OpenCable™ Specifications

OCAP Headend Common Download and Unbound Application Signaling Interface Specification

OC-SP-OHI-I01-061208

ISSUED

Document Status Sheet

Document Control Number:	OC-SP-OHI-I01-061208			
Document Title:	OCAP Headend Common Download and Unbound Application Signaling Interface Specification			
Revision History:	I01 – Released December 8, 2006			
Date:	December 8, 2006			
Status:	~~Work in Progress~~	~~Draft~~	Issued	~~Closed~~
Distribution Restrictions:	~~Author Only~~	~~CL/Member~~	~~CL/ Member/ Vendor~~	Public

Key to Document Status Codes:

Work in Progress	An incomplete document, designed to guide discussion and generate feedback, that may include several alternative requirements for consideration.
Draft	A document in specification format considered largely complete, but lacking review by Members and vendors. Drafts are susceptible to substantial change during the review process.
Issued	A stable document, which has undergone rigorous member and vendor review and is suitable for product design and development, cross-vendor interoperability, and for certification testing.
Closed	A static document, reviewed, tested, validated, and closed to further engineering change requests to the specification through CableLabs.

Trademarks:

Contents

Figures

1 SCOPE AND PURPOSE

1.1 Scope

This specification provides a simple, open interface between a generic Stream Generator (SG) and a headend controller. Specifically, this specification defines a common Web Service (WS) interface to the headend controller that enables an SG to deliver and control transmission of OpenCable Common Download and OCAP Application Download tables (CVT and XAIT, respectively) on the headend cable system via the headend controller.

The primary goal of this interface specification is to provide SG and headend controller developers with a detailed description of the WS in order to build their client and server (respectively) side interfaces. Cable operators may then employ any SG and headend controller adhering to this interface and be assured that they will be capable of interfacing with one another using the interface features described herein.

1.2 Purpose

The purpose of this specification is to provide an open interface between a Stream Generator and a headend controller.

2 REFERENCES

2.1 Normative References

In order to claim compliance with this specification, it is necessary to conform to the following standards and other works as indicated, in addition to the other requirements of this specification. Notwithstanding, intellectual property rights may be required to use or implement such normative references.

[CCIF2.0] CableCARD Interface 2.0 Specification, OC-SP-CCIF2.0-I08-061031, October 31, 2006, Cable Television Laboratories, Inc.

[CDL2.0] OpenCable Common Download 2.0 Specification, OC-SP-CDL2.0-I01-061031, October 31, 2006, Cable Television Laboratories, Inc.

[HTTP] Hypertext Transfer Protocol – HTTP/1.0, http://www.w3.org/Protocols/HTTP/1.0/draft-ietf-http-spec.html

[MHP] ETSI TS 102 812 V1.2.1, Digital Video Broadcasting (DVB): Multimedia Home Platform (MHP) Specification 1.1.1.

[OCAP] OpenCable Application Platform Profile 1.0 Specification, OC-SP-OCAP1.0-I16-050803, August 3, 2005, Cable Television Laboratories, Inc.

[WDSL] Web Services Description Language (WSDL) 1.1, W3C Note 15, March 2001, http://www.w3.org/TR/2001/NOTE-wsdl-20010315.

2.2 Informative References

[SCTE 28] ANSI/SCTE 28 2004, HOST-POD Interface Standard.

[SGO] Stream Generator Overview, CL-SP-SGO-D01-060814, August 14, 2006, Cable Television Laboratories, Inc.

2.3 Reference Acquisition

CableLabs Specifications:

- Cable Television Laboratories, Inc., 858 Coal Creek Circle, Louisville, CO 80027;
  Phone 303-661-9100; Fax 303-661-9199; Internet: http://www.cablelabs.com /

DVB specifications:

- DVB Group, c/o EBU, 17a Ancienne Route, CH-1218 Grand Saconnex, Geneva, Switzerland;
  Telephone: + 41 22 717 27 14; Fax: + 41 22 717 27 27; Internet: http://www.dvb.org/

W3C:

- http://www.w3.org

3 TERMS AND DEFINITIONS

This specification uses the following terms:

Headend controller	A device used to configure and provision headend equipment, set-tops, and CableCARDs.
Handle	A number assigned to an object by one system or device which is used as a reference to that object by another system or device.
Stream Generator	A DSM-CC (Digital Storage Media - Command and Control) Carousel Server.
Well-formed	Constructed correctly as per a specification.

4 ABBREVIATIONS, ACRONYMS AND CONVENTIONS

4.1 Abbreviations and Acronyms

This specification uses the following abbreviations:

CVT	Code Version Table
HE	Headend
MSO	Multiple-Systems Operator
OCAP	OpenCable Application Platform
OOB	Out-of-Band
SG	Stream Generator
SGIF	Stream Generator Interface
WSDL	Web Services Description Language
XAIT	Extended Application Information Table

4.2 Conventions

Throughout this document, the words that are used to define the significance of particular requirements are capitalized. These words are:

"SHALL"	This word means that the item is an absolute requirement of this specification.
"SHALL NOT"	This phrase means that the item is an absolute prohibition of this specification.
"SHOULD"	This word means that there may exist valid reasons in particular circumstances to ignore this item, but the full implications should be understood and the case carefully weighed before choosing a different course.
"SHOULD NOT"	This phrase means that there may exist valid reasons in particular circumstances when the listed behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
"MAY"	This word or the adjective "OPTIONAL" means that this item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because it enhances the product, for example; another vendor may omit the same item.

5 OVERVIEW

The approach this specification describes is one in which the Stream Generator (SG) is responsible for building a complete XAIT and/or CVT. After building the XAIT and/or CVT, the SG transmits it to the headend controller for subsequent transmission to an OCAP-compliant end user device. Multiple XAITs and CVTs are supported to enable capable headend systems to provide geographical and/or device level targeted downloads. An example diagram in which multiple XAITs are being transmitted on the cable system is shown in Figure I–1 and Figure I–2 in Appendix I.

It is the role of the headend controller to act as the conduit by which XAITs are transmitted on the cable system. The headend controller SHALL NOT be responsible for building or merging XAITs originating from multiple Stream Generators into either a single XAIT or multiple XAITs. Therefore, any merging of XAITs or XAIT data into an XAIT SHALL occur before transmission to the headend controller. An example diagram is shown in Figure I–3.

In order for the SG to build a complete XAIT or CVT, it may require information that is known only to the headend controller. There are two methods by which this information may be provided to the SG:

- Via a network interface (i.e., Web Service) or
- Via the OpenCable stream generator operator manually entering information obtained visually from the headend controller.

Some of the command and control features described below are defined as optional to permit both headend controller and SG vendors the opportunity to phase in support and/or provide support in varying degrees/levels to increase the value added to their customers.

The current SGIF system is described in [SGO].

6 INTERFACE DESCRIPTION

6.1 Security

To secure communications between the SG and headend controller, HTTP Basic Authentication (see [HTTP], section 11) SHALL be used to authenticate SGs to the headend controller. In addition, all communications between the SG and the headend controller SHALL be encrypted using standard Hypertext Transfer Protocol over Secure Socket Layer (HTTPS = HTTP over SSL).

SSL communications SHALL conform to the following:

- There will be no SSL authentication of the client by the server or of the server by the client. There will, therefore, be no need for signed certificate or public key exchange.
- Protocol used may be SSLv2, SSLv3 or TLSv1.
- The following cipher suite SHALL be supported:
    - Anonymous Diffie-Hellman authentication/key exchange, RC4 encoding, MD5 hash function.

Other cipher suites MAY optionally be supported.

