubstitutionUnit DefineDisplayChannelMap QueryDisplayChannelMap RetireDisplayChannelMap
99	LineUpGroup has outstanding substitutions	RetireLineUpGroup
100	DisplayChannelNumber not defined	DefineSubstitutionUnit AcceptSubstitutionUnit
102	Expiration Time has passed	DefineSubstitutionUnit AcceptSubstitutionUnit ExtendSubstitutionUnit RetireSubstitutionUnit
103	SubstitutionUnitId not defined	DefineSubstitutionUnit AcceptSubstitutionUnit ExtendSubstitutionUnit RetireSubstitutionUnit
104	Invalid HostId	DefineCertificateRevocationList AddHostToList RemoveHostFromList
105	CRL not found	AddHostToList RemoveHostFromList QueryCertificateRevocationList
9998	Unspecified error, request parsed correctly	All transactions
9999	Unspecified error	All transactions

4.2.66 SecurityMode

This descriptor is sent in the DefineSource transaction to indicate whether the segments that originate from a particular source are to be encrypted. The securityMode field in the descriptor may carry the following values:

Name
SM_Clear
SM_Encrypted

If a service is sent in the clear, no conditional access may be applied to the service, regardless of the service's package membership.

4.2.67 SegmentName

The SegmentName descriptor is sent to identify the service being described by the current descriptor loop.

4.2.68 SegmentSecurity

The SegmentSecurity descriptor is sent to the BCS to define the security control attributes of a new segment, or to modify the security control attributes of an existing segment. It is also sent by the BCS to report the definition of a segment.

The blackoutActive field allows a given segment to be blacked out or spotlighted within a geographical area specified by centroidX, centroidY, and radius. If blackoutActive is set to False, the values of centroidX, centroidY, and radius are ignored.

The fingerprintEnable bit is set to cause all DHCTs receiving the segment to display the serial number of the secure micro being used to decrypt the program. This capability may be used to track theft of authorization. fingerprintEnable can be turned off, set to display a visible fingerprint on the output video, or set to encode an invisible fingerprint on the output video.

PowerKEY defines a management mechanism for externally licensed copy protection systems. BCS is capable of managing the desired state of the copy protection system and securely communicating that state to the DHCT security element in a secure fashion. Actual control of the copy protection system is relegated to an application running on the DHCT.

The copyProtectionLevel field defines three copy protection states. They are:

- Copying permitted
- Copying not permitted
- Copying can be purchased

Copy protection management is supported by BCS, but the actual copy protection mechanism is a separately licensed technology. If the owner of the DNCS has not licensed a copy protection system, the only correct value for this field is Copying permitted.

The copyProtectionMode field is used to control the copy protection to be used for a given segment.

4.2.69 ServiceId

The ServiceId descriptor is returned from the DefineService transaction as a unique identifier for a service. The serviceId may be used to select the parameters for a given service when an application is written to send the parameters for all its supported services in a single file. The ServiceId can also be used in the QueryService and RetireService transactions.

4.2.70 ServiceName

The ServiceName descriptor is sent in the DefineService transaction to provide a unique identification of the service being defined.

4.2.71 ShortDescription

This descriptor is sent in the DefineService transaction to provide a short description of the service being defined.

4.2.72 SoftwareTOC

The SoftwareTOC decscriptor is sent to the Inventory & Directory Manager to define the Software Table-of-Contents for a DHCT or DHCT Type. The Software TOC is used to specify the name of the file that contains the table-of-contents to be used during software download.

4.2.73 SourceId

The SourceId descriptor is sent to identify the source being described by the current descriptor loop.

4.2.74 SourceName

The SourceName descriptor is sent to provide a human-readable name for the source being defined.

4.2.75 SplitChannelMap

The SplitChannelMap descriptor is sent in the DefineDisplayChannelMap transaction to provide a list of channels which can carry different services depending on the time of day.

4.2.76 StartTime

This descriptor is sent in the DefineSubstitutionUnit transaction to indicate the time at which the substitution unit will take effect, if accepted.

4.2.77 SubstitutionUnit

This descriptor is sent in the DefineSubstitutionUnit transaction to provide a set of old/new channel substitutions to be effected as a unit.

4.2.78 SubstitutionUnitID

The SubstitutionUnitId is sent to identify the Substitution Unit to which the transaction applies.

4.2.79 TimeOffset

The TimeOffset descriptor is sent to the Inventory & Directory Manager to provide time-of-day information for a DHCT. This descriptor is optional, such that when it is not included the DHCT will use the default time-of-day for the hub in which the DHCT is installed.

The offsetMinutes field allows a DHCT to use a time-of-day different from the default time-of-day.

The daylightSaving field allows a DHCT to observe a setting for Daylight Saving that is different from the default condition.

4.2.80 UploadFileSize

The UploadFileSize is sent in response to an I*nitializeIppvUpload* or InitializeNonRespondingUpload transaction to specify the number of records in the upload file created.

4.2.81 VaspId

The VaspId descriptor carries a Service (a.k.a. Video Information) Provider identifier.

4.2.82 VaspInfo

The VaspInfo descriptor is sent to the Inventory & Directory Manager to inform it of VASP profile information to be associated with a given VASP. It is also sent by the Inventory & Directory Manager to report the current values of the profile information.

The vaspStatus field of this descriptor indicates the network's view of the VASP. It can take one of two values:

Status	Description
1	In Service, indicating that, administratively, the VASP is in service and operational.
2	Out of Service, indicating that the VASP has been administratively disabled. The network will neither deliver DSM-CC messages to, nor accept DSM-CC messages from, the VASP.

5. BOSS Interface Transactions

In the following sections, each BOSS operation includes example request parameters and response values.

5.1 BOSS Transaction Summary

Table 5-1 provides a listing of BOSS transactions with a short description of each transaction. The table includes a list of valid descriptors for each transaction. Some transactions do not make use of descriptors and are so noted.

Table 4-1 provides the structural details of all BOSS descriptors.

Table 5-1

IDM Transactions			
RegisterDhct		This transaction adds a DHCT to the IDM database.	
Request	*Descriptor*	*M/O*	*Comments*
	DhctMacAddr	M	
	DhctseSerialNumber	O	
	KeyCertificate	O	
	IpAddress	O	
	DhctAdminStatus	M	
	BootPage	O	
	DhctType	M	
	TimeOffset	O	
	SwToc	O	
	NsapAddress	O	
	PrimaryVaspNsap	O	
	BillingId	O	
Response	Result	M	Can produce the following result codes: • No error • Bootterm Page not found • Invalid BootPage • Invalid Mac Address • Invalid DHCTSE Serial Number • Duplicate DHCTSE Serial Number • Invalid IP Address • Duplicate IP Address • Invalid NSAP Address • Missing Required Descriptor • Extraneous Descriptor present • DHCT previously registered • DHCT type not found

			• Invalid KeyCertificate • Invalid DhctAdminStatus • SwToc not found • Invalid BillingId • Unspecified error, request parsed correctly • Unspecified error
FetchDhct (apportioned)		This transaction fetches a complete DHCT record from the IDM.	
Request	*Descriptor*	*M/O*	*Comments*
	DhctMacAddr	M/O	DHCT profiles can be retrieved by MacAddr or by IpAddress. The two descriptors are mutually exclusive. This descriptor is required in the absence of an IpAddress descriptor.
	IpAddress	M/O	DHCT profiles can be retrieved by MacAddr or by IpAddress. The two descriptors are mutually exclusive. This descriptor is required in the absence of a DhctMacAddr descriptor.
Response	Result	M	Can produce the following result codes: • No error • DHCT Not Registered • Invalid Mac Address • Invalid IP Address • Missing Required Descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
	DhctseSerialNumber	M	
	KeyCertificate	M	
	IpAddress	M	
	DhctMacAddr	M	
	DhctAdminStatus	M	
	DhctOpnStatus	M	
	BootPage	M	
	DhctType	M	
	TimeOffset	M	
	SwToc	O	
	NsapAddress	O	
	PrimaryVaspNsap	M	
	BillingId	O	
UpdateDhct		This transaction is used to change any data in the DHCT profile except the MAC address of the DHCT.	
Request	*Descriptor*	*M/O*	*Comments*
	DhctMacAddr	M	

	DhctseSerialNumber	O	
	KeyCertificate	O	
	IpAddress	O	
	DhctAdminStatus	O	
	BootPage	O	
	DhctType	O	
	TimeOffset	O	
	SwToc	O	
	NsapAddress	O	
	PrimaryVaspNsap	O	
	BillingId	O	
Response	Result	M	Can produce the following result codes: • No error • Bootterm Page not found • Invalid BootPage • Invalid Mac Address • Invalid DHCTSE Serial Number • Duplicate DHCTSE Serial Number • Invalid IP Address • Duplicate IP Address • Invalid NsapAddress • Missing Required Descriptor • Extraneous Descriptor present • DHCT not registered • DHCT type not found • Invalid KeyCertificate • Invalid DhctAdminStatus • SwToc not found • Invalid BillingId • Unspecified error, request parsed correctly • Unspecified error
ModifyDhctAdminStatus		This transaction is used to change the Administrative Status in the DHCT.	
Request	*Descriptor*	*M/O*	*Comments*
	DhctMacAddr	M	
	DhctAdminStatus	M	
	BillingId	O	
Response	Result	M	Can produce the following result codes: • No error • Invalid Mac Address • Missing Required Descriptor • Extraneous Descriptor present • DHCT not registered • Invalid DhctAdminStatus

			• Invalid BillingId • Unspecified error, request parsed correctly • Unspecified error
ModifyDhctHubId (apportioned)		This transaction is used to change a DHCT's hub assignment.	
Request	*Descriptor*	*M/O*	*Comments*
	DhctMacAddr	M	
	HubId	O	
Response	Result	M	Can produce the following result codes: • No error • Invalid Mac Address • Missing Required Descriptor • Extraneous Descriptor present • DHCT not registered • Invalid HubId • Unspecified error, request parsed correctly • Unspecified error
DeregisterDhct		This transaction deletes DHCT records from the IDM database.	
Request	*Descriptor*	*M/O*	*Comments*
	DhctMacAddr	M	
Response	Result	M	Can produce the following result codes: • No error • Invalid Mac Address • Missing Required Descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
RegisterDhctType		This request registers a new DHCT type.	
Request	*Descriptor*	*M/O*	*Comments*
	DhctType	M	
	DhctTypeInfo	M	
	BootPage	O	
	SwToc	O	
Response	Result	M	Can produce the following result codes: • No error • Bootterm page not found • Invalid BootPage • Missing Required Descriptor • Extraneous Descriptor present • DHCT Type previously registered • SwToc not found • Unspecified error, request parsed correctly • Unspecified error
FetchDhctType (apportioned)		This transaction causes the IDM to return the definition of the indicated DHCT Type.	

Request	*Descriptor*	*M/O*	*Comments*
	DhctType	M	
Response	Result	M	Can produce the following result codes: • No error • DHCT Type not found • Missing Required Descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
	DhctTypeInfo	M	
	BootPage	M	
	SwToc	O	
UpdateDhctType		This transaction changes any data in the indicated DHCT Type except the DhctType.	
Request	*Descriptor*	*M/O*	*Comments*
	DhctType	M	
	DhctTypeInfo	M	
	BootPage	O	
	SwToc	O	
Response	Result	M	Can produce the following result codes: • No error • DHCT Type not found • Invalid BootPage • Bootterm Page not found • Missing Required Descriptor • Extraneous Descriptor present • SwToc not found • Unspecified error, request parsed correctly • Unspecified error
DeleteDhctType		This transaction causes the IDM to delete a DHCT Type based on the type ID.	
Request	*Descriptor*	*M/O*	*Comments*
	DhctType	M	
Response	Result	M	Can produce the following result codes: • No error • DHCT Type not found • DHCT Type in use • Missing Required Descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
RegisterVasp		This transaction registers the data for a new VASP with the IDM.	
Request	*Descriptor*	*M/O*	*Comments*

	VaspId	M	
	VaspInfo	M	
	NsapAddress	M	
	KeyCertificate	O	
Response	Result	M	Can produce the following result codes: • No error • VASP previously registered • Invalid NSAP address • Invalid IP address • Invalid KeyCertificate • Missing Required Descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
FetchVaspProfile (apportioned)		This transaction causes the IDM to return the data in the VASP profile indicated by its VASP ID.	
Request	*Descriptor*	*M/O*	*Comments*
	VaspId	M/O	VASP profiles can be retrieved by VaspId or by NsapAddress. The two descriptors are mutually exclusive. This descriptor is required in the absence of a NsapAddress descriptor.
	NsapAddress	M/O	VASP profiles can be retrieved by VaspId or by NsapAddress. The two descriptors are mutually exclusive. This descriptor is required in the absence of a VaspId descriptor.
Response	Result	M	Can produce the following result codes: • No error • VASP not registered • Invalid NSAP address • Missing Required Descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
	VaspId	M	
	VaspInfo	M	
	NsapAddress	M	
	KeyCertificate	M	
UpdateVaspProfile		This transaction changes any data in the indicated VASP profile.	
Request	*Descriptor*	*M/O*	*Comments*
	VaspId	M	
	VaspInfo	M	

	NsapAddress	M	
	KeyCertificate	O	
Response	Result	M	Can produce the following result codes: • No error • VASP not registered • Invalid NSAP address • Invalid IP address • Invalid KeyCertificate • Missing Required Descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
DeleteVasp		This transaction causes the IDM to delete the VASP Profile indicated by VASP ID.	
Request	*Descriptor*	*M/O*	*Comments*
	VaspId	M/O	VASP profiles can be deleted by VaspId or by NsapAddress. The two descriptors are mutually exclusive. This descriptor is required in the absence of a NsapAddress descriptor.
	NsapAddress	M/O	VASP profiles can be deleted by VaspId or by NsapAddress. The two descriptors are mutually exclusive. This descriptor is required in the absence of a VaspId descriptor.
Response	Result	M	Can produce the following result codes: • No error • Invalid NSAP address • VASP not registered • Missing Required Descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
RegisterBoottermPage		This transaction introduces a Bootterm page to the IDM.	
Request	*Descriptor*	*M/O*	*Comments*
	BootPage	M	
	BoottermPageInfo	M	
Response	Result	M	Can produce the following result codes: • No error • Bootterm page already exists • Invalid BootPage • Missing Required Descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
FetchBoottermPage (apportioned)		This transaction causes the IDM to return the bootterm page named by BootPage.	

(apportioned)			
Request	*Descriptor*	*M/O*	*Comments*
	BootPage	M	
Response	Result	M	Can produce the following result codes: • No error • Bootterm page not found • Missing Required Descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
	BootPage	M	
	BoottermPageInfo	M	
UpdateBoottermPage		This transaction changes any data in the indicated Bootterm Page record.	
Request	*Descriptor*	*M/O*	*Comments*
	BootPage	M	
	BoottermPageInfo	M	
Response	Result	M	Can produce the following result codes: • No error • Bootterm page not found • Missing Required Descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
DeleteBoottermPage		This transaction causes the IDM to delete the Bootterm Page record indicated by Bootterm Page Name.	
Request	*Descriptor*	*M/O*	*Comments*
	BootPage	M	
Response	Result	M	Can produce the following result codes: • No error • Bootterm page not found • Missing Required Descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
PowerKEY™ Control Transactions			
ModifyDhctConfiguration		This transaction modifies the status and authorizations of one or more DHCT records in the CAM database.	
Request	*Descriptor*	*M/O*	*Comments*
	DhctMacAddr	M	
	DhctState	M	
	PackageAuthorization	M	

	RefreshAll	M	
Response	Result	M	Can produce the following result codes: • No error • DHCT Not Registered • Invalid Mac Address • Package Not Defined • Invalid DhctState • Missing Required Descriptor • Extraneous Descriptor present • Invalid PackageAuthorization • Invalid creditLimit • Invalid maxIppvEvents • Missing PIN • Invalid Key Certificate • PPV table full • Unspecified error, request parsed correctly • Unspecified error
ModifyDhctState		This transaction modifies the status of one or more DHCT records in the CAM database.	
Request	*Descriptor*	*M/O*	*Comments*
	DhctMacAddr	M	
	DhctState	M	
Response	Result	M	Can produce the following result codes: • No error • DHCT Not Registered • Invalid Mac Address • Invalid DhctState • Missing Required Descriptor • Extraneous Descriptor present • Invalid creditLimit • Invalid maxIppvEvents • Missing PIN • Invalid Key Certificate • Unspecified error, request parsed correctly • Unspecified error
DhctInstantHit		This transaction causes the CAM to move the indicated DHCTs to the front of the authorization update queue.	
Request	*Descriptor*	*M/O*	*Comments*
	DhctMacAddr	M	
Response	Result	M	Can produce the following result codes: • No error • DHCT not registered • Invalid MAC Address • Missing Required Descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error

QueryDhct (apportioned)		This transaction obtains the current status of a DHCT's Enable DMS flag and IPPV purchase count, as well as the list of authorized packages.	
Request	*Descriptor*	*M/O*	*Comments*
	DhctMacAddr	M	
Response	Result	M	Can produce the following result codes: • No error • DHCT not registered • Invalid MAC Address • Missing Required Descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
	DhctState	O	This descriptor will not be produced if the QueryDhct request fails.
	PackageAuthorization	O	This descriptor will not be produced if the QueryDhct request fails.
ReplaceDhct		This transaction reassigns a DHCT record to a different DHCT.	
Request	*Descriptor*	*M/O*	*Comments*
	DhctMacAddr	M	Specifies an existing DHCT
	DhctMacAddrRep	M	
Response	Result	M	Can produce the following result codes: • No error • DHCT not registered • Invalid MAC Address • Invalid Key Certificate • Missing Required Descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
AddAuthorizations			
Request	*Descriptor*	*M/O*	*Comments*
	DhctMacAddr	M	Specifies an existing DHCT
	PackageAuthorization	M	
Response	Result	M	Can produce the following result codes: • No error • DHCT Not Registered • Invalid Mac Address • Invalid PackageAuthorization • Invalid Key Certificate • PPV table full • Missing Required Descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error

RemoveAuthorizations			
Request	*Descriptor*	*M/O*	*Comments*
	DhctMacAddr	M	Specifies an existing DHCT
	PackageAuthorization	M	
Response	Result	M	Can produce the following result codes: • No error • DHCT Not Registered • Invalid Mac Address • Invalid PackageAuthorization • Invalid Key Certificate • Missing Required Descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
IppvPoll		This transaction is used to retrieve purchased IPPV events for the specified DHCT.	
Request	*Descriptor*	*M/O*	*Comments*
	DhctMacAddr	M	Specifies an existing DHCT
Response	Result	M	Can produce the following result codes: • No error • DHCT Not Registered • Invalid Mac Address • No Purchases Reported • No Response from DHCT • Missing Required Descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
	RecordType	O	This descriptor will not be produced if the IppvPoll request fails.
	PackageName	O	This descriptor will not be produced if the IppvPoll request fails.
	IppvPurchase	O	This descriptor will not be produced if the IppvPoll request fails.
DefinePackage		This transaction enables the definition of a group of segments to be authorized as a single unit.	
Request	*Descriptor*	*M/O*	*Comments*
	PackageName	M	
	PackageMembership	M	
	PayPerView	O	Omission of this descriptor indicates that the package is authorized as a continuing segment or as an IPPV package. Presence of this descriptor indicates that authorization for a package may be purchased for a limited time.

	ImpulsePayPerView	O	Omission of this descriptor indicates that the package is not purchasable via the DHCT.
	LimitedDuration	O/M	Omission of this descriptor indicates the package is to be considered active immediately, and will continue to be active until a DeletePackage transaction is processed. This descriptor is mandatory if the PayPerView or ImpulsePayPerView descriptor is used.
	AllowExtension	O/M	This descriptor is used to indicate that the authorization for a package associated with a pay-per-view event may be extended in length. This descriptor is mandatory if the PayPerView or ImpulsePayPerView descriptor is used.
Response	Result	M	Can produce the following result codes: • No error • Invalid PackageName • Invalid PackageMembership • Invalid PayPerView • Invalid ImpulsePayPerView • Invalid LimitedDuration • Segment not defined • Missing required descriptor • Extraneous Descriptor present • Invalid Package Type modification • PPV Event Window Error • Unspecified error, request parsed correctly • Unspecified error
	EntitlementId	O	This descriptor will not be produced if the DefinePackage request fails.
QueryPackage (apportioned)		This transaction provides access to the current definition of a package.	
Request	*Descriptor*	*M/O*	*Comments*
	PackageName	M	
Response	Result	M	Can produce the following result codes: • No error • Invalid PackageName • Package not defined • Missing required descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified failure
	PackageName	O	This descriptor will not be produced if the QueryPackage request fails.
	EntitlementId	O	This descriptor will not be produced if the QueryPackage request fails.
	PackageMembership	O	This descriptor will not be produced if the QueryPackage request fails.
	PayPerView	O	This descriptor will not be produced if the package was not defined as PPV.

	ImpulsePayPerView	O	This descriptor will not be produced if the package was not defined as IPPV.
	LimitedDuration	O	This descriptor will not be produced if the package was not defined as PPV or IPPV.
	AllowExtension	O	This descriptor will not be produced if the package was not defined as either PPV or IPPV.
DeletePackage		This transaction is used to retire previously defined packages.	
Request	*Descriptor*	*M/O*	*Comments*
	PackageName	M	
Response	Result	M	Can produce the following result codes: • No error • Invalid PackageName • Package not defined • Missing required descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
ExtendEvent		This transaction is used to extend authorization of a package associated with a PPV or IPPV event.	
Request	*Descriptor*	*M/O*	*Comments*
	PackageName	M	
	EventExtension	M	
Response	Result	M	Can produce the following result codes: • No error • Invalid PackageName • PackageName Not Defined • Event Already Extended • Event Not Extendible • Missing required descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
InitializeIppvUpload		This transaction is used to request that an IPPV Upload file be created.	
Request	*Descriptor*	*M/O*	*Comments*
	IppvUploadFile	M	
	BillingId	O	If this descriptor is omitted, all IPPV purchase records will be included in the IPPV upload file.
Response	Result	M	Can produce the following result codes: • No error • Invalid BillingId • Invalid file name • File already exists • Missing required descriptor • Extraneous Descriptor present

			• Unspecified error, request parsed correctly • Unspecified error
	IppvUploadPath	O	This descriptor will not be produced if the initialize IPPV upload request fails.
	UploadFileSize	O	This descriptor will not be produced if the initialize IPPV upload request fails.
IppvUploadComplete		This transaction is used to notify the DNCS that an IPPV Upload file has been successfully processed by the AG.	
Request	*Descriptor*	*M/O*	*Comments*
	IppvUploadPath	M	
Response	Result	M	Can produce the following result codes: • No error • Invalid file name • File not found • Missing required descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
	IppvArchivePath	O	This descriptor will not be produced if the IPPV upload complete request fails.
GetIppvRecords		This transaction is used to retrieve IPPV purchase records from the DNCS.	
Request	*Descriptor*	*M/O*	*Comments*
	IppvUploadPath	M	
	StartRecordNumber	M	
	RecordCount	M	
Response	Result	M	Can produce the following result codes: • No error • Invalid file name • File not found • File read failure • Invalid record number • Invalid record count • End of file • Missing required descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
	DhctMacAddr	O	This descriptor will not be produced if the GetIppvRecords request fails.
	RecordType	O	This descriptor will not be produced if the GetIppvRecords request fails.
	PackageName	O	This descriptor will not be produced if the GetIppvRecords request fails.