6.2 Versioning

Different versions of this specification SHALL be distinguished by the endpoint URL used to access the service. As a suggestion, the trailing portion of the URL could match the document control number of this specification as shown in the example below.

https://example.com/service/OC-SP-OHI-I01-060128

6.3 XAIT Support

In order for the SG to have all of the information it needs to create the XAIT message, it SHALL be provided with the following headend information:

sourceID – At least one sourceID SHALL be provided; however, several may be needed. The number of sourceIDs needed depends on how the SG chooses to configure the carousel server applications. For example, the SG operator may choose to associate all OCAP applications with a single sourceID, or may choose to associate each OCAP application with a different sourceID, or some combination of the two.

Note: A sourceID is required only if the protocol for carrying the application is an Object Carousel. For example, if the protocol for carrying the application is the Interaction Channel, then an HTTP URL would be used by the stream generator in lieu of a sourceID.

XAIT Destination Handle – In order to support receiving multiple XAIT messages from the SG, the SG SHALL be provided with a handle for each XAIT it creates and transmits to the headend controller.

A handle is required so that the headend controller, upon receiving an XAIT, can identify and determine where that particular XAIT should be transmitted on the cable system. Transmission of an XAIT from the SG to the headend controller SHALL, therefore, always be accompanied by an XAIT destination handle for that XAIT.

Optionally, the following headend information MAY be provided:

XAIT Destination Handle Description – The XAIT Destination Handle Description is intended to be a brief description indicating where the headend controller, upon receiving the XAIT message from the SG, would deliver the message on the OOB. Inclusion of this description may make it easier for the SG operator to identify which handle, from a list of handles that may be provided by the headend controller, to associate with an XAIT. The description is not required.

6.3.1 XAIT Command and Control

It is intended that the headend controller would provide the server side of a Web Service for which the operations listed below would be made available. An SG would act as a client to the Web Service through which these operations would be invoked. The headend controller SHALL return a single response to every operation and SHALL NOT send an unsolicited response. See Annex B for a list of return status values and descriptions for each of the operations that follow.

6.3.1.1 storeXAIT

The storeXAIT operation SHALL be supported by both the headend controller and the SG.

The objective of the storeXAIT operation is to transmit an XAIT table from the SG to the headend controller and for the headend controller to store the XAIT. For convenience, the storeXAIT operation SHALL also specify whether the headend controller should immediately start transmission of the XAIT on the headend cable system.

The storeXAIT operation SHALL include three parameters:

- data for a single completely formatted XAIT as defined in references [OCAP] and [MHP];
- an XAIT handle assigned to that XAIT by the SG;
- a transmission request parameter.

The headend controller SHALL store the XAIT (in non-volatile memory) and associate the XAIT handle with the XAIT provided in the operation, if the XAIT handle is defined and the XAIT is well-formed. A return status value of "SUCCESS" as defined in the list of operation return status values in Annex B SHALL be returned, if the XAIT handle is defined and the XAIT is well-formed. The headend controller SHALL NOT store the XAIT and SHALL ignore the association of the XAIT handle and XAIT provided in the operation, if the XAIT handle is undefined and/or if the XAIT is not well-formed. A return status value of either "HANDLE_NOT_DEFINED" or "CORRUPT_DATA" as defined in the list of operation return status values in Annex B SHALL be returned, if the XAIT handle is undefined or the XAIT is not well-formed, respectively. The transmission request parameter SHALL indicate to the headend controller the action to take with regard to the transmission of the XAIT on the headend cable system. The transmission request parameter SHALL be set to either "ENABLED" or "DISABLED". The headend controller SHALL immediately begin transmitting the XAIT on the headend cable system, if the transmission request parameter is set to "ENABLED". The headend controller SHALL NOT transmit the XAIT on the headend cable system until further operations requesting action are received, if the transmission request parameter is set to "DISABLED".

Multiple MPEG (private) sections may be required (or used) to carry an XAIT. All sections for the XAIT SHALL be transmitted in a single storeXAIT operation, if multiple sections are required (or used) to carry an XAIT. In order to ensure (at a high level) that each section is well-formed, the headend controller SHALL verify three parameters within each section. The headend controller SHALL verify that the table_id parameter in the XAIT section is equal to 0x74, that the section_length parameter is equal to the size of the XAIT section, and that the CRC_32 parameter in the XAIT section is correct. The headend controller SHALL NOT store the XAIT provided in the operation if any of these parameters are found to be invalid in any of the sections and SHALL reply to the SG with an indication of the condition using a return status value of "CORRUPT_DATA" as defined in the list of operation return status values in Annex B.

The headend controller SHALL replace a currently stored XAIT with the XAIT in a newly-received storeXAIT operation, if the following three conditions exist: 1) the handle in the newly-received storeXAIT operation is valid (currently defined in the headend controller), 2) the XAIT currently stored is associated with the XAIT handle in the newly-received storeXAIT operation, and 3) the XAIT in the newly-received storeXAIT operation is valid (the headend controller verified that the XAIT binary data is well-formed). The headend controller SHALL delete (only the XAIT binary data is deleted, the XAIT handle is not deleted) a currently stored XAIT, if the following three conditions exist: 1) the handle in the newly-received storeXAIT operation is valid (currently defined in the headend controller), 2) the XAIT currently stored is associated with the XAIT handle in the newly-received storeXAIT operation, and 3) the XAIT in the newly-received storeXAIT operation is corrupt (the headend controller verified that the XAIT binary data is not well-formed).

6.3.1.2 setXAITTransmissionState

The setXAITTransmissionState operation SHALL be supported by both the headend controller and the SG.

The objective of the setXAITTransmissionState operation is to request the headend controller to either enable or disable transmission of an XAIT on the headend cable system.

The setXAITTransmissionState operation SHALL include two parameters:

- an XAIT handle assigned to an XAIT by the SG
- a transmission state, equal to either "ENABLED" or "DISABLED", to be set upon the XAIT associated with the XAIT handle

The headend controller SHALL reply with an indication of whether the value of the XAIT handle is defined within the headend controller and whether an XAIT associated with the XAIT handle is currently stored in the headend controller. A return status value of "SUCCESS" as defined in the list of operation return status values in Annex B SHALL be returned, if the XAIT handle is defined and is associated with a currently stored XAIT. A defined handle associated with a currently stored XAIT SHALL result in the headend controller either enabling or disabling (as indicated by the transmission state parameter) transmission of the XAIT, associated with the XAIT handle, on the headend cable system. An undefined handle or a handle defined but not associated with an XAIT currently stored within the headend controller SHALL result in the headend controller taking no action other than replying with the appropriate return status value ("HANDLE_NOT_DEFINED" or "NO_STORED_DATA_FOR_HANDLE", respectively) indicating that condition as defined in the list of operation return status values in Annex B.

There SHALL be no pre-conditions required for this operation. That is, a request to enable transmission of an XAIT SHALL result in the XAIT transmission state being set to "ENABLED" regardless of whether the previous state was "ENABLED" or "DISABLED". A request to disable transmission of an XAIT SHALL result in the XAIT transmission state being set to "DISABLED" regardless of whether the previous state was "ENABLED" or "DISABLED".

6.3.1.3 deleteXAIT

Support of the deleteXAIT operation SHALL be supported by both the headend controller and the SG.

The objective of the deleteXAIT operation is to request the headend controller to delete an XAIT currently stored in the headend controller.

The deleteXAIT operation SHALL include a single parameter: an XAIT handle assigned to an XAIT by the SG. The headend controller SHALL reply with an indication of whether the value of the XAIT handle is defined within the headend controller and whether an XAIT associated with the XAIT handle is currently stored in the headend controller. A return status of "SUCCESS" as defined in the list of operation return status values in Annex B SHALL be returned, if the XAIT handle is defined and is associated with a currently stored XAIT. A defined handle associated with a currently stored XAIT SHALL result in the headend controller disabling transmission of the XAIT, disassociating the XAIT handle with the XAIT, and deleting the XAIT from the headend controller. An

undefined handle or a handle defined but not associated with an XAIT currently stored within the headend controller SHALL result in the headend controller taking no action other than replying with the appropriate return status value ("HANDLE_NOT_DEFINED" or "NO_STORED_DATA_FOR_HANDLE", respectively) indicating that condition, as defined in the list of operation return status values in Annex B.

6.3.1.4 getSourceIds

The headend controller and the SG MAY support the getSourceIds operation.

The objective of the getSourceIds operation is to request from the headend controller a list of source IDs and source ID descriptions currently defined by the headend controller.

The SG vendor may choose to display this list of Source IDs and their associated descriptions to the SG operator for selection when defining an XAIT. Alternatively, the SG may simply require the Source ID to be manually entered. The extent of support is vendor-specific.

The getSourceIds operation SHALL include no parameters. The headend controller SHALL reply with a list of Source IDs along with a description for each Source ID, if the getSourceIds operation is supported. The Source ID list SHALL not include duplicate Source ID values. The Source ID list MAY be ordered. An operation return status of "SUCCESS" indicating that the operation is supported, as is defined in the list of operation return status values in Annex B, SHALL be returned if the headend controller supports the getSourceIds operation. An operation return status of "OPERATION_NOT_SUPPORTED" indicating that the operation is not supported, as is defined in the list of operation return status values in Annex B, SHALL be returned if the headend controller does not support the getSourceIds operation.

Because Source IDs are assigned by the headend controller and may change over time, it is recommended that the getSourceIds operation be called often to ensure that the SG operator is viewing the most current list of Source IDs.

6.3.1.5 getXAITHandles

The headend controller and the SG MAY support the getXAITHandles operation.

The objective of the getXAITHandles operation is to request from the headend controller a list of XAIT handles and XAIT handle descriptions currently defined by the headend controller.

The SG vendor may choose to display the list of XAIT handles and their associated descriptions to the SG operator for selection when defining an XAIT. Alternatively, the SG may simply require the XAIT handle to be manually entered. The extent of support is vendor-specific.

The getXAITHandles operation SHALL include no parameters. The headend controller SHALL reply with a list of XAIT handles along with a description for each XAIT handle, if the getXAITHandles operation is supported. An operation return status of "SUCCESS" indicating that the operation is supported, as is defined in the list of operation return status values in Annex B, SHALL be returned if the headend controller supports the getXAITHandles operation. An operation return status of "OPERATION_NOT_SUPPORTED" indicating that the operation is not supported, as is defined in the list of operation return status values in Annex B, SHALL be returned if the headend controller does not support the getXAITHandles operation.

6.3.1.6 getXAITTransmissionStatus

The headend controller and the SG MAY support the getXAITTransmissionStatus operation.

The objective of the getXAITTransmissionStatus operation is to request from the headend controller the transmission state of an XAIT on the headend cable system.

The getXAITTransmissionStatus operation SHALL include a single parameter: an XAIT handle assigned to an XAIT by the SG. The headend controller SHALL reply with an indication of whether the value of the XAIT handle is defined within the headend controller and whether an XAIT associated with the XAIT handle is currently stored in the headend controller. A return status value of "SUCCESS" as defined in the list of operation return status values in Annex B SHALL be returned, if the getXAITTransmissionStatus operation is supported, the XAIT handle is defined, and the XAIT handle is associated with a currently stored XAIT. A defined handle associated with a currently stored XAIT SHALL result in the headend controller replying with the transmission state of the XAIT associated with the XAIT handle. The transmission state SHALL be either "ENABLED" or "DISABLED". An undefined handle or a handle defined but not associated with an XAIT currently stored within the headend controller SHALL result in the headend controller replying with a transmission state that is undefined and an appropriate return status value ("HANDLE_NOT_DEFINED" or "NO_STORED_DATA_FOR_HANDLE", respectively) indicating that condition, as defined in the list of operation return status values in Annex B. An operation return status of "OPERATION_NOT_SUPPORTED" indicating that the operation is not supported, as is defined in the list of operation return status values in Annex B, SHALL be returned, if the headend controller does not support the getXAITTransmissionStatus operation.

6.4 CVT Support

In order for the SG to have all of the information it needs to create the CVT message, it must be provided with the following headend information:

CVT Destination Handle –The SG SHALL be provided with a handle for each CVT it creates and transmits to the headend controller, in order to support receiving multiple CVT messages from the SG.

A handle is required so that the headend controller, upon receiving a CVT, can identify and determine where that particular CVT message should be delivered. Transmission of a CVT message from the SG to the headend controller SHALL therefore always be accompanied by a CVT destination handle for that CVT.

Optionally, the following headend information may be provided:

CVT Destination Handle Description – A brief description intending to indicate where the headend controller, upon receiving the CVT message from the SG, would deliver the message on the OOB. Inclusion of this description may make it easier for the SG operator to identify which handle, from a list of handles that may be provided by the headend controller, to associate with a CVT. The description is not required.

sourceID – A sourceID MAY be required when the SG is creating a CVT as specified by reference [CCIF2.0]. A getSourceIds operation to obtain a list of Source IDs is described in 6.3.1.4 and is intended for both XAIT and CVT support.

6.4.1 CVT Command and Control

It is intended that the headend controller would provide the server side of a Web Service for which the operations listed below would be made available. An SG would act as a client to the Web Service through which these operations would be invoked. The headend controller SHALL return a single response to every operation and SHALL NOT send an unsolicited response. See Annex B for a list of return status values and descriptions for each of the operation that follow.

6.4.1.1 storeCVT

The storeCVT operation SHALL be supported by both the headend controller and the SG.

The objective of the storeCVT operation is to transmit a CVT table from the SG to the headend controller and for the headend controller to store the CVT. For convenience, the storeCVT operation SHALL also specify whether the headend controller should immediately start transmission of the CVT on the headend cable system.

The storeCVT operation SHALL include three parameters:

- data for a single, completely formatted CVT as defined in [CDL2.0];
- a CVT handle assigned to that CVT by the SG;
- a transmission request parameter.

The headend controller SHALL store the CVT (in non-volatile memory) and associate the CVT handle with the CVT provided in the operation, if the CVT handle is defined and the CVT is well-formed. A return status value of "SUCCESS" as defined in the list of operation return status values in Annex B SHALL be returned, if the XAIT handle is defined and the CVT is well-formed. The headend controller SHALL NOT store the CVT and SHALL ignore the association of the CVT handle and CVT provided in the operation, if the CVT handle is undefined and/or if the CVT is not well-formed. A return status value of either "HANDLE_NOT_DEFINED" or "CORRUPT_DATA" as defined in the list of operation return status values in Annex B SHALL be returned, if the CVT handle is undefined or the CVT is not well-formed, respectively. The transmission request parameter SHALL indicate to the headend controller the action to take with regard to the transmission of the CVT on the headend cable system. The transmission request parameter SHALL be set to either "ENABLED" or "DISABLED". The headend controller SHALL immediately begin transmitting the CVT on the headend cable system, if the transmission request parameter is set to "ENABLED". The headend controller SHALL NOT transmit the CVT on the headend cable system until further operations requesting action are received, if the transmission request parameter is set to "DISABLED".

The headend controller SHALL also verify two parameters within the CVT itself to ensure (at a high level) that the CVT is well-formed. The headend controller SHALL verify that the code_version_table_tag parameter in the CVT is equal to either 0x9F9C02 or 0x9F9C05 and that the length_field() parameter in the CVT is equal to the size of the CVT data provided in the operation. The headend controller SHALL NOT store the CVT and SHALL ignore the association of the CVT handle and CVT provided in the operation if any of these parameters are found to be invalid and SHALL reply to the SG with an indication of the condition using the appropriate return status value of "CORRUPT_DATA" as defined in the list of operation return status values in Annex B.