	IppvPurchase	O	This descriptor will not be produced if the GetIppvRecords request fails.
	BillingId	O	This descriptor will not be produdced if the GetIppvRecords request fails.
AddRppvOrder (apportioned)		This transaction is used to add a RPPV order to the PPV Purchase database for inclusion in an IPPV upload file.	
Request	*Descriptor*	*M/O*	*Comments*
	DhctMacAddr	M	Specifies an existing DHCT
	PackageName	M	Identifies the RPPV package.
	OrderTime		Used to specify the time at which the order was added
	OrderSource		Used to identify the originator of the order
Response	Result	M	Can produce the following result codes: • No error • DHCT Not Registered • Invalid Mac Address • Missing Required Descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
AddRppvCancel (apportioned)		This transaction is used to add a RPPV order cancellation to the PPV Purchase database for inclusion in an IPPV upload file.	
Request	*Descriptor*	*M/O*	*Comments*
	DhctMacAddr	M	Specifies an existing DHCT
	PackageName	M	Identifies the RPPV package.
	OrderTime		Used to specify the time at which the order was cancelled
	OrderSource		Used to identify the originator of the cancellation
Response	Result	M	Can produce the following result codes: • No error • DHCT Not Registered • Invalid Mac Address • Missing Required Descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
# Broadcast Control Transactions			
DefineSourceID		This transaction is used to define a source ID to the BCS.	
Request	*Descriptor*	*M/O*	*Comments*
	SourceId	M	
	SourceName	M	
Response	Result	M	Can produce the following result codes: • No error

			• Invalid SourceId • Invalid SourceName • Missing Required Descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
RetireSourceId		This transaction is used to remove a previously defined SourceId.	
Request	*Descriptor*	*M/O*	*Comments*
	SourceId	M	
Response	Result	M	Can produce the following result codes: • No error • Invalid SourceId • SourceId not defined • Segment definitions exist • Missing Required Descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
DefineSourceSecurity		This transaction is used to define the security mode for a source.	
Request	*Descriptor*	*M/O*	*Comments*
	SourceId	M	
	SecurityMode	M	
	EffectiveTime	O	Omission of this descriptor indicates the source security is to be considered active immediately, and will continue to be active until another DefineSourceSecurity transaction or a RetireSourceSecurity transaction is processed.
Response	Result	M	Can produce the following result codes: • No error • Invalid SourceId • SourceID not defined • Invalid SecurityMode • Effective time has passed • Missing required descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
QuerySourceSecurity (apportioned)		This transaction provides access to the current security mode of a source.	
Request	*Descriptor*	*M/O*	*Comments*
	SourceId	M	
	EffectiveTime	O	Omission of this descriptor indicates that all source security modes for this SourceId are to be included in the response.

Response	Result	M	Can produce the following result codes: • No error • Invalid SourceId • SourceId not defined • Missing required descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
	SecurityMode	M	
	EffectiveTime	M	
RetireSourceSecurity		This transaction is used to retire a previously defined source security mode.	
Request	*Descriptor*	*M/O*	*Comments*
	SourceId	M	
	EffectiveTime	O/M	This descriptor is mandatory if multiple source security modes with different effective times have been defined for this SourceId.
Response	Result	M	Can produce the following result codes: • No error • Invalid SourceId • SourceId not defined • Missing required descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
DefineSource		This transaction is used to define a source to the BCS. Each segment to be defined must be assigned to a source.	
Request	*Descriptor*	*M/O*	*Comments*
	SourceId	M	
	DsmccSessionId	M/O	Omission of this descriptor indicates that the source is being on analog bandwidth.. In this case the ChannelId descriptor is mandatory.
	ChannelId	M/O	Omission of this descriptor indicates that the source is being defined on digital bandwidth In this case the DsmccSessionId descriptor is mandatory.
	EffectiveTime	O	Omission of this descriptor indicates the source definition is to be considered active immediately, and will continue to be active until another DefineSource transaction or a RetireSource transaction is processed.
	HubId	O	Omission of this descriptor indicates the source definition is to be applied to all hubs.
Response	Result	M	Can produce the following result codes: • No error • Invalid SourceId • SourceId not defined • Effective time has passed

			• Missing required descriptor • Extraneous Descriptor present • Invalid ChannelId • Invalid HubId • Unspecified error, request parsed correctly • Unspecified error
QuerySource (apportioned)		This transaction provides access to the current definition of a source.	
Request	*Descriptor*	*M/O*	*Comments*
	SourceId	M	
	EffectiveTime	O	Omission of this descriptor indicates that all source definitions for this SourceID and HubId (if included) are to be included in the response.
	HubId	O	Omission of this descriptor indicates that all source definitions for this SourceID and EffectiveTime (if included) are to be included in the response.
Response	Result	M	Can produce the following result codes: • No error • Invalid SourceId • SourceId not defined • Missing required descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
	DsmccSessionId	O	This descriptor is only included in the response if originally included in the definition.
	ChannelId	O	This descriptor is only included in the response if originally included in the definition.
	EffectiveTime	O	This descriptor is only included in the response if originally included in the definition.
	HubId	O	This descriptor is only included in the response if originally included in the definition.
RetireSource		This transaction is used to retire a previously defined source.	
Request	*Descriptor*	*M/O*	*Comments*
	SourceId	M	
	EffectiveTime	M/O	This descriptor is mandatory if multiple source definitions with different effective times have been defined for this SourceId.
	HubId	M/O	This descriptor is mandatory if multiple source definitions with different Hub Ids have been defined for this SourceId.
Response	Result	M	Can produce the following result codes: • No error • Invalid SourceId • SourceId not defined

			• • Missing required descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
DefineSegment		This transaction is used to provide the BCS with the information it needs to connect a segment to its bandwidth and package memberships.	
Request	*Descriptor*	*M/O*	*Comments*
	SegmentName	M	
	SourceId	M	
	SegmentSecurity	M/O	
	LimitedDuration	M/O	Omission of this descriptor indicates the segment is to be considered active immediately, and will continue to be active until a RetireSegment transaction is processed. This descriptor is mandatory if the Recurring descriptor is used.
	Recurring	O	Omission of this descriptor indicates the segment occurs precisely once. Use of this descriptor requires the LimitedDuration descriptor.
	EffectiveTime	M/O	Omission of this descriptor indicates the segment definition is to be considered active as defined by the LimitedDuration descriptor. This descriptor is mandatory if the LimitedDuration descriptor is not used.
Response	Result	M	Can produce the following result codes: • No error • Invalid SegmentName • Invalid SourceId • SourceId not defined • Invalid LimitedDuration • Invalid SegmentSecurity • Start time has passed • Invalid Recurring • Recurs prior to end • Effective time has passed • Missing required descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
QuerySegment (apportioned)		This transaction provides access to the current definition of a segment.	
Request	*Descriptor*	*M/O*	*Comments*
	SegmentName	M	
Response	Result	M	Can produce the following result codes: • No error • Invalid SegmentName

			• Segment not defined • Missing required descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
	SourceId	M	
	SegmentSecurity	O	This descriptor is only included in the response if originally included in the definition.
	LimitedDuration	O	This descriptor is only included in the response if originally included in the definition.
	Recurring	O	This descriptor is only included in the response if originally included in the definition.
	EffectiveTime	O	This descriptor is only included in the response if originally included in the definition.
RetireSegment		This transaction is used to retire a previously defined segment.	
Request	*Descriptor*	*M/O*	*Comments*
	SegmentName	M	
Response	Result	M	Can produce the following result codes: • No error • Invalid SegmentName • SegmentName not defined • Missing required descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
InitializeNonRespondingUpload (apportioned)		This transaction is used to request that a non-responding upload file be created.	
Request	*Descriptor*	*M/O*	*Comments*
	NonRespondingFile	M	
	NonRespondingDuration	M	
	BillingId	O	If this descriptor is omitted, all non-responding DHCTs will be included in the non-responding upload file.
Response	Result	M	Can produce the following result codes: • No error • File already exists • Missing required descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
	NonRespondingPath	O	This descriptor will not be produced if the initialize non-responding upload request fails.

	UploadFileSize	O	This descriptor will not be produced if the initialize non-responding upload request fails.
NonRespondingUploadComplete (apportioned)		This transaction is used to notify the DNCS that an non-responding upload file has been successfully processed by the AG.	
Request	*Descriptor*	*M/O*	*Comments*
	NonRespondingPath	M	
Response	Result	M	Can produce the following result codes: • No error • File not found • Missing required descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
	NonRespondingArchivePath	O	This descriptor will not be produced if the non-responding upload complete request fails.
GetNonRespondingRecords (apportioned)		This transaction is used to retrieve non-responding DHCT records from the DNCS.	
Request	*Descriptor*	*M/O*	*Comments*
	NonRespondingPath	M	
	StartRecordNumber	M	
	RecordCount	M	
Response	Result	M	Can produce the following result codes: • No error • File not found • File read failure • Missing required descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
	DhctMacAddr	M	
	ResponseTime	M	UNIX time.
	BillingId	O	
GetDhctDiagnostics ~~(apportioned)~~		This transaction is used to retrieve diagnostics information (in the form of SNMP MIB variables) for a DHCT from the DNCS.	
Request	*Descriptor*	*M/O*	*Comments*
	DhctMacAddr	M	
	MibObjectIdList	M	
	MibCommunityString	O	
	MibQueryMode	M	
Response	Result	M	Can produce the following result codes:

			• No error • DHCT not registered • Invalid Mac Address • Invalid MibObjectId • Invalid MibQueryMode • Too many oids requested • No read access • No response from DHCT • Missing required descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
	MibObjectList	O	This descriptor will not be produced if the GetDHCTDiagnostics transaction is not processed successfully.
	MibObjectId	O	This descriptor will only be produced if the GetDHCTDiagnostics transactions fails because of an invalid oid. It will contain the failing oid.
BootDhct ~~(apportioned)~~		This transaction is used to direct a DHCT to perform a network boot.	
Request	*Descriptor*	*M/O*	*Comments*
	DhctMacAddr	M	
Response	Result	M	Can produce the following result codes: • No error • DHCT not registered • Invalid Mac Address • Missing required descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error

Application Control Transactions

DefineService		This transaction is used to introduce a new application/parameter combination to the SAM, or to modify an existing service definition.	
Request	*Descriptor*	*M/O*	*Comments*
	ServiceName	M	
	ShortDescription	M	
	LongDescription	M	
	Logo	M	
	ApplicationUrl	M	
	ParameterString	M/O	This descriptor is required if the ParameterNumber descriptor is omitted.
	ParameterNumber	M/O	This descriptor is required if the ParameterString descriptor is omitted.

	DatabaseOperation	O	This descriptor is used by user interfaces that need to strictly control the appearance of a human presentable representation of the server's state. Administrative Gateways (such as cable billing systems) are not expected to use this descriptor.
Response	Result	M	Can produce the following result codes: • No error • Invalid ServiceName • Invalid ShortDescription • Invalid LongDescription • Invalid Logo • Invalid ApplicationUrl • Invalid ParameterString • Invalid ParameterNumber • Invalid DatabaseOperation • Database Insert Failed • Database Update Failed • Start Time has passed • Missing required descriptor • Unspecified error, request parsed correctly • Unspecified error
	ServiceId	O	This descriptor will only be returned when the Result descriptor carries a resultCode of No error.
QueryService		This transaction provides access to the current definition of a service. A given service may be queried either by its serviceId or by its serviceName.	
Request	*Descriptor*	*M/O*	*Comments*
	ServiceId	M/O	This descriptor is required if the ServiceName descriptor is omitted.
	ServiceName	M/O	This descriptor is required if the ServiceId descriptor is omitted.
Response	Result	M	Can produce the following result codes: • No error • Invalid ServiceId • Invalid ServiceName • ServiceId not defined • Missing required descriptor • Unspecified error, request parsed correctly • Unspecified failure
	ServiceId	M	
	ServiceName	M	
	ShortDescription	M	
	LongDescription	M	
	Logo	M	
	ApplicationUrl	M	This descriptor is only included in the response if originally included in the definition.