The headend controller SHALL replace a currently stored CVT with the CVT in a newly-received storeCVT operation, if the following three conditions exist: 1) the handle in the newly-received storeCVT operation is valid (currently defined in the headend controller), 2) the CVT currently stored is associated with the CVT handle in the newly-received storeCVT operation, and 3) the CVT in the newly-received storeCVT operation is valid (the headend controller verified that the CVT binary data is well-formed). The headend controller SHALL delete (only the CVT binary data is deleted, the CVT handle is not deleted) a currently stored CVT, if the following three conditions exist: 1) the handle in the newly-received storeCVT operation is valid (currently defined in the headend controller), 2) the CVT currently stored is associated with the CVT handle in the newly-received storeCVT operation, and 3) the CVT in the newly-received storeCVT operation is corrupt (the headend controller verified that the CVT binary data is not well-formed).

6.4.1.2 setCVTTransmissionState

The setCVTTransmissionState operation SHALL be supported by both the headend controller and the SG.

The objective of the setCVTTransmissionState operation is to request the headend controller to either enable or disable transmission of a CVT on the headend cable system.

The setCVTTransmissionState operation SHALL include two parameters:

- a CVT handle assigned to a CVT by the SG;
- a transmission state, equal to either "ENABLED" or "DISABLED", to be set upon the CVT associated with the CVT handle.

The headend controller SHALL reply with an indication of whether the value of the CVT handle is defined within the headend controller and whether a CVT associated with the CVT handle is currently stored in the headend controller. A return status value of "SUCCESS", as defined in the list of operation return status values in Annex B, SHALL be returned if the CVT handle is defined and is associated with a currently stored CVT. A defined handle associated with a currently stored CVT SHALL result in the headend controller either enabling or disabling (as indicated by the transmission state parameter) transmission of the CVT, associated with the CVT handle, on the headend cable system. An undefined handle or a handle defined but not associated with a CVT currently stored within the headend controller SHALL result in the headend controller taking no action, other than replying with the appropriate return status value ("HANDLE_NOT_DEFINED" or "NO_STORED_DATA_FOR_HANDLE", respectively) indicating that condition as defined in the list of operation return status values in Annex B.

There SHALL be no pre-conditions required for this operation. That is, a request to enable transmission of a CVT SHALL result in the CVT transmission state being set to "ENABLED", regardless of whether the previous state was "ENABLED" or "DISABLED". A request to disable transmission of a CVT SHALL result in the CVT transmission state being set to "DISABLED", regardless of whether the previous state was "ENABLED" or "DISABLED".

6.4.1.3 deleteCVT

The deleteCVT operation SHALL be supported by both the headend controller and the SG.

The objective of the deleteCVT operation is to request the headend controller to delete a CVT currently stored in the headend controller.

The deleteCVT operation SHALL include a single parameter: a CVT handle assigned to a CVT by the SG. The headend controller SHALL reply with an indication of whether the value of the CVT handle is defined within the headend controller and whether a CVT associated with the CVT handle is currently stored in the headend controller. A return status of "SUCCESS", as defined in the list of operation return status values in Annex B, SHALL be returned if the CVT handle is defined and is associated with a currently stored CVT. A defined handle associated with a currently stored CVT SHALL result in the headend controller disabling transmission of the CVT, disassociating the CVT handle with the CVT, and deleting the CVT from the headend controller. An undefined handle or a handle defined but not associated with a CVT currently stored within the headend controller SHALL result in the headend controller taking no action other than replying with the appropriate return status value ("HANDLE_NOT_DEFINED" or "NO_STORED_DATA_FOR_HANDLE", respectively) indicating that condition, as defined in the list of operation return status values in Annex B.

6.4.1.4 getCVTHandles

The headend controller and the SG MAY support the getCVTHandles operation.

The objective of the getCVTHandles operation is to request from the headend controller a list of CVT handles and CVT handle descriptions currently defined by the headend controller.

The SG vendor may choose to display the list of CVT handles and their associated descriptions to the SG operator for selection when defining a CVT. Alternatively, the SG may simply require the CVT handle to be manually entered. The extent of support is vendor-specific.

The getCVTHandles operation SHALL include no parameters. The headend controller SHALL reply with a list of CVT handles along with a description for each CVT handle, if the getCVTHandles operation is supported. An operation return status of "SUCCESS" indicating that the operation is supported, as is defined in the list of operation return status values in Annex B, SHALL be returned if the headend controller supports the getCVTHandles operation. An operation return status of "OPERATION_NOT_SUPPORTED" indicating that the operation is not supported, as is defined in the list of operation return status values in Annex B, SHALL be returned if the headend controller does not support the getCVTHandles operation.

6.4.1.5 getCVTTransmissionStatus

The headend controller and the SG MAY support the getCVTTransmissionStatus operation.

The objective of the getCVTTransmissionStatus operation is to request from the headend controller the transmission state of a CVT on the headend cable system.

The getCVTTransmissionStatus operation SHALL include a single parameter: a CVT handle assigned to a CVT by the SG. The headend controller SHALL reply with an indication of whether the value of the CVT handle is defined within the headend controller and whether a CVT associated with the CVT handle is currently stored in the headend controller. A return status value of "SUCCESS", as defined in the list of operation return status values in Annex B, SHALL be returned if the getCVTTransmissionStatus operation is supported, the CVT handle is defined, and the CVT handle is associated with a currently stored CVT. A defined handle associated with a currently stored CVT SHALL result in the headend controller replying with the transmission state of the CVT associated with the CVT handle. The transmission state SHALL be either "ENABLED" or "DISABLED". An undefined handle or a handle defined but not associated with a CVT currently stored within the headend controller SHALL result in the headend controller replying with a transmission state that is undefined and an appropriate return status value ("HANDLE_NOT_DEFINED" or "NO_STORED_DATA_FOR_HANDLE", respectively) indicating that condition, as defined in the list of operation return status values in Annex B. An operation return status of "OPERATION_NOT_SUPPORTED" indicating that the operation is not supported, as is defined in the list of operation return status values in Annex B, SHALL be returned if the headend controller does not support the getCVTTransmissionStatus operation.

Annex A Web Services Description Language (WSDL) - (Normative)

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:intf="http://www.cablelabs.com/HEControllerDownloadService"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
    targetNamespace="http://www.cablelabs.com/HEControllerDownloadService">
    <wsdl:types>
        <schema
            targetNamespace="http://www.cablelabs.com/HEControllerDownloadService"
            xmlns="http://www.w3.org/2001/XMLSchema">
            <simpleType name="TransmissionState">
                <restriction base="xsd:string">
                    <enumeration value="ENABLED" />
                    <enumeration value="DISABLED" />
                </restriction>
            </simpleType>
            <complexType name="XAITHandleInfos">
                <complexContent>
                    <extension base="intf:HandleInfos"/>
                </complexContent>
            </complexType>
            <complexType name="SourceInfo">
                <sequence>
                    <element name="sourceId" type="xsd:unsignedShort" />
                    <element name="sourceName" nillable="false"
                        type="xsd:string" />
                </sequence>
            </complexType>
            <complexType name="ArrayOfSourceInfo">
                <sequence>
                    <element name="sourceInfo" type="intf:SourceInfo"
                        minOccurs="0" maxOccurs="unbounded" />
                </sequence>
            </complexType>
            <complexType name="SourceIDInfo">
                <sequence>
                    <element name="sourceInfos" nillable="true"
                        type="intf:ArrayOfSourceInfo" />
                    <element name="status" nillable="false"
                        type="intf:returnStatus" />
                </sequence>
            </complexType>
            <complexType name="HandleInfo">
                <sequence>
                    <element name="description" nillable="true"
                        type="xsd:string" />
                    <element name="handle" type="xsd:int" />
                </sequence>
            </complexType>
            <complexType name="ArrayOfHandleInfo">
                <sequence>
                    <element name="handleInfo"
                        type="intf:HandleInfo" minOccurs="0" maxOccurs="unbounded" />
                </sequence>
            </complexType>
            <complexType name="ArrayOfByteArray">
                <sequence>
                    <element name="byteArray" type="xsd:base64Binary"
                        minOccurs="0" maxOccurs="unbounded" />
                </sequence>
            </complexType>
            <complexType name="TransmissionStatusInfo">
                <sequence>
                    <element name="state" type="intf:TransmissionState" />
                    <element name="status" nillable="false"
                        type="intf:returnStatus" />
                </sequence>
            </complexType>
            <complexType name="HandleInfos">
                <sequence>
                    <element name="handles" nillable="true"
                        type="intf:ArrayOfHandleInfo" />
```

```
                <element name="status" nillable="false"
                    type="intf:returnStatus" />
            </sequence>
        </complexType>
        <complexType name="CVTHandleInfos">
            <complexContent>
                <extension base="intf:HandleInfos"/>
            </complexContent>
        </complexType>
        <simpleType name="returnStatus">
            <restriction base="xsd:string">
                <enumeration value="SUCCESS" />
                <enumeration value="HANDLE_NOT_DEFINED" />
                <enumeration value="NO_STORED_DATA_FOR_HANDLE" />
                <enumeration value="CORRUPT_DATA" />
                <enumeration value="OPERATION_NOT_SUPPORTED" />
                <enumeration value="CONTROLLER_ERROR" />
            </restriction>
        </simpleType>
        <complexType name="TransmissionRequest">
            <sequence>
                <element name="handle" nillable="false"
                    type="xsd:int" />
                <element name="transmissionState" nillable="false"
                    type="intf:TransmissionState" />
            </sequence>
        </complexType>
        <complexType name="XAITStoreRequest">
            <sequence>
                <element name="sections" nillable="false"
                    type="intf:ArrayOfByteArray" />
                <element name="transmissionRequest" nillable="false"
                    type="intf:TransmissionRequest" />
            </sequence>
        </complexType>
        <complexType name="CVTStoreRequest">
            <sequence>
                <element name="cvt" nillable="false"
                    type="xsd:base64Binary" />
                <element name="transmissionRequest" nillable="false"
                    type="intf:TransmissionRequest" />
            </sequence>
        </complexType>
        <element name="returnStatus" type="intf:returnStatus" />
        <element name="XAITStoreRequest" type="intf:XAITStoreRequest" />
        <element name="XAITTransmissionRequest" type="intf:TransmissionRequest" />
        <element name="TransmissionStatusInfo" type="intf:TransmissionStatusInfo" />
        <element name="XAITHandleInfos" type="intf:XAITHandleInfos" />
        <element name="SourceIDInfo" type="intf:SourceIDInfo" />
        <element name="CVTStoreRequest" type="intf:CVTStoreRequest" />
        <element name="CVTTransmissionRequest" type="intf:TransmissionRequest" />
        <element name="CVTHandleInfos" type="intf:CVTHandleInfos" />
        <element name="xaitHandleForStatus" type="xsd:int" />
        <element name="cvtHandleForStatus" type="xsd:int" />
        <element name="xaitHandleToDelete" type="xsd:int" />
        <element name="cvtHandleToDelete" type="xsd:int" />
        <element name="SourceIdsRequest">
          <complexType/>
        </element>
        <element name="XAITHandlesRequest">
          <complexType/>
        </element>
        <element name="CVTHandlesRequest">
          <complexType/>
        </element>


    </schema>
</wsdl:types>
<wsdl:message name="XAITStoreRequest">
    <wsdl:part name="xaitStoreRequest"
        element="intf:XAITStoreRequest" />
</wsdl:message>
<wsdl:message name="XAITStoreResponse">
    <wsdl:part name="XAITStoreReturn" element="intf:returnStatus" />
</wsdl:message>
<wsdl:message name="XAITTransmissionRequest">
    <wsdl:part name="transmissionRequest"
        element="intf:XAITTransmissionRequest" />
</wsdl:message>
<wsdl:message name="XAITTransmissionResponse">
```

```
    <wsdl:part name="XAITTransmissionReturn"
        element="intf:returnStatus" />
</wsdl:message>
<wsdl:message name="XAITTransmissionStatusRequest">
    <wsdl:part name="handle" element="intf:xaitHandleForStatus" />
</wsdl:message>
<wsdl:message name="XAITTransmissionStatusResponse">
    <wsdl:part name="XAITTransmissionStatusReturn"
        element="intf:TransmissionStatusInfo" />
</wsdl:message>
<wsdl:message name="CVTTransmissionStatusRequest">
    <wsdl:part name="handle" element="intf:cvtHandleForStatus" />
</wsdl:message>
<wsdl:message name="CVTTransmissionStatusResponse">
    <wsdl:part name="CVTTransmissionStatusReturn"
        element="intf:TransmissionStatusInfo" />
</wsdl:message>
<wsdl:message name="XAITDeleteRequest">
    <wsdl:part name="handle" element="intf:xaitHandleToDelete" />
</wsdl:message>
<wsdl:message name="XAITDeleteResponse">
    <wsdl:part name="XAITDeleteReturn" element="intf:returnStatus" />
</wsdl:message>
<wsdl:message name="XAITHandlesRequest">
    <wsdl:part name="dummyPart" element="intf:XAITHandlesRequest" />
</wsdl:message>
<wsdl:message name="XAITHandlesResponse">
    <wsdl:part name="XAITHandlesReturn"
        element="intf:XAITHandleInfos" />
</wsdl:message>
<wsdl:message name="SourceIdsRequest">
    <wsdl:part name="dummyPart" element="intf:SourceIdsRequest" />
</wsdl:message>
<wsdl:message name="SourceIdsResponse">
    <wsdl:part name="SourceIdsReturn" element="intf:SourceIDInfo" />
</wsdl:message>
<wsdl:message name="CVTTransmissionRequest">
    <wsdl:part name="transmissionRequest"
        element="intf:CVTTransmissionRequest" />
</wsdl:message>
<wsdl:message name="CVTTransmissionResponse">
    <wsdl:part name="CVTTransmissionReturn"
        element="intf:returnStatus" />
</wsdl:message>
<wsdl:message name="CVTDeleteRequest">
    <wsdl:part name="handle" element="intf:cvtHandleToDelete" />
</wsdl:message>
<wsdl:message name="CVTDeleteResponse">
    <wsdl:part name="CVTDeleteReturn" element="intf:returnStatus" />
</wsdl:message>
<wsdl:message name="CVTStoreRequest">
    <wsdl:part name="cvtStoreRequest"
        element="intf:CVTStoreRequest" />
</wsdl:message>
<wsdl:message name="CVTStoreResponse">
    <wsdl:part name="CVTStoreReturn" element="intf:returnStatus" />
</wsdl:message>
<wsdl:message name="CVTHandlesRequest">
    <wsdl:part name="dummyPart" element="intf:CVTHandlesRequest" />
</wsdl:message>
<wsdl:message name="CVTHandlesResponse">
    <wsdl:part name="CVTHandlesReturn"
        element="intf:CVTHandleInfos" />
</wsdl:message>
<wsdl:portType name="HEControllerDownloadServicePortType">
    <wsdl:operation name="storeXAIT">
        <wsdl:input name="XAITStoreRequest"
            message="intf:XAITStoreRequest" />
        <wsdl:output name="XAITStoreResponse"
            message="intf:XAITStoreResponse" />
    </wsdl:operation>
    <wsdl:operation name="setXAITTransmissionState">
        <wsdl:input name="XAITTransmissionRequest"
            message="intf:XAITTransmissionRequest" />
        <wsdl:output name="XAITTransmissionResponse"
            message="intf:XAITTransmissionResponse" />
    </wsdl:operation>
    <wsdl:operation name="getXAITTransmissionStatus">
        <wsdl:input name="XAITTransmissionStatusRequest"
            message="intf:XAITTransmissionStatusRequest" />
        <wsdl:output name="XAITTransmissionStatusResponse"
```