	ParameterString	O	This descriptor is only included in the response if originally included in the definition.
	ParameterNumber	O	This descriptor is only included in the response if originally included in the definition.
RetireService		This transaction is used to retire previously defined services. Services to be retired may be identified either by serviceId or by serviceName.	
Request	*Descriptor*	*M/O*	*Comments*
	ServiceId	M/O	This descriptor is required if the ServiceName descriptor is omitted.
	ServiceName	M/O	This descriptor is required if the ServiceId descriptor is omitted.
Response	Result	M	Can produce the following result codes: • No error • Invalid ServiceId • Invalid ServiceName • ServiceId not defined • Service has events defined • Missing required descriptor • Unspecified error, request parsed correctly • Unspecified error
DefineDisplayChannelMap		This transaction is used to define Display Channel Maps (DCMs). DCMs may be defined on a per-hub basis. There is a default DCM which is sent to any hub that does not have a specific DCM defined.	
Request	*Descriptor*	*M/O*	*Comments*
	LineUpGroupId	M	
	DisplayChannelMap	M	
	SplitChannelMap	O	The omission of this descriptor indicates that there are no split channels being defined for the DCM being defined.
Response	Result	M	Can produce the following result codes: • No error • Invalid DisplayChannelMap • Invalid SplitChannelMap • Missing required descriptor • Unspecified error, request parsed correctly • Unspecified error
QueryDisplayChannelMap		This transaction provides access to the current definition of a DisplayChannelMap	
Request	*Descriptor*	*M/O*	*Comments*
	LineUpGroupId	M	
Response	Result	M	Can produce the following result codes: • No error • LineUpGroup not defined • DisplayChannelMap not defined • Missing required descriptor

			• Unspecified error, request parsed correctly • Unspecified error
	DisplayChannelMap	M	
RetireDisplayChannelMap		This transaction is used to retire previously defined DCMs.	
Request	*Descriptor*	*M/O*	*Comments*
	LineUpGroupId	M	
Response	Result	M	Can produce the following result codes: • No error • LineUpGroup not defined • DisplayChannelMap not defined • Missing required descriptor • Unspecified error, request parsed correctly • Unspecified error
DefineLineUpGroup ~~(apportioned)~~		This transaction identifies a collection of hubs that will all have the same Display Channel Map.	
Request	*Descriptor*	*M/O*	*Comments*
	LineUpGroupId	M	
	LineUpGroupName	O	
	HubIdList	M	
Response	Result	M	Can produce the following result codes: • No error • Invalid Hub • Hub already assigned to LUG • Missing Required Descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
RetireLineUpGroup ~~(apportioned)~~		This transaction removes all the Hubs from the Line Up Group, and deletes it.	
Request	*Descriptor*	*M/O*	*Comments*
	LineUpGroupId	M	
Response	Result	M	Can produce the following result codes: • No error • LineUpGroup not defined • LineUpGroup has outstanding substitutions • Missing Required Descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
DefineSubstitutionUnit ~~(apportioned)~~		This transaction is used to define a collection of channel substitutions which may be effected as a unit.	
Request	*Descriptor*	*M/O*	*Comments*

	SubstitutionUnitId	M	
	LineUpGroupIdList	M	
	SubstitutionUnit	M	
	StartTime	O	This descriptor is optional, such that if it is not included the substitution is immediate (best effort). Substitution Units defined without this descriptor do not require the AcceptSubstitutionUnit transaction.
	ExpirationTime	O	This descriptor is optional, such that if it is not included the substitution must be manually retired.
Response	Result	M	Can produce the following result codes: • No error • Line Up Group not defined • Display Channel Number not defined • StartTime is past • ExpirationTime is past • Missing Required Descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
AcceptSubstitutionUnit ~~(apportioned)~~		This transaction authorizes a previously defined SubstitutionUnit to take effect at its specified StartTime. Note that SubstitutionUnits defined without a StartTime do not require this transaction in order to take effect.	
Request	*Descriptor*	*M/O*	*Comments*
	SubstitutionUnitId	M	
	DisplayChannelNumberList	O	This descriptor is included to indicate that only a portion of the originally defined SubstitutionUnit is to be accepted. Only the substitutions identified in this descriptor by their unsubstituted DCNs will be accepted. Omission of this descriptor indicates that all substitions in the SubstitutionUnit are to be accepted.
Response	Result	M	Can produce the following result codes: • No error • SubstitutionUnitId not defined • DisplayChannelNumber not defined • ExpirationTime has passed • Missing Required Descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
ExtendSubstitutionUnit ~~(apportioned)~~		This transaction is used to extend the lifetime of a Substitution Unit.	
Request	*Descriptor*	*M/O*	*Comments*
	SubstitutionUnitId	M	
	ExpirationTime	M	

Response	Result	M	Can produce the following result codes: • No error • SubstitutionUnitId not defined • ExpirationTime has passed • Missing Required Descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
RetireSubstitutionUnit ~~(apportioned)~~		This transaction retires a previously defined and accepted SubstitutionUnit.	
Request	*Descriptor*	*M/O*	*Comments*
	SubstitutionUnitId	M	
Response	Result	M	Can produce the following result codes: • No error • SubstitutionUnitId not defined • ExpirationTime has passed • Missing Required Descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error

CA Module Transactions

RegisterHost (apportioned)		This transaction is used to pair a Host device with a CA Module	
Request	*Descriptor*	*M/O*	*Comments*
	DhctMacAddr	M	
	HostId	M	
Response	Result	M	Can produce the following result codes: • No error • DHCT not registered • Invalid Mac Address • Invalid Host ID • Missing required descriptor • Extraneous Descriptor present • Unspecified error, request parsed correctly • Unspecified error
DeregisterHost (apportioned)		This transaction is used to disconnect a CA Module from its associated Host device.	
Request	*Descriptor*	*M/O*	*Comments*
	DhctMacAddr	M	
Response	Result	M	Can produce the following result codes: • No error • DHCT not registered • Invalid Mac Address • Missing required descriptor • Extraneous Descriptor present

			• Unspecified error, request parsed correctly • Unspecified error
DefineCertificateRevocationList (Apportioned)		This transaction is used to define the contents of the Certificate Revocation List (CRL)	
Request	*Descriptor*	*M/O*	*Comments*
	HostIdList	M	A list of HostIds to define the membership of the CRL. If a DatabaseOperation descriptor is included and set to update, this list will be treated as a replacement of the existing CRL.
	DatabaseOperation	O	If this descriptor is omitted, the group will be inserted if not previously defined, or updated if previously defined.
Response	Result	M	Can produce the following result codes: • No error • Invalid HostId • Invalid DatabaseOperation • Missing required descriptor • Unspecified error, request parsed correctly • Unspecified error
AddHostTo~~Group~~List (Apportioned)		This transaction is used to add a list of Host devices to the specified CRL.	
Request	*Descriptor*	*M/O*	*Comments*
	HostIdList	M	The list of Host devices to be added to the CRL.
Response	Result	M	Can produce the following result codes: • No error • CRL not found • Invalid HostId • Missing required descriptor • Unspecified error, request parsed correctly • Unspecified error
RemoveHostFrom~~Group~~List (Apportioned)		This transaction is used to remove a list of Host devices from the CRL.	
Request	*Descriptor*	*M/O*	*Comments*
	HostIdList	M	The list of Host devices to be removed from the CRL.
Response	Result	M	Can produce the following result codes: • No error • CRL not found • Invalid HostId • Missing required descriptor • Unspecified error, request parsed correctly • Unspecified error
QueryCertificateRevocationList (Apportioned)		This transaction is used to retrieve the current contents of the CRL	

Response	Result	M	Can produce the following result codes: • No error • CRL not found • Unspecified error, request parsed correctly • Unspecified error
	HostIdList	O	This descriptor will not be produced if the QueryCRL request fails.

5.2 General Transaction Format

This section provides a description of the fields that are common to every BOSS transaction, and of the general form of the request and response messages.

5.2.1 Request Format

Each BOSS transaction request contains a messageId field. Because the BOSS interface is asynchronous, this field is provided as a mechanism for the requester to correlate responses with their associated requests. DNCS makes no attempt to edit the messageId field, but will provide it in every subsequent communication concerning the request. It is the responsibility of the requester to define values for the messageID field, and to ensure that the value is unique. The requester may choose to maintain the uniqueness just for the life of the transaction (to insure the response can be matched to the request) or may wish to maintain the uniqueness for a longer period to allow historical records to have unique messageIds. In any event the management of the messageId field is left with the requester.

Each BOSS transaction request also contains a responseRpcProgram field. Because multiple BOSS client programs may be submitting transaction requests to a BOSS server, this field is provided as a mechanism for the server to route transaction responses to the proper client program.

The responseRpcProgram is followed by a descriptorLoopCount field. This field is an indication of the number of objects being described in the request.

Each descriptorLoop begins with a descriptorCount field, which is an indication of the number of descriptors included to describe the object in the current descriptor loop.

5.2.2 Simple Response Format

Every BOSS transaction response begins with the messageId, which was copied verbatim from the messageId field in the request message.

The messageId in the response message is followed by an errorCount field, which provides the total number of failure Result descriptors to be found in the response message. The errorCount field is incremented for each descriptor loop in the request that yielded a resultCode other than *No error*. This field may be examined to quickly determine whether the request was processed completely successfully.

Following the errorCount is a resultDescriptorCount, which provides the number of Result descriptors to follow. This field should be exactly equal to the descriptorLoopCount field in the request.

The remainder of the transaction is composed of Result descriptors, one per descriptorLoop in the request message.

5.2.3 Complex Response Format

The complex response format is similar to the simple response format, in that it begins with a messageId field and an errorCount field. These fields are followed by a descriptorLoopCount. Following the descriptorLoopCount is a series of descriptor loops which describe the result being returned. Each loop is required to contain exactly one descriptorCount, indicating the number of descriptors in the loop, exactly one Result descriptor, and any number of other descriptors required to completely specify the response.

This format is used to satisfy query-type transactions, which include a Result descriptor and other descriptors to carry the results of a successful query, and for transactions with failure codes that require parameters to fully specify the reason for failure.

In general, responses to query transactions include only optional descriptors for which values were specified in definitional BOSS transactions. If a query transaction fails, descriptors listed as mandatory are not returned.

5.2.4 General Format Summary Table

Table 5.3

General Format Summary Table		
	Field	*Type*
Request	messageId	unsigned long
	responseRpcProgram	unsigned int
	descriptorLoopCount	unsigned int
	descriptorCount	unsigned int
	<descriptors>	
Simple Response	messageId	unsigned long
	errorCount	unsigned int
	resultDescriptorCount	unsigned int
	<Result descriptors>	
Complex Response	messageId	unsigned long
	errorCount	unsigned int
	descriptorLoopCount	unsigned int
	descriptorCount	unsigned int

	< *descriptors*>	

5.3 Inventory & Directory Manager Transactions

A subset of the BOSS Interface provides provisioning operations that define DHCTs and VASPs to the DBDS. These operations are directed to the Inventory & Directory Manager. They include operations to provision DHCTs and their Bootterm pages and to define VASPs.

5.3.1 DHCT Transactions

The following sections describe the transactions for registering DHCTs that will be supported on the BOSS interface.

5.3.1.1 RegisterDhct

This transaction adds a DHCT to the IDM database. Note that a DHCT type must be registered before it can be used in a DHCT registration.

Upon receipt of a *RegisterDhct* request, the IDM will create a database record for each DHCT included in the request.

Upon completion of a *RegisterDhct* request, the IDM will provide a response with a result for each DHCT included in the request.

5.3.1.1.1 Request Structure

Multiple DHCTs can be specified. The descriptorLoopCount identifies the number of DHCTs to be added.

5.3.1.1.2 Response Structure

Each *RegisterDhct* response will include the messageId of the associated request, an errorCount which represents the number of DHCTs that were not processed successfully, and a result for each DHCT included in the request. The results will be provided in an array that matches the sequence of the array defined in the request. The Result Count identifies the number of elements in the array. A Return Code will be provided for each DHCT

5.3.1.2 FetchDhct

This transaction fetches a complete DHCT record from the Inventory & Directory Manager.

FetchDhct is an apportioned feature and is not currently implemented in the DNCS.

5.3.1.2.1 Request Structure

On receipt of a *FetchDhct* request, the Inventory & Directory Manager searches its database for a matching DHCT MAC address. It returns the matching record in response. In the event that the DHCT MAC address is not found, the Inventory & Directory Manager responds with an error indicating this.

5.3.1.2.2 Response Structure

For each DHCT named in the fetch request, the IDM will return the following descriptors. Null values or missing descriptors indicate that no value has been

registered with the IDM for the attribute. In the case of an error for a particular DHCT, a Result descriptor indicating the error will be returned in place of data for that DHCT. Note that the descriptors need not appear in this order.

Descriptor	Null Value
NsapAddress	zero-length opaque array
KeyCertificate	zero-length opaque array
IpAddress	null (zero-length) character string
DhctMacAddr	NA
DhctseSerialNumber	null (zero-length) character string
DhctType	zero (0)

5.3.1.3 UpdateDhct

The *UpdateDhct* request is identical in effect to a *DeregisterDhct* request followed by a *RegisterDhct* request. This transaction can be used to change any data in the DHCT profile except the MAC address of the DHCT.

5.3.1.3.1 Request Structure

The *UpdateDhct* request has the same structure as the *RegisterDhct* request.

5.3.1.3.2 Response Structure

Each *UpdateDhct* response will include the messageId of the associated request, an errorCount which represents the number of DHCTs that were not processed successfully, and a result for each DHCT included in the request. The results will be provided in an array that matches the sequence of the array defined in the request. The Result Count identifies the number of elements in the array.

5.3.1.4 ModifyDhctAdminStatus

The *ModifyDhctAdminStatus* request can be used to change the Administrative Status in the DHCT profile.

5.3.1.4.1 Request Structure

Multiple DHCTs will be provided as an array. The DHCT Count identifies the number of elements in the array

5.3.1.4.2 Response Structure

Each *ModifyDhctAdminStatus* response will include the messageId of the associated request, an errorCount which represents the number of DHCTs that were not processed successfully, and a result for each DHCT included in the request. The results will be provided in an array that matches the sequence of the array defined in the request. The Result Count identifies the number of elements in the array.

5.3.1.5 ModifyDhctHubId

The *ModifyDhctHubId* request can be used to change a DHCT's hub assignment.

5.3.1.5.1 Request Structure

Multiple DHCTs will be provided as an array. The DHCT Count identifies the number of elements in the array

5.3.1.5.2 Response Structure

Each *ModifyDhctHubId* response will include the messageId of the associated request, an errorCount which represents the number of DHCTs that were not processed successfully, and a result for each DHCT included in the request. The results will be provided in an array that matches the sequence of the array defined in the request. The Result Count identifies the number of elements in the array.

5.3.1.6 DeregisterDhct

This transaction deletes a DHCT record from the IDM database.

Upon receipt of a *DeregisterDhct* request, the IDM will search the database for a matching DHCT MAC address. If the MAC address is found, the IDM will delete the record from the database. If the MAC address is not found, the IDM will respond with an indication that the DHCT has not been registered.

Upon completion of a *DeregisterDhct* request, the IDM will provide a response indicating the success or failure of the transaction.

5.3.1.6.1 Request Structure

Multiple DHCTs will be provided as an array. The DHCT Count identifies the number of elements in the array

5.3.1.6.2 Response Structure

Each *DeregisterDhct* response will include the messageId of the associated request, an errorCount which represents the number of DHCTs that were not processed successfully, and a result for each DHCT included in the request. The results will be provided in an array that matches the sequence of the array defined in the request. The Result Count identifies the number of elements in the array. A Return Code will be provided for each DHCT included in the request.

5.3.2 DHCT Type Transactions

The DHCT registration operations described in the previous section use *DHCT Type* as part of the DHCT profile. The type communicates information about the model or revision of DHCT in place. The following operations act on the *DHCT Type* records to which the DHCT profiles refer.

Note that a DHCT Type must be defined before it is used in a DHCT registration transaction.

5.3.2.1 RegisterDhctType

This request registers a new DHCT type.

5.3.2.1.1 Request Structure

Multiple DHCT types can be specified. The descriptorLoopCount identifies the number of DHCT types to be added.

5.3.2.1.2 Response Structure

Each *RegisterDhctType* response will include the messageId of the associated request, an errorCount which represents the number of DHCT Types that were not processed successfully, and a result for each DHCT Type included in the request. The results will be provided in an array that matches the sequence of the array defined in the request. The Result Count identifies the number of elements in the array. A Return Code will be provided for each DHCT Type included in the request.

5.3.2.2 FetchDhctType

When provided with the DHCT type number, the IDM will search its database and return the DHCT Type record matching that ID. If none exists, the IDM will return an error indicating this.

FetchDhctType is an apportioned feature and is not currently implemented in the DNCS.

5.3.2.2.1 Request Structure

Multiple DHCT Types will be provided as an array. The DHCT Type Count identifies the number of elements in the array.

5.3.2.2.2 Response Structure

Errors are indicated with Result descriptors in the place of DHCT Type data.

5.3.2.3 UpdateDhctType

This request is equivalent to a *DeleteDhctType* followed by a *RegisterDhctType* request. This transaction can be used to change any data in the DHCT Type profile except the Type ID.

5.3.2.3.1 Request Structure

The UpdateDhctType has the same request structure as the RegisterDhctType request.

5.3.2.3.2 Response Structure

Each *UpdateDhctType* response will include the messageId of the associated request, an errorCount which represents the number of DHCT Types that were not processed successfully, and a result for each DHCT Type included in the request. The results will be provided in an array that matches the sequence of the array defined in the request. The resultDescriptorCount identifies the number of elements in the array. A Return Code will be provided for each DHCT Type included in the request.

5.3.2.4 DeleteDhctType

The Inventory & Directory Manager will delete a DHCT Type based on the DHCT Type ID. This transaction is constrained to accept only a single DHCT Type for deletion.

5.3.2.4.1 Request Structure

The DeleteDhctType request can contain only a single DHCT Type ID.

5.3.2.4.2 Response Structure

The *DeleteDhctType* response is constrained to return only a single result descriptor.

5.3.3 VASP Registration Transactions

As VASPs are introduced to the network, certain configuration data must be registered. The data include the name of the provider, the IP address to which service requests should be delivered, and the public key certificate holding the VASP's public cryptographic key.

It is worth clarifying the relationship of the VASP NSAP and VASP IP addresses. The VASP NSAP address is the address used in DSM-CC messages to address the VASP. The VASP IP address is the IP address used by the DNCS to deliver DSM-CC messages. Note that it is possible, though unlikely, that a single host system could represent several VASPs in DSM-CC communications. In this case, each VASP entry in the IDM would have a unique NSAP address, while they would share IP addresses.

5.3.3.1 RegisterVasp

This request registers the data for a new VASP with the network.

5.3.3.1.1 Request Structure

Multiple VASPs can be specified. The descriptorLoopCount identifies the number of VASPs to be added.

5.3.3.1.2 Response Structure

Each *RegisterVasp* response will include the messageId of the associated request, an errorCount which represents the number of VASPs that were not processed successfully, and a result for each VaspId included in the request. The results will be provided in an array that matches the sequence of the array defined in the request. The Result Count identifies the number of elements in the array. A Return Code will be provided for each VaspId included in the request.

5.3.3.2 FetchVaspProfile

When provided with a VASP NSAP ADDRESS or ID, the Inventory & Directory Manager will search its database and return the VASP record matching that ID. If none exists, the Inventory & Directory Manager will return an error indicating this.

FetchVaspProfile is an apportioned feature and is not currently implemented in the DNCS.

5.3.3.2.1 Request Structure

VASPs may be named using either ID or NSAP address. Two request forms cover these cases:

Data Element	Example
messageId	*123*
descriptorCount	*3*
NsapAddress	<>
...	
NsapAddress	<>

Data Element	Example
messageId	*123*
descriptorCount	*3*
VaspId	<>
...	
VaspId	<>

5.3.3.2.2 Response Structure

The Inventory & Directory Manager returns the complete record matching either the supplied VASP NSAP ADDRESS or VASP ID. Errors yield a Result descriptor in the place of VASP data..

5.3.3.3 UpdateVaspProfile

This request is equivalent to a *DeleteVasp* followed by a *RegisterVasp* request.

5.3.3.3.1 Request Structure

The UpdateVaspProfile request has the same structure as the RegisterVasp request.

5.3.3.3.2 Response Structure

The UpdateVaspProfile response has the same structure as the RegisterVasp response.

5.3.3.4 DeleteVasp

The Inventory & Directory Manager will delete a VASP's profile based on either VASP NSAP ADDRESS or VASP ID. Once a VASP's record has been removed from the Inventory & Directory Manager, new requests for service will not be forwarded by the network to that provider.

5.3.3.4.1 Request Structure

VASPs may be named using either ID or NSAP address. Two request forms cover these cases:

Data Element	Example
messageId	*123*
descriptorCount	*3*
NsapAddress	<>
...	
NsapAddress	<>

Data Element	Example
messageId	*123*
descriptorCount	*3*
VaspId	*127.0.0.1*
...	
VaspId	*127.0.0.3*

5.3.3.4.2 Response Structure

Each *DeleteVasp* response will include the messageId of the associated request, an errorCount which represents the number of VASPs that were not processed successfully, and a result for each VaspId included in the request. The results will be provided in an array that matches the sequence of the array defined in the request. The Result Count identifies the number of elements in the array. A Return Code will be provided for each VaspId included in the request.

5.3.4 Bootterm Page Administration

Bootterm pages are stored by the Inventory & Directory Manager for use by the Element Manager during DHCT initialization. The Bootterm page descriptions in the Inventory & Directory Manager include both the page contents and owner. Bootterm pages are identified with a textual name of up to 20 characters.

5.3.4.1 RegisterBoottermPage

This transaction introduces Bootterm pages to the access network. If a page with the given name already exists, the Inventory & Directory Manager will return an error indicating this.

5.3.4.1.1 Request Structure

Multiple Bootterm pages can be specified. The descriptorLoopCount identifies the number of Bootterm pages to be added.

5.3.4.1.2 Response Structure

Each *RegisterBoottermPage* response will include the messageId of the associated request, an errorCount which represents the number of Bootterm pages that were not processed successfully, and a result for each Boot Page included in the request. The results will be provided in an array that matches the sequence of the array defined in the request. The Result Count identifies the number of elements in the array. A Return Code will be provided for each Boot Page included in the request.

5.3.4.2 FetchBoottermPage

Given a Bootterm page name, the Inventory & Directory Manager returns the contents of the page. If no page exists with the given name, the Inventory & Directory Manager returns an error to that effect.

FetchBoottermPage is an apportioned feature and is not currently implemented in the DNCS.

5.3.4.2.1 Request Structure

Multiple Boot Pages will be provided as an array. The descriptorCount identifies the number of elements in the array.

5.3.4.2.2 Response Structure

The Inventory & Directory Manager returns the complete record matching the supplied Boot Page. Errors yield Result Descriptors in the place of bootterm page data.

5.3.4.3 UpdateBoottermPage

This operation is equivalent to a *DeleteBoottermPage* request followed by a *RegisterBoottermPage* request.

5.3.4.3.1 Request Structure

The UpdateBoottermPage request has the same structure as the RegisterBoottermPage request.

5.3.4.3.2 Response Structure

The UpdateBoottermPage response has the same structure as the RegisterBoottermPage response.

5.3.4.4 DeleteBoottermPage

The Inventory & Directory Manager searches its database for a Bootterm page with the given name and deletes that page. The *DeleteBoottermPage* request is constrained to delete a single bootterm page.

5.3.4.4.1 Request Structure

The DeleteBoottermPage request can contain only a single Boot Page.

5.3.4.4.2 Response Structure

The DeleteBoottermPage response is constrained to contain only a single result descriptor.

5.4 PowerKEY™ Control Transactions

The following sections describe the transactions for authorizing use of Digital Broadcast Services for DHCTs that will be supported on the BOSS interface. These transactions will be received and processed by the Conditional Access Manager (CAM), which is a component of the PowerKEY™ Control Suite.