```
            message="intf:XAITTransmissionStatusResponse" />
        </wsdl:operation>
        <wsdl:operation name="deleteXAIT">
            <wsdl:input name="XAITDeleteRequest"
                message="intf:XAITDeleteRequest" />
            <wsdl:output name="XAITDeleteResponse"
                message="intf:XAITDeleteResponse" />
        </wsdl:operation>
        <wsdl:operation name="getCVTTransmissionStatus">
            <wsdl:input name="CVTTransmissionStatusRequest"
                message="intf:CVTTransmissionStatusRequest" />
            <wsdl:output name="CVTTransmissionStatusResponse"
                message="intf:CVTTransmissionStatusResponse" />
        </wsdl:operation>
        <wsdl:operation name="getXAITHandles">
            <wsdl:input name="XAITHandlesRequest"
                message="intf:XAITHandlesRequest" />
            <wsdl:output name="XAITHandlesResponse"
                message="intf:XAITHandlesResponse" />
        </wsdl:operation>
        <wsdl:operation name="getSourceIds">
            <wsdl:input name="SourceIdsRequest"
                message="intf:SourceIdsRequest" />
            <wsdl:output name="SourceIdsResponse"
                message="intf:SourceIdsResponse" />
        </wsdl:operation>
        <wsdl:operation name="setCVTTransmissionState">
            <wsdl:input name="CVTTransmissionRequest"
                message="intf:CVTTransmissionRequest" />
            <wsdl:output name="CVTTransmissionResponse"
                message="intf:CVTTransmissionResponse" />
        </wsdl:operation>
        <wsdl:operation name="deleteCVT">
            <wsdl:input name="CVTDeleteRequest"
                message="intf:CVTDeleteRequest" />
            <wsdl:output name="CVTDeleteResponse"
                message="intf:CVTDeleteResponse" />
        </wsdl:operation>
        <wsdl:operation name="storeCVT">
            <wsdl:input name="CVTStoreRequest"
                message="intf:CVTStoreRequest" />
            <wsdl:output name="CVTStoreResponse"
                message="intf:CVTStoreResponse" />
        </wsdl:operation>
        <wsdl:operation name="getCVTHandles">
            <wsdl:input name="CVTHandlesRequest"
                message="intf:CVTHandlesRequest" />
            <wsdl:output name="CVTHandlesResponse"
                message="intf:CVTHandlesResponse" />
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="HEControllerDownloadServiceSoapBinding"
        type="intf:HEControllerDownloadServicePortType">
        <soap:binding style="document"
            transport="http://schemas.xmlsoap.org/soap/http" />
        <wsdl:operation name="storeXAIT">
            <soap:operation />
            <wsdl:input>
                <soap:body use="literal" />
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal" />
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="setXAITTransmissionState">
            <soap:operation />
            <wsdl:input>
                <soap:body use="literal" />
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal" />
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="getXAITTransmissionStatus">
            <soap:operation />
            <wsdl:input>
                <soap:body use="literal" />
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal" />
            </wsdl:output>
```

```
        </wsdl:operation>
        <wsdl:operation name="deleteXAIT">
            <soap:operation />
            <wsdl:input>
                <soap:body use="literal" />
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal" />
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="getCVTTransmissionStatus">
            <soap:operation />
            <wsdl:input>
                <soap:body use="literal" />
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal" />
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="getXAITHandles">
            <soap:operation />
            <wsdl:input>
                <soap:body use="literal" />
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal" />
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="getSourceIds">
            <soap:operation />
            <wsdl:input>
                <soap:body use="literal" />
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal" />
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="setCVTTransmissionState">
            <soap:operation />
            <wsdl:input>
                <soap:body use="literal" />
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal" />
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="deleteCVT">
            <soap:operation />
            <wsdl:input>
                <soap:body use="literal" />
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal" />
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="storeCVT">
            <soap:operation />
            <wsdl:input>
                <soap:body use="literal" />
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal" />
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="getCVTHandles">
            <soap:operation />
            <wsdl:input>
                <soap:body use="literal" />
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal" />
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>


  <wsdl:service name="HEControllerDownloadService">
    <wsdl:documentation>Headend Controller Download Service</wsdl:documentation>
    <wsdl:port name="HEControllerDownloadServicePort"
               binding="intf:HEControllerDownloadServiceSoapBinding">
      <soap:address location="http://localhost/services/HEControllerDownloadService"/>
```

```
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Annex B Web Service Operation Return Status Values

Table B–1 - Return Status Values

	SUCCESS	HANDLE_NOT_DEFINED	NO_STORED_DATA_FOR_HANDLE	CORRUPT_DATA	OPERATION_NOT_SUPPORTED	CONTROLLER_ERROR
storeXAIT	x	x		x		x
setXAITTransmissionState	x	x	x			x
deleteXAIT	x	x	x			x
getXAITHandles	x				x	x
getXAITTransmissionStatus	x	x	x		x	x
getSourceIds	x				x	x
storeCVT	x	x		x		x
setCVTTransmissionState	x	x	x			x
deleteCVT	x	x	x			x
getCVTHandles	x				x	x
getCVTTransmissionStatus	x	x	x		x	x

Return Status Value Description

SUCCESS: The requested operation was successful.

HANDLE_NOT_DEFINED: The requested operation was not successful. The value of the handle provided in the operation was not defined on the headend controller.

NO_STORED_DATA_FOR_HANDLE: The requested operation was not successful. No stored data in the headend controller was available to act upon.

CORRUPT DATA: The requested operation was not successful. Some portion of the supplied XAIT or CVT binary data failed the headend controller's validation process.

OPERATION_NOT_SUPPORTED: The requested operation was not successful. The requested operation is optional and not supported by the headend controller.

CONTROLLER ERROR: The requested operation was not successful. A software or hardware error occurred on the headend controller, which prevented successful completion of the operation.

Annex C Message Protocol Flow (Normative)

The figure below is a normative example of the message exchange between the Stream Generator and the headend controller.