Table 5-1 provides a definitive summary of the CAM transactions. For each transaction it lists a short description of the usage of the transaction and a list of descriptors used in the transaction, broken into a request section and a response section. Each descriptor also carries an indication of whether the descriptor is mandatory or optional, and a short comment about the descriptor. This table also lists the result codes that may be produced by each transaction.

5.4.1 DHCT State Management Transactions

5.4.1.1 ModifyDhctConfiguration

This transaction configures one or more DHCTs in the CAM database. *ModifyDhctConfiguration* requests enable DMS, enable DIS, enable analog, enable IPPV purchases, configure maximum IPPV purchases to be stored, configure IPPV purchase credit limit, configure the pin, set the location in the network, configure the refresh rate, and authorize packages for one or more DHCTs with a single transaction. A DHCT must be registered in the IDM database prior to being configured in the CAM database.

This transaction involves the identification of the packages a DHCT is authorized to receive. Package Names are used to identify packages, which may consist of either a single bandwidth segment, such as Subscription HBO, or a set of segments, such as the package of basic services.

The DhctState descriptor causes the indicated DHCT's state to be discarded and replaced in completion with the state specified in the DhctState descriptor. Likewise, the PackageAuthorization descriptor causes all package authorizations for the specified DHCT to be discarded and replaced with those specified in the PackageAuthorization descriptor in the descriptor loop. This provides a method for synchronizing the DHCT's state and authorizations with the Administrative Gateway.

Upon completion of the *ModifyDhctConfiguration* request, the CAM will respond with indications of the success or failure of the transaction.

5.4.1.1.1 Request Structure

Multiple DHCTs can be specified. The descriptorLoopCount identifies the number of DHCTs to be modified.

5.4.1.1.2 Response Structure

Each *ModifyDhctConfiguration* response will include the messageId of the associated request, an errorCount which represents the number of DHCTs that were not processed successfully, and a result for each descriptor loop included in the request.

5.4.1.2 ModifyDhctState

This transaction configures the conditional access state of one or more DHCTs in the CAM database. ModifyDhctState requests enable DMS, enable DIS, enable analog, enable IPPV purchases, configure maximum IPPV purchases to be stored, configure IPPV purchase credit limit, configure the pin, set the location in the network, and configure the refresh rate for one or more DHCTs. A DHCT must be registered in the IDM database prior to being configured in the CAM database.

The DhctState descriptor causes the indicated DHCT's state to be discarded and replaced in completion with the state specified in the DhctState descriptor. This transaction will not affect the package authorizations field of the DHCT. However, disabling DMS with this transaction will cause the DHCT to disallow access to any broadcast service. This provides a method for the Administrative Gateway to configure the DHCT's state without requiring the reconstruction and re-encryption of all the DHCT's authorization EMMs which is much more efficient if it is anticipated that service will be restored as defined by the authorizations.

Upon completion of the ModifyDhctState request, the CAM will respond with indications of the success or failure of the transaction.

5.4.1.2.1 Request Structure

Multiple DHCTs can be specified. The descriptorLoopCount identifies the number of DHCTs to be modified.

5.4.1.2.2 Response Structure

Each *ModifyDhctState* response will include the messageId of the associated request, an errorCount which represents the number of DHCTs that were not processed successfully, and a result for each descriptor loop included in the request.

5.4.1.3 DhctInstantHit

The CAM constructs Entitlement Management Messages (EMMs) from the information it receives in the ModifyDhctConfiguration transaction. The CAM periodically transmits these EMMs to the DHCTs it maintains. To do this, the CAM maintains a queue of authorization EMMs that are to be sent to the DHCTs.

This transaction causes the CAM to move the indicated DHCTs to the front of this authorization update queue. This results in an expedited transmission of the DHCT's configuration information to the DHCT, without requiring a reload of that information to the CAM, which would in turn require the CAM to reformat and resecure the EMMs. This configuration information includes all the fields specified in the ModifyDhctConfiguration transaction.

5.4.1.3.1 Request Structure

Multiple DHCTs can be specified. The descriptorLoopCount identifies the number of DHCTs to be modified.

5.4.1.3.2 Response Structure

Each *ModifyDhctState* response will include the messageId of the associated request, an errorCount which represents the number of DHCTs that were not processed successfully, and a result for each descriptor loop included in the request.

5.4.1.4 QueryDhct

This transaction provides access to the current status of a DHCT's DhctState descriptor, as well as a list of authorized Packages.

Upon receipt of a *QueryDhct* request, the CAM will search its database for a matching DHCT MAC address. If the MAC address is found, the CAM will respond with the status of DhctState descriptor, and a list of currently authorized packages. If the DHCT MAC address is not found, the CAM will respond with an indication that the DHCT has not been setup.

QueryDhct is an apportioned feature and is not currently implemented in the DNCS.

5.4.1.4.1 Request Structure

Multiple DHCTs can be specified. The descriptorLoopCount identifies the number of DHCTs to be retrieved.

5.4.1.4.2 Response Structure

In the response structure, the returned descriptor loops match positionally with those in the request structure. In the case of a failure, only a result descriptor will be returned.

5.4.1.5 ReplaceDhct

This transaction transfers a DHCT's configuration information to a different DHCT.

Upon receipt of a *ReplaceDhct* request, the CAM will query the IDM to determine that the replacement DHCT has been provisioned to the system. If the replacement DHCT's mac address is not known to the IDM, the CAM will fail the replacement of the existing DHCT associated with the unprovisioned replacement DHCT. If the replacement DHCT is found, the CAM will search its database for the MAC address of the DHCT to be replaced. If the MAC address is found, the CAM will overwrite the existing DHCT MAC address with the new MAC address included in the request.

In the event the DHCT MAC address is not found, the CAM will respond with an indication that the DHCT is not setup.

Upon completion of the *ReplaceDhct* request, the CAM will respond with an indication of the success or failure of the transaction.

5.4.1.5.1 Request Structure

Multiple DHCTs can be specified. The descriptorLoopCount identifies the number of DHCTs to be replaced.

5.4.1.5.2 Response Structure

Each *ReplaceDhct* response will include the messageId of the associated request, an errorCount which represents the number of DHCTs that were not processed successfully, and a result for each DHCT included in the request.

5.4.1.6 AddAuthorizations

This transaction configures one or more DHCTs in the CAM database. *AddAuthorizations* requests add packages to the list of previously authorized packages for one or more DHCTs with a single transaction. A DHCT must be registered in the IDM database prior to being configured in the CAM database.

This transaction involves the identification of the packages a DHCT is authorized to receive. Package Names are used to identify packages, which may consist of either a single bandwidth segment, such as Subscription HBO, or a set of segments, such as the package of basic services.

Upon completion of the *AddAuthorizations* request, the CAM will respond with indications of the success or failure of the transaction.

5.4.1.6.1 Request Structure

Multiple DHCTs can be specified. The descriptorLoopCount identifies the number of DHCTs to be authorized.

5.4.1.6.2 Response Structure

Each *AddAuthorizations* response will include the messageId of the associated request, an errorCount which represents the number of DHCTs that were not processed successfully, and a result for each descriptor loop included in the request.

5.4.1.7 RemoveAuthorizations

This transaction configures one or more DHCTs in the CAM database. *RemoveAuthorizations* requests delete packages from the list of previously authorized packages for one or more DHCTs with a single transaction. A DHCT must be registered in the IDM database prior to being configured in the CAM database.

This transaction involves the identification of the packages a DHCT is authorized to receive. Package Names are used to identify packages, which may consist of either a single bandwidth segment, such as Subscription HBO, or a set of segments, such as the package of basic services.

Upon completion of the *RemoveAuthorizations* request, the CAM will respond with indications of the success or failure of the transaction.

5.4.1.7.1 Request Structure

Multiple DHCTs can be specified. The descriptorLoopCount identifies the number of DHCTs to be deauthorized.

5.4.1.7.2 Response Structure

Each *RemoveAuthorizations* response will include the messageId of the associated request, an errorCount which represents the number of DHCTs that were not processed successfully, and a result for each descriptor loop included in the request.

5.4.1.8 IppvPoll

This transaction causes the CAM to issue an IPPV Poll for one DHCT and provides access to any purchased IPPV events that are reported.

Upon receipt of a *IppvPoll* request, the CAM will search its database for a matching DHCT MAC address. If the MAC address is found, the CAM will respond with any purchased IPPV events. If the DHCT MAC address is not found, the CAM will respond with an indication that the DHCT has not been setup. This transaction is constrained to accept only a single DHCT MAC address.

5.4.1.8.1 Request Structure

Only a single DHCT may be polled.

5.4.1.8.2 Response Structure

Multiple purchased IPPV events may be included in the response. The descriptorLoopCount identifies the number of IPPV events being returned.

5.4.2 Package Definition Transactions

5.4.2.1 DefinePackage

The *DefinePackage* transaction enables the definition of a group of segments or packages to be authorized as a single unit. A single package may consist of one or more segments, and a single segment may belong to one or more packages. Whenever a segment is retired, either as a result of expiration as specified with a LimitedDuration descriptor or as a result of a RetireSegment transaction, the membership of all packages whose membership includes the retired segment is adjusted.

5.4.2.1.1 Request Structure

Multiple Packages can be specified. The descriptorLoopCount identifies the number of Packages to be added.

5.4.2.1.2 Response Structure

Each *DefinePackage* response will include the messageId of the associated request, an errorCount which represents the number of packages that were not processed successfully, and a result for each descriptor loop included in the request.

5.4.2.2 QueryPackage

This transaction provides access to the current definition of a package.

Upon receipt of a *QueryPackage* request, the CAM will search its database for a matching package. If the package is found, the CAM will respond with the package membership and conditional access attributes of the requested package. If the package is not found, the CAM will respond with an error result.

QueryPackage is an apportioned feature and is not currently implemented in the DNCS.

5.4.2.2.1 Request Structure

Multiple Packages can be specified. The descriptorLoopCount identifies the number of Packages to be retrieved.

5.4.2.2.2 Response Structure

In the response structure, the returned descriptor loops match positionally with those in the request structure. In the case of a failure, only a result descriptor will be returned.

5.4.2.3 DeletePackage

This transaction is used to retire previously defined packages. As a side effect of this transaction, all authorizations for the retired packages will be also be retired.

5.4.2.3.1 Request Structure

Multiple Packages can be specified. The descriptorLoopCount identifies the number of Packages to be deleted.

5.4.2.3.2 Response Structure

In the response structure, the returned descriptor loops match positionally with those in the request structure.

5.4.2.4 ExtendEvent

This transaction is used to extend authorization of a package associated with a PPV or IPPV event. This capability is used for programming, such as live sporting events, that extends beyond its scheduled running time.

It is important to note that PowerKEY defines that PPV packages may only be extended one single time. Additional requests for extension will be failed by the CAM.

5.4.2.4.1 Request Structure

Multiple PPV events can be specified. The descriptorLoopCount identifies the number of events to be extended.

5.4.2.4.2 Response Structure

In the response structure, the returned descriptor loops match positionally with those in the request structure.

5.4.3 IPPV Purchase Upload Transactions

5.4.3.1 InitializeIppvUpload

This transaction is used to request the creation of a file containing IPPV purchase data for upload to the AG.

5.4.3.1.1 Request Structure

This transaction is constrained to accept only a single IPPV file creation request.

5.4.3.1.2 Response Structure

Each *InitializeIppvUpload* response will include a result code, an IPPV Upload file path, and the number of IPPV purchase records contained in the file. If the transaction fails, only the result code will be included.

5.4.3.2 IppvUploadComplete

This transaction is used to notify the DNCS that an IPPV upload file has been successfully processed by the AG.

5.4.3.2.1 Request Structure

This transaction is constrained to accept only a single IPPV upload complete request.

5.4.3.2.2 Response Structure

The *IppvUploadComplete* response is constrained to include only a single result descriptor.

5.4.3.3 GetIppvRecords

This transaction is used to retrieve IPPV purchase records from a previously created IPPV upload file.

5.4.3.3.1 Request Structure

The *GetIppvRecords* request contains starting record number and number of records to be retrieved.

5.4.3.3.2 Response Structure

In the response structure, the returned descriptor loops contain the records retrieved. If the end of file is encountered before the number of records requested is satisfied, the value of the descriptorLoopCount will indicate the number of records included in the response. If the transaction fails, only the result code will be included.

5.4.4 RPPV

5.4.4.1 AddRppvOrder

This transaction is used to add RPPV orders to the PPV Purchase database for inclusion in an IPPV upload file.

This transaction is an apportioned feature and is not available in the DNCS.

5.4.4.1.1 Request Structure

Multiple RPPV order can be specified. The descriptorLoopCount identifies the number of orders to be added.

5.4.4.1.2 Response Structure

In the response structure, the returned descriptor loops match positionally with those in the request structure.

5.4.4.2 AddRppvCancell

This transaction is used to add RPPV order cancellations to the PPV Purchase database for inclusion in an IPPV upload file.

This transaction is an apportioned feature and is not available in the DNCS.

5.4.4.2.1 Request Structure

Multiple RPPV cancellation can be specified. The descriptorLoopCount identifies the number of cancellations to be added.

5.4.4.2.2 Response Structure

In the response structure, the returned descriptor loops match positionally with those in the request structure.

5.5 Broadcast Control Transactions

The following sections describe the transactions for defining Digital Broadcast Services that will be supported on the BOSS interface. These transactions will be received and processed by the Broadcast Control Suite (BCS).

Table 5-1 provides a definitive summary of the BCS transactions. For each transaction it lists a short description of the usage of the transaction and a list of descriptors used in the transaction, broken into a request section and a response section. Each descriptor also carries an indication of whether the descriptor is mandatory or optional, and a short comment about the descriptor. This table also lists the result codes that may be produced by each transaction.

The sections that follow use tabular examples of the BOSS transactions. These examples are not normative indications of the descriptors that may or may not be present in a transaction. Refer to Table 5-1 for specific transaction details.

5.5.1 Source Definition Transactions

5.5.1.1 DefineSourceId

The *DefineSourceId* transaction is used to define a Source ID to the BCS.

5.5.1.1.1 Request Sructure

Multiple Source IDs can be specified. The descriptorLoopCount identifies the number of Source IDs to be added.

5.5.1.1.2 Response Structure

In the response structure, the returned descriptor loops match positionally with those in the request structure.

5.5.1.2 RetireSourceId

The *RetireSourceId* transaction is used to delete a Source ID.

5.5.1.2.1 Request Sructure

Multiple Source IDs can be specified. The descriptorLoopCount identifies the number of Source IDs to be deleted.

5.5.1.2.2 Response Structure

In the response structure, the returned descriptor loops match positionally with those in the request structure.

5.5.1.3 DefineSourceSecurity

The *DefineSourceSecurity* transaction is used to specify the security mode of any source ID defined. All source definitions defined with the source ID specified will use this security mode.

If a given source is defined to be unencrypted, by setting it's securityMode field to be Clear, no conditional access is performed for any segment assigned to that source.

5.5.1.3.1 Request Structure

Multiple Sources can be specified. The descriptorLoopCount identifies the number of Sources to be added.

5.5.1.3.2 Response Structure

In the response structure, the returned descriptor loops match positionally with those in the request structure.

5.5.1.4 QuerySourceSecurity

This transaction retrieves the current security mode for each named source ID.

Upon receipt of a *QuerySourceSecurity* request, the BCS will search its database for a matching source ID. If the source ID is found, the BCS will respond with security mode of the requested source. If the source ID is not found, the BCS will respond with an error result.

QuerySourceSecurity is an apportioned feature and is not currently implemented in the DNCS.

5.5.1.4.1 Request Structure

Multiple Sources can be specified. The descriptorLoopCount identifies the number of Sources to be retrieved.

5.5.1.4.2 Response Structure

The BCS returns the security mode matching the source ID specified. Errors yield a Result descriptor in place of source definition data.

5.5.1.5 RetireSourceSecurity

This transaction is used to retire a previously defined source security mode.

5.5.1.5.1 Request Structure

Multiple Sources can be specified. The descriptorLoopCount identifies the number of Sources to be deleted.

5.5.1.5.2 Response Structure

In the response structure, the returned descriptor loops match positionally with those in the request structure.

5.5.1.6 DefineSource

The *DefineSource* transaction is used to identify the programming source of any broadcast services to be supported on the BOSS interface. The source of the programming to be carried on any bandwidth segment must be defined to the BCS in advance and specified when the segment is defined.

5.5.1.6.1 Request Structure

Multiple Sources can be specified. The descriptorLoopCount identifies the number of Sources to be added.

5.5.1.6.2 Response Structure

In the response structure, the returned descriptor loops match positionally with those in the request structure.

5.5.1.7 QuerySource

This transaction retrieves the current definition for each named source.

Upon receipt of a *QuerySource* request, the BCS will search its database for a matching source. If the source is found, the BCS will respond with definition of the requested source. If the source is not found, the BCS will respond with an error result.

QuerySource is an apportioned feature and is not currently implemented in the DNCS.

5.5.1.7.1 Request Structure

Multiple Sources can be specified. The descriptorLoopCount identifies the number of Sources to be retrieved.

5.5.1.7.2 Response Structure

The BCS returns the complete record matching the source specified. Errors yield a Result descriptor in place of source definition data.

Note that if the EffectiveTime descriptor is omitted, all source definitions that match the SourceId and HubId (if included in the request) will be returned. If the HubId descriptor is omitted, all source definitions that match the SourceId and EffectiveTime (if included in the request) will be returned. If both EffectiveTime and HubId are omitted, all source definitions that match the SourceId will be returned.

5.5.1.8 RetireSource

This transaction is used to retire a previously defined source. A RetireSource transaction will be failed if any defined segments are assigned to this source. Therefore all segments assigned to this source must be removed prior to the issuance of the RetireSource transaction.

5.5.1.8.1 Request Structure

Multiple Sources can be specified. The descriptorLoopCount identifies the number of Sources to be deleted.

5.5.1.8.2 Response Structure

In the response structure, the returned descriptor loops match positionally with those in the request structure.

5.5.2 Segment Definition Transactions

5.5.2.1 DefineSegment

In the BCS context, a segment is some externally defined offering, carried over a given piece of bandwidth, for which the CAM may be required to supply the conditional access services of authorization and encryption.

The CAM provides authorizations to DHCTs via EMMs, which are addressed to individual DHCTs. An EMM contains references to the packages for which a given DHCT is authorized. When a user makes a service selection via some application, such as an IPG, the DHCT is expected to validate its authorized packages against the package affiliations for the selected service. The package affiliations are delivered securely in ECMs, which are imbedded in the bandwidth over which the selected segment is delivered. The BCS, therefore, must have knowledge of the bandwidth over which a given segment offering is carried, so that the CAM may arrange for the encryption of the bandwidth, and so that it may insert ECMs into the segment's bandwidth.

If a given segment is defined with a SourceId whose securityMode field is set to Clear, that segment will be unencrypted and no conditional access is performed for the segment. It is important to note that segments running over the same bandwidth over a given time interval must be defined with a consistent security mode. If a segment is defined against bandwidth which is already carrying another segment, and the existing segment is assigned to a source that has a security mode which is inconsistent with that of the source assigned to the new segment, the new segment's definition will be rejected.

The *DefineSegment* transaction is used to provide BCS with the information it needs to connect a segment to its bandwidth and, via the *DefinePackage* transaction, its package membership.

5.5.2.1.1 Request Structure

Multiple Segments can be specified. The descriptorLoopCount identifies the number of Segments to be added.

5.5.2.1.2 Response Structure

In the response structure, the returned descriptor loops match positionally with those in the request structure.

5.5.2.2 QuerySegment

This transaction retrieves the current definition for each named segment.

QuerySegment is an apportioned feature and is not currently implemented in the DNCS.

5.5.2.2.1 Request Structure

Multiple Segments can be specified. The descriptorLoopCount identifies the number of Segments to be retrieved.

5.5.2.2.2 Response Structure

In general, responses to query transactions only include descriptors that were specified in the associated define transactions. Optional descriptors which were omitted in the define are likewise omitted in the response to the query. Because the BCS can determine additional bandwidth specification information via non-BOSS channels, in the specific case of the response to the *QuerySegment* transaction the BCS will return bandwidth descriptors for which it has values, whether those values were determined via the *DefineSegment* transaction or via other channels.

5.5.2.3 RetireSegment

This transaction is used to retire a previously defined segment. A *RetireSegment* transaction will be failed if the indicated segment is a member of a package, therefore all package memberships for the segment must be removed prior to the issuance of the *RetireSegment* transaction.

5.5.2.3.1 Request Structure

The *RetireSegment* request is constrained to accept only a single segment for deletion.

5.5.2.3.2 Response Structure

The *RetireSegment* response is constrained to return only a single result descriptor.

5.5.3 Non-Responding Upload Transactions

5.5.3.1 InitializeNonRespondingUpload

This transaction is used to request the creation of a file containing non-responding DHCT data for upload to the AG.

InitializeNonRespondingUpload is an apportioned feature and is not currently implemented in the DNCS.

5.5.3.1.1 Request Structure

This transaction is constrained to accept only a single non-responding file creation request.

5.5.3.1.2 Response Structure

Each *InitializeNonRespondingUpload* response will include a result code, a non-responding upload file path, and the number of non-responding DHCT records contained in the file. If the transaction fails, only the result code will be included.

5.5.3.2 NonRespondingvUploadComplete

This transaction is used to notify the DNCS that a non-responding upload file has been successfully processed by the AG.

NonRespondingUploadComplete is an apportioned feature and is not currently implemented in the DNCS.

5.5.3.2.1 Request Structure

This transaction is constrained to accept only a single non-responding upload complete request.

5.5.3.2.2 Response Structure

The *NonRespondingUploadComplete* response is constrained to include only a single result descriptor.

5.5.3.3 GetNonRespondingRecords

This transaction is used to retrieve non-responding DHCT records from a previously created non-responding upload file.

GetNonRespondingRecords is an apportioned feature and is not currently implemented in the DNCS.

5.5.3.3.1 Request Structure

The *GetNonRespondingRecords* request contains starting record number and number of records to be retrieved.

5.5.3.3.2 Response Structure

In the response structure, the returned descriptor loops contain the records retrieved. If the end of file is encountered before the number of records requested is satisfied, the value of the descriptorLoopCount will indicate the number of records included in the response. If the transaction fails, only the result code will be included.

5.5.4 DHCT Diagnostics Transactions

5.5.4.1 GetDhctDiagnostics

This transaction is used to retrieve diagnostic information (in the form of SNMP MIB variables) for a DHCT from the DNCS.

5.5.4.1.1 Request Structure

Multiple DHCTs can be specified. The descriptorLoopCount identifies the number of DHCTs to be included in the response.

5.5.4.1.2 Response Structure

In the response structure, the returned descriptor loops match positionally with those in the request structure. If the case of a failure, only a result descriptor will be returned.

5.5.4.2 BootDhct

This transaction is used to direct a DHCT to perform a network boot.

5.5.4.2.1 Request Structure

Multiple DHCTs can be specified. The descriptorLoopCount identifies the number of DHCTs to be included in the response.

5.5.4.2.2 Response Structure

In the response structure, the returned descriptor loops match positionally with those in the request structure. If the case of a failure, only a result descriptor will be returned.

5.6 Application Control Transactions

5.6.1 Service Definition Transactions

5.6.1.1 DefineService

This transaction is used to introduce a new application/parameter combination to the SAM, or to modify an existing service definition.

5.6.1.1.1 Request Structure

Multiple Services can be specified. The descriptorLoopCount identifies the number of Services to be added.

5.6.1.1.2 Response Structure

In the response structure, the returned descriptor loops match positionally with those in the request structure.

5.6.1.2 QueryService

This transaction provides access to the current definition of a service. A given service may be queried either by its serviceId or by its shortDescription.

5.6.1.2.1 Request Structure

Multiple Services can be specified. The descriptorLoopCount identifies the number of Services to be retrieved.