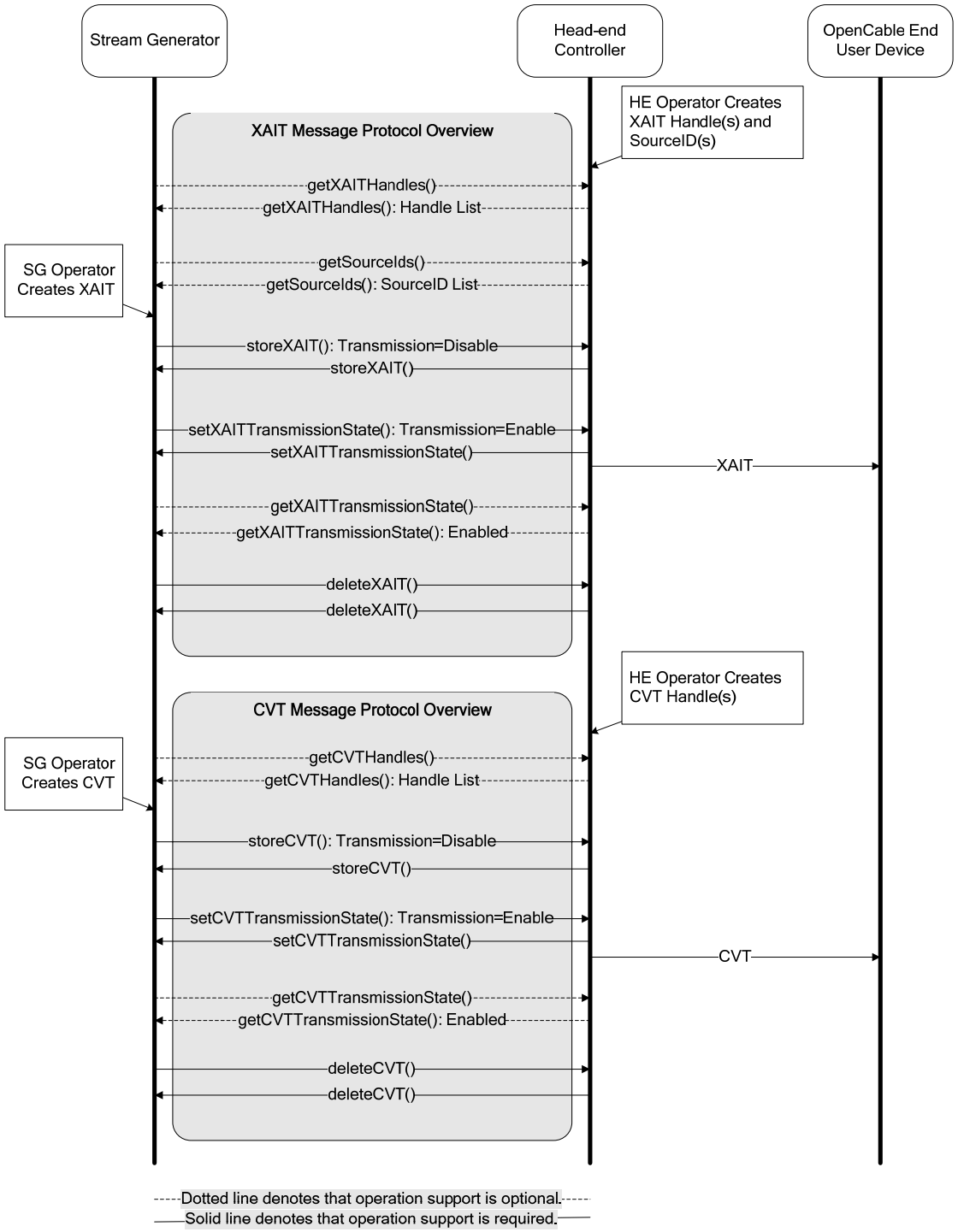

Figure C–1 - Message Protocol Flow

Appendix I Cable System Configuration (Informative)

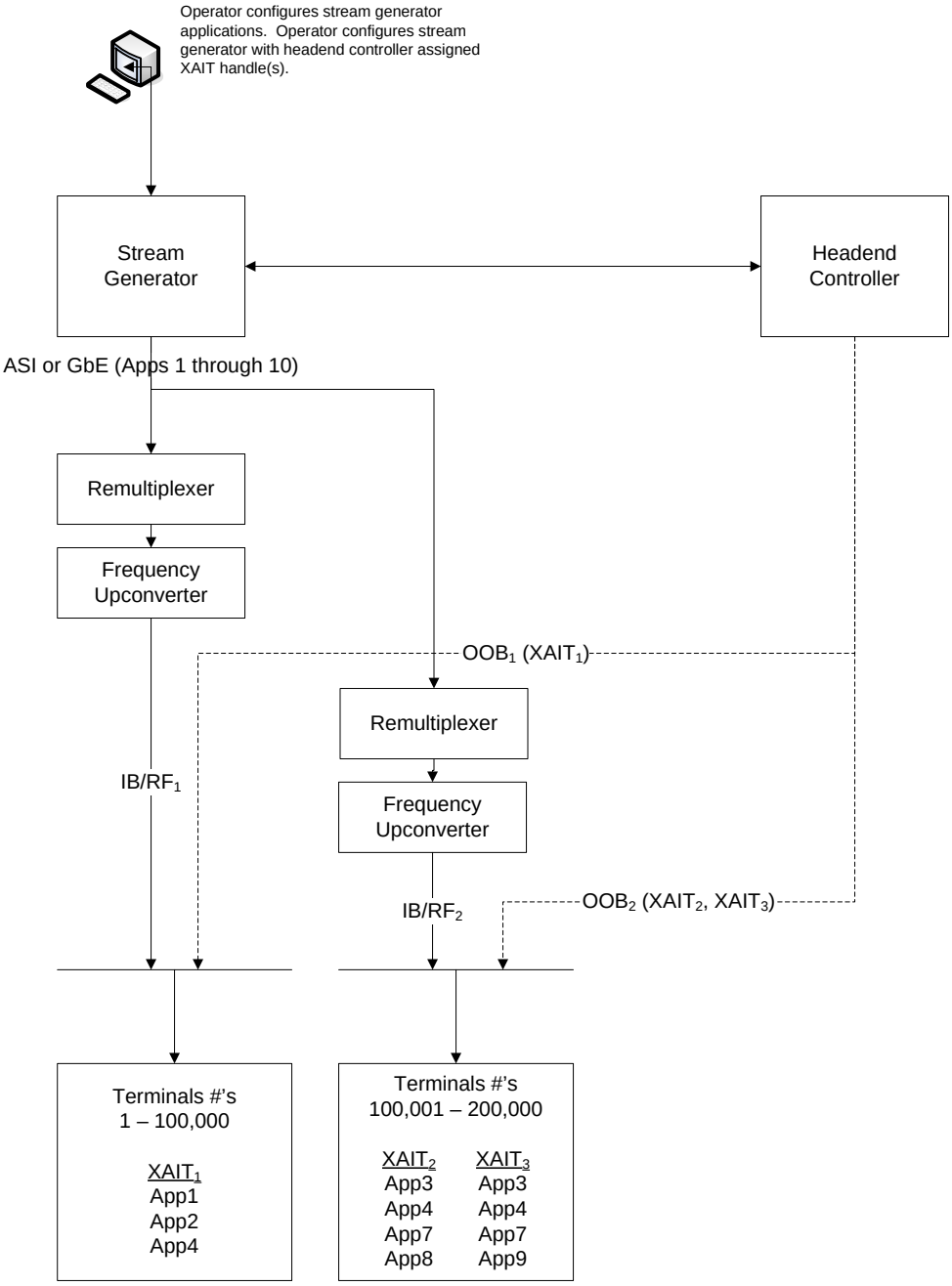

Figure I–1 - Cable System configured with multiple XAITs originating from a single Stream Generator