5.6.1.2.2 Response Structure

In the response structure, the returned descriptor loops match positionally with those in the request structure. In the case of a failure, only a result descriptor will be returned.

5.6.1.3 RetireService

This transaction is used to retire previously defined services. Services to be retired may be identified either by serviceId or by shortDescription.

5.6.1.3.1 Request Structure

Multiple Services can be specified. The descriptorLoopCount identifies the number of Services to be removed.

5.6.1.3.2 Response Structure

In the response structure, the returned descriptor loops match positionally with those in the request structure.

5.6.2 Channel Map Transactions

5.6.2.1 DefineDisplayChannelMap

This transaction is used to define Display Channel Maps (DCMs). DCMs may be defined on a per-lineup group basis. There is a default DCM which is sent to any lineup group that does not have a specific DCM defined.

5.6.2.1.1 Request Structure

Multiple DCMs can be specified. The descriptorLoopCount identifies the number of DCMs to be added.

5.6.2.1.2 Response Structure

In the response structure, the returned descriptor loops match positionally with those in the request structure.

5.6.2.2 QueryDisplayChannelMap

This transaction provides access to the current definition of a DCM. The DCM for a particular lineup group or the default DCM may be retrieved.

5.6.2.2.1 Request Structure

Multiple DCMs can be specified. The descriptorLoopCount identifies the number of DCMs to be retrieved.

5.6.2.2.2 Response Structure

In the response structure, the returned descriptor loops match positionally with those in the request structure. In the case of a failure, only a result descriptor will be returned.

5.6.2.3 RetireDisplayChannelMap

This transaction is used to retire previously defined DCMs.

5.6.2.3.1 Request Structure

Multiple DCMs can be specified. The descriptorLoopCount identifies the number of DCMs to be removed.

5.6.2.3.2 Response Structure

In the response structure, the returned descriptor loops match positionally with those in the request structure.

5.6.3 Channel Substitution Transactions

5.6.3.1 DefineLineupGroup

This transaction is used to identify a collection of Hubs that will all have the same Display Channel Map.

5.6.3.1.1 Request Structure

Multiple LUGs can be specified. The descriptorLoopCount identifies the number of LUGs to be added.

5.6.3.1.2 Response Structure

In the response structure, the returned descriptor loops match positionally with those in the request structure.

5.6.3.2 RetireLineupGroup

This transaction is used to remove all Hubs from a LUG and delete the LUG.

5.6.3.2.1 Request Structure

Multiple LUGs can be specified. The descriptorLoopCount identifies the number of LUGs to be removed.

5.6.3.2.2 Response Structure

In the response structure, the returned descriptor loops match positionally with those in the request structure.

5.6.3.3 DefineSubstitutionUnit

This transaction is used to define a collection of channel substitutions which may be applied as a unit.

5.6.3.3.1 Request Structure

Multiple SubstitutionUnitIds can be specified. The descriptorLoopCount identifies the number of Substitution Units to be added.

5.6.3.3.2 Response Structure

In the response structure, the returned descriptor loops match positionally with those in the request structure.

5.6.3.4 AcceptSubstitutionUnit

This transaction is used to authorize a previously defined SubstitutionUnit to take effect at its specified StartTime. Note that Substitution units defined without a StartTime do not require this transaction to take effect.

5.6.3.4.1 Request Structure

Multiple SubstitutionUnitIds can be specified. The descriptorLoopCount identifies the number of Substitution Units to be applied.

5.6.3.4.2 Response Structure

In the response structure, the returned descriptor loops match positionally with those in the request structure.

5.6.3.5 ExtendSubstitutionUnit

This transaction is used to extend the lifetime of a SubstitutionUnit.

5.6.3.5.1 Request Structure

Multiple SubstitutionUnitIds can be specified. The descriptorLoopCount identifies the number of Substitution Units to be extended.

5.6.3.5.2 Response Structure

In the response structure, the returned descriptor loops match positionally with those in the request structure.

5.6.3.6 RetireSubstitutionUnit

This transaction is used to retire a previously defined and accepted SubstitutionUnit.

5.6.3.6.1 Request Structure

Multiple SubstitutionUnitIds can be specified. The descriptorLoopCount identifies the number of Substitution Units to be removed.

5.6.3.6.2 Response Structure

In the response structure, the returned descriptor loops match positionally with those in the request structure.

5.7 OpenCable Transactions

5.7.1 Host Registration Transactions

5.7.1.1 RegisterHost

This transaction is used to pair a Host device with a CA Module.

RegisterHost is an apportioned feature and is not currently implemented in the DNCS.

5.7.1.1.1 Request Structure

Multiple Host/CA Module pairs can be specified. The descriptorLoopCount identifies the number of Hosts to be registered.

5.7.1.1.2 Response Structure

In the response structure, the returned descriptor loops match positionally with those in the request structure.

5.7.1.2 DeregisterHost

This transaction is used to disconnect a CA Module from its associated Host device.

DeregisterHost is an apportioned feature and is not currently implemented in the DNCS.

5.7.1.2.1 Request Structure

Multiple CA Modules can be specified. The descriptorLoopCount identifies the number of CA Modules to be initialized.

5.7.1.2.2 Response Structure

In the response structure, the returned descriptor loops match positionally with those in the request structure.

5.7.2 Certificate Revocation List Transactions

5.7.2.1 DefineCertificateRevocationList

This transaction is used to define the contents of the Certificate Revocation List.

DefineCertificateRevocationList is an apportioned feature and is not currently implemented in the DNCS.

5.7.2.1.1 Request Structure

This transaction is constrained to accept only a single CRL definition.

5.7.2.1.2 Response Structure

The response structure for this transaction is constrained to return only a single result descriptor.

5.7.2.2 AddHostToList

This transaction is used to add a list of Host devices to the CRL.

AddHostToList is an apportioned feature and is not currently implemented in the DNCS.

5.7.2.2.1 Request Structure

This transaction is constrained to accept only a single list of Host Ids.

5.7.2.2.2 Response Structure

The response structure for this transaction is constrained to return only a single result descriptor.

5.7.2.3 RemoveHostFromList

This transaction is used to remove a list of Host devices from the CRL.

RemoveHostFromList is an apportioned feature and is not currently implemented in the DNCS.

5.7.2.3.1 Request Structure

This transaction is constrained to accept only a single list of Host Ids.

5.7.2.3.2 Response Structure

The response structure for this transaction is constrained to return only a single result descriptor.

5.7.2.4 QueryCertificateRevocationList

This transaction is used to retrieve the contents of the Certificate Revocation List.

QueryCertificateRevocationList is an apportioned feature and is not currently implemented in the DNCS.

5.7.2.4.1 Request Structure

There are no descriptors supplied with the request structure for this transaction.

5.7.2.4.2 Response Structure

The response structure for this transaction contains a list of all Host Ids in the CRL

5.8 Usage Data Interface

Several DNCS components, such as the CAM, can produce information describing usage of some system resource. This section describes the retrieval interface that will apply to usage data supplied by any DNCS component. Subsequent sections describe specific characteristics of each usage data interface, such as record formats.

In general, each DNCS component that produces usage data produces a data file, according to its specific file format, which is be retrieved by the AG using File Transfer Protocol (FTP). The file is retained by the DNCS component for a limited period of time for protection against accidental corruption of the file.

Each DNCS component maintains three directories for the purposes of storing the data files and communicating with the AG. When the usage data file is produced, it will be placed into the primary directory, indicating the file has not been retrieved by the AG. The AG is provided read-permission on the primary directory. The DNCS component will provide a minimum of five days storage of usage data files in the primary directory. If the storage limit is reached, usage data files will be moved from the primary directory to the secondary directory, oldest first, until sufficient space is recovered in the primary storage area.

The AG periodically retrieves new usage files from the primary directory via FTP. When the retrieval is successful, the AG deposits a file into the acknowledge directory, again via FTP. The AG is provided write-permission on the acknowledge directory. The acknowledge file contains a newline-delimited list of file names which have been successfully retrieved from the primary directory.

As the DNCS component processes the acknowledge file, it moves each named file from the primary directory to the secondary directory. The AG is provided read-permission on the secondary directory. The AG may examine this directory for files it has missed, and may issues retrievals from the secondary directory. The DNCS component will provide a minimum five days of storage of usage data files in the secondary directory. If the storage limit is reached, usage data files will be deleted, oldest first, until sufficient space is recovered in the secondary storage area.

All data will be represented using the eXternal Data Representation (XDR) standard. UNIX time will be used for all date/time data. Elapsed time will be provided in units of seconds. Bandwidth data will be provided in units of bits per second.

5.8.1 Session Usage Data Interface

Session usage data is collected by the Session Manager. A Record Type indicates that the record is a session record. The Record Count identifies the number of session records in a session file.

5.8.1.1 Session Record Format

The Session Manager creates a unique session record containing the appropriate session usage data at the completion of each digital interactive session. A new session record will be created whenever a change in network resource utilization occurs which will have a DSM-CC Session ID that can be correlated to the DSM-CC Session ID of related sessions.

The following data elements are included in session records.

Data Element	Example
recordType	S
dhctMacAddress	*A183DB4645EF*
dsmccSessionId	<>
vaspNsapAddress	<>
sessionType	Exc
completionStatusFlag	Comp
startTime	*(UNIX Time)*
elapsedTime	*7200*
downstreamBandwidth	*3,000,000*
upstreamBandwidth	*800*

5.8.2 Impulse Pay-per-view Purchase Data Interface

Impulse Pay-per-view purchase data is collected by the CAM either just prior to authorization of a IPPV segment (dynamic model), or soon after the DHCT has recorded the fact of a IPPV purchase in its security element (store-and-forward model). IPPV purchase data is stored in a file for retrieval by the Administrative Gateway.

The DHCT user is allowed to cancel an IPPV purchase for a period of time following the purchase, as specified in the cancelWindowInterval field of the ImpulsePayPerView descriptor. Cancellations are required to be reported to the CAM, and are recorded by the CAM in the IPPV purchase data file.

The IPPV purchase data file contains two types of records: Digital IPPV purchase records and Digital IPPV purchase cancellation records. A record type field will be used to distinguish the various records.

5.8.2.1 Digital IPPV Purchase Record

The following data elements are included in Digital IPPV purchase records:

Data Element	Type	Range	Length	Example
recordType	enum	RT_DigitalIppvPurchase RT_DigitalIppvCancel		*RT_DigitalIppvPurchase*
dhctMacAddress	char		*17*	*A183DB4645EF*
packageName	string		20	*"Tyson-Douglas Fight"*
purchaseTime	unsigned int			*(UNIX time)*
Length	unsigned int			*90*
cost	unsigned int			*1*
rightToCopy	enum	RTC_CopyPermitted RTC_CopyProhibited		*RTC_CopyProhibited*
billingId	unsigned int			*10*

5.8.2.2 Digital IPPV Purchase Cancellation Record

The following data elements are included in Digital IPPV purchase cancellation records:

Data Element	Type	Range	Length	Example
recordType	enum	RT_DigitalIppvPurchase RT_DigitalIppvCancel		*RT_DigitalIppvCancel*
dhctMacAddress	char		*17*	*A183DB4645EF*
packageName	string		20	*"Tyson-Douglas Fight"*
purchaseTime	unsigned int			*(UNIX time)*
Length	unsigned int			*90*
cost	unsigned int			*1*
rightToCopy	enum	RTC_CopyPermitted RTC_CopyProhibited		*RTC_CopyProhibited*

billingId	unsigned int			*10*

5.8.3 Non-Responding DHCT Data Interface

The DNCS maintains a database that contains the last time that a reverse path message was received from each DHCT. A file that contains the MAC Address of every DHCT that has not successfully transmitted a reverse path message can be created for retrieval by the Administrative Gateway.

5.8.3.1 Non-Responding DHCT Record

The following data elements are included in Non-Responding DHCT records:

Data Element	Type	Range	Length	Example
dhctMacAddress	char		*17*	*A183DB4645EF*
responseTime	unsigned long			*(UNIX time)*
billingId	unsigned int			*10*