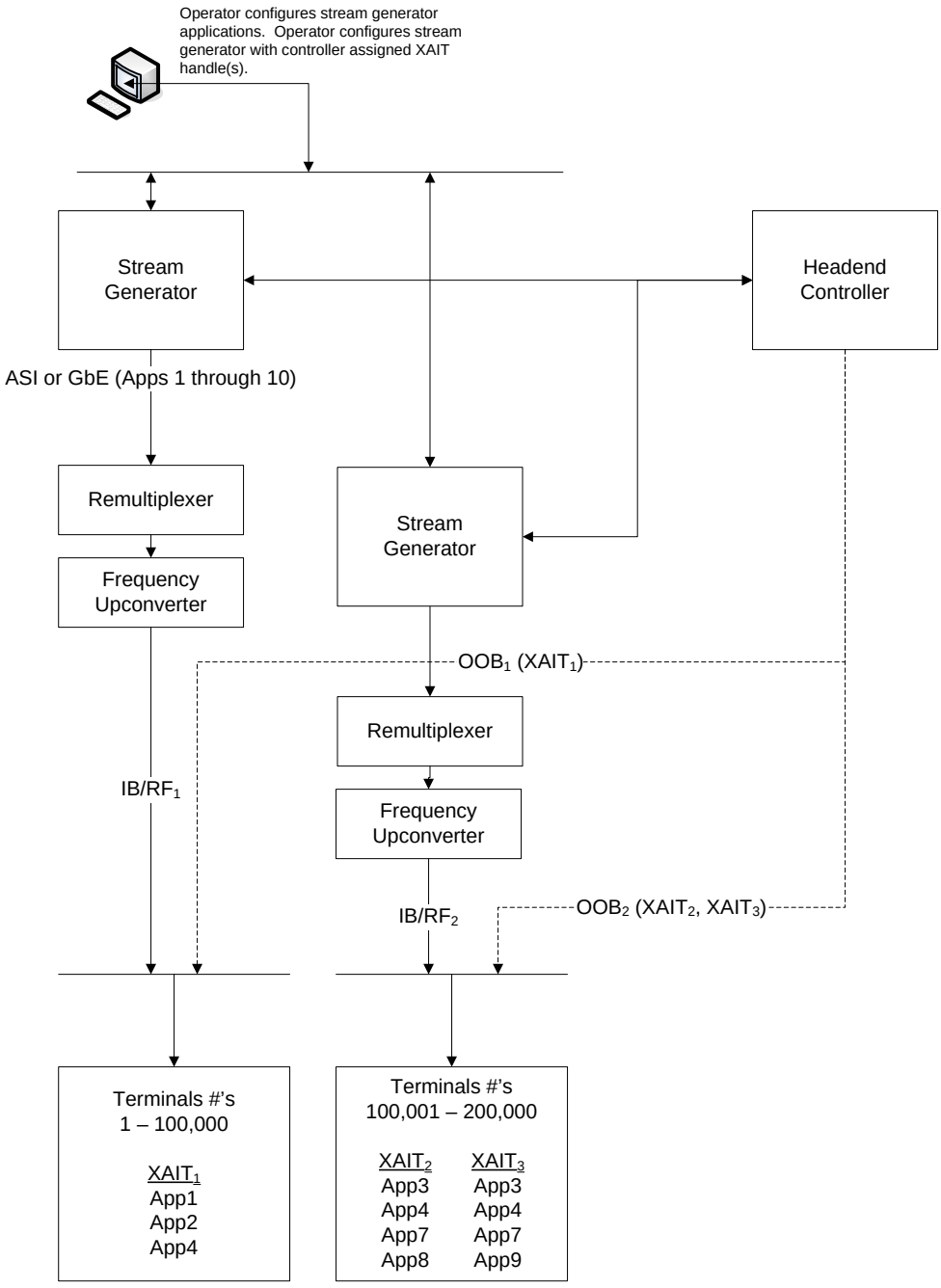

Figure I–2 - Cable system configured with multiple XAITs originating from multiple Stream Generators

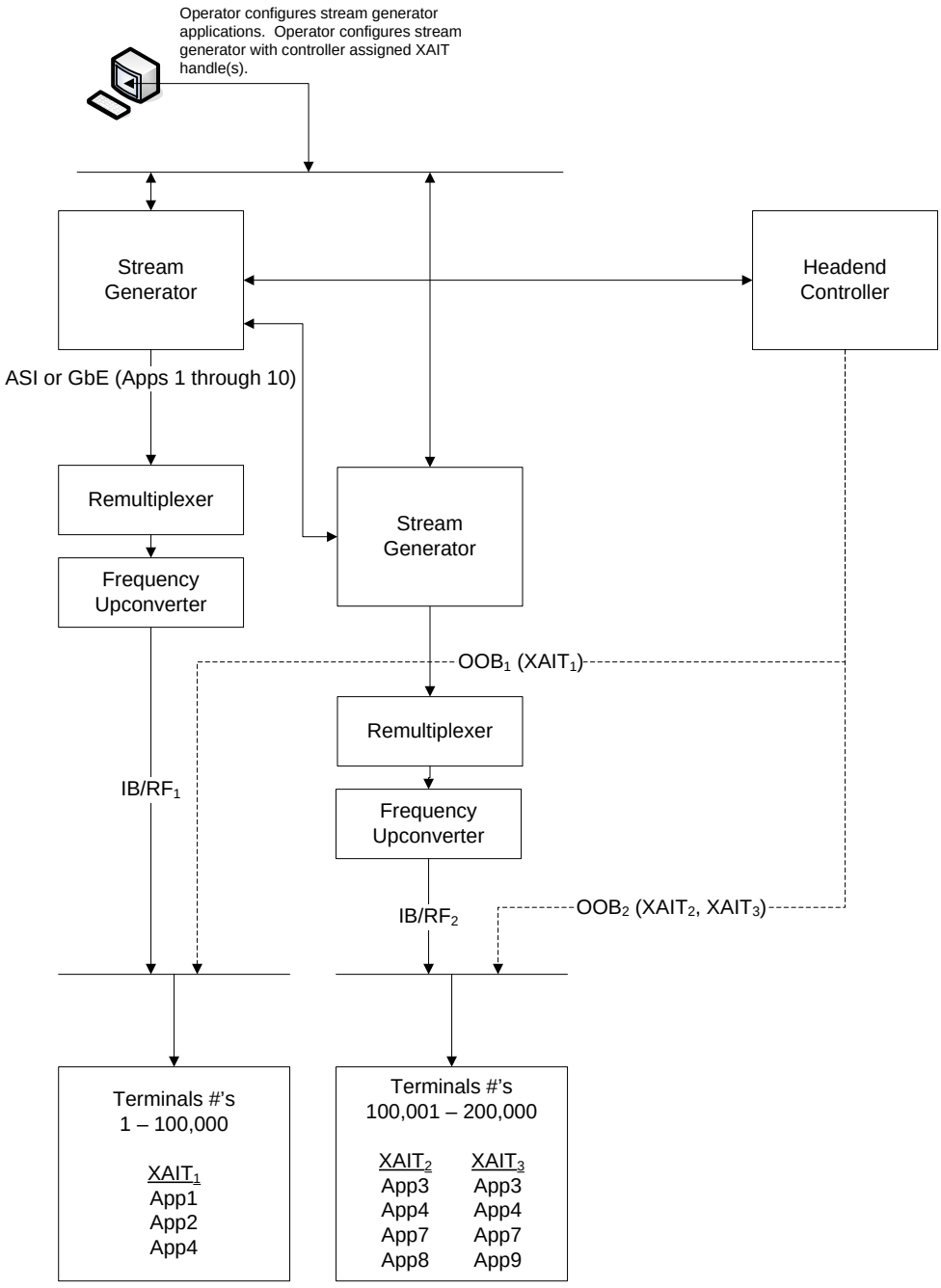

Figure I–3 - Cable system configured with multiple XAITs originating from a single OpenCable Stream Generator*

* This configuration suggests that XAIT data from one Stream Generator is merged with XAIT data from other Stream Generators (by a Stream Generator) into the XAITs transmitted to the headend controller. Merging of XAIT data may be done by a Stream Generator or some other device. The intent is to show that the headend controller is simply the conduit by which XAITs are placed onto the cable system and is not responsible for merging XAIT data from multiple Stream Generators into either a single or multiple XAITs.

Appendix II Acknowledgments

CableLabs wishes to thank Steve Iaquinto of CCAD, Adrian Fowkes of Strategy and Technology, LTD, and Adam Mauger of Softel-USA for their contributions to this specification.