ISO/IEC 29341-15-10

Edition 1.0 2011-09

INTERNATIONAL STANDARD

colour inside

Information technology – UPnP device architecture –
Part 15-10: Content Synchronization Device Control Protocol – Synchronization Service

ISO/IEC 29341-15-10

Edition 1.0 2011-09

INTERNATIONAL STANDARD

colour inside

Information technology – UPnP device architecture – Part 15-10: Content Synchronization Device Control Protocol – Synchronization Service

INTERNATIONAL ELECTROTECHNICAL COMMISSION

PRICE CODE X

ICS 35.200

ISBN 978-2-88912-655-2

CONTENTS

1 Overview and Scope....4
1.1 Introduction....4
1.1.1 ContentSync Function....6
1.1.2 Media Server Device and ContentDirectory Service....6
1.2 Notation....7
1.2.1 Data Types....8
1.3 Vendor-defined Extensions....8
1.4 Namespace for ContentSync Service....8
1.5 References....8
2 Service Modeling Definitions....10
2.1 ServiceType....10
2.2 Terms....10
2.2.1 Synchronization Object and Pair....10
2.2.2 Synchronization Data Structure....11
2.2.3 Synchronization Policy and Behavior....12
2.2.4 Minimally Complete Synchronization Relationship Data Structure....15
2.3 Synchronization Data Structure Management....16
2.3.1 Synchronization Data Structure Addition....16
2.3.2 Synchronization Data Structure Modification....16
2.3.3 Synchronization Data Structure Deletion....17
2.4 Synchronization Operation (CDS to CDS)....17
2.5 Synchronization Operation (CDS to non CDS)....19
2.6 Garbage Collection....20
2.7 State Variables....20
2.7.1 *SyncChange*....21
2.7.2 *SyncStatusUpdate*....22
2.7.3 *A_ARG_TYPE_ActionCaller*....28
2.7.4 *A_ARG_TYPE_SyncData*....28
2.7.5 *A_ARG_TYPE_SyncPair*....33
2.7.6 *A_ARG_TYPE_SyncID*....33
2.7.7 *A_ARG_TYPE_ObjectID*....33
2.7.8 *A_ARG_TYPE_SyncStatus*....33
2.7.9 *A_ARG_TYPE_ChangeLog*....34
2.7.10 *A_ARG_TYPE_Index*....35
2.7.11 *A_ARG_TYPE_Count*....35
2.7.12 *A_ARG_TYPE_ResetObjectList*....35
2.8 Eventing and Moderation....36
2.9 Actions....36
2.9.1 *AddSyncData()*....37
2.9.2 *ModifySyncData()*....38
2.9.3 *DeleteSyncData()*....40
2.9.4 *GetSyncData()*....41
2.9.5 *ExchangeSyncData()*....42
2.9.6 *AddSyncPair()*....43
2.9.7 *ModifySyncPair()*....44

2.9.8 DeleteSyncPair() 45
2.9.9 StartSync() 47
2.9.10 AbortSync() 48
2.9.11 GetChangeLog() 49
2.9.12 49
2.9.13 ResetChangeLog() 50
2.9.14 ResetStatus() 51
2.9.15 51
2.9.16 GetSyncStatus() 52
2.9.17 Non-Standard Actions Implemented by a UPnP Vendor 53
2.9.18 Common Error Codes 53
2.10 Theory of Operation 54
2.10.1 Introduction 54
2.10.2 CDS Synchronization 54
2.10.3 Synchronization of a Reference Object 72
3 XML Service Description 77
4 Test 82
Annex A (normative) AV Working Committee Properties 83
A.1 Base Properties Overview 83
A.1.1 @id 83
A.2 Resource Encoding Characteristics Properties 83
A.2.1 res@avcs:syncAllowed 83
A.2.2 res@avcs:resModified 83
A.3 Content Synchronization-related Properties 84
A.3.1 avcs:syncable 84
A.3.2 avcs:syncInfo 84
A.3.3 avcs:syncInfo@updateID 85
A.3.4 avcs:syncInfo::pair 85
A.3.5 avcs:syncInfo::pair@syncRelationshipID 85
A.3.6 avcs:syncInfo::pair@partnershipID 85
A.3.7 avcs:syncInfo::pair@pairGroupID 86
A.3.8 avcs:syncInfo::pair::remoteObjID 86
A.3.9 avcs:syncInfo::pair::remoteParentObjID 86
A.3.10 avcs:syncInfo::pair::virtualRemoteParentObjID 86
A.3.11 avcs:syncInfo::pair::policy 86
A.3.12 avcs:syncInfo::pair::status 87
Annex B (normative) Syncable Objects and Properties 88
B.1 Deciding Syncability of CDS Object 88
B.2 Synchronization of CDS object properties (Informative) 89

Figure 1 — Content Synchronization Model 4
Figure 2 — High-level Synchronization Flow Diagram 5
Figure 3 — Unidirectional Contents Synchronization 6
Figure 4 — ContentSync Function 6
Figure 5 — Types of Synchronization Pair 11
Figure 6 — Synchronization Data Structure 12
Figure 7 — Interaction Diagram (Synchronization Operation) 17

Figure 8 — Synchronization Relationship between two CDSs 55
Figure 9 — Synchronization Relationship between two CDSs 73

Table 1-1 — Namespace Definitions 8
Table 2-1 — State Variables 20
Table 2-2 — Status Codes of Synchronization Operation...27
Table 2-3 — Event Moderation..36
Table 2-4 — Actions 37
Table 2-5 — Arguments for AddSyncData()..38
Table 2-6 — Error Codes for AddSyncData()..38
Table 2-7 — Arguments for ModifySyncData()..39
Table 2-8 — Error Codes for ModifySyncData()..40
Table 2-9 — Arguments for DeleteSyncData() 41
Table 2-10 — Error Codes for DeleteSyncData() 41
Table 2-11 — Arguments for GetSyncData() 41
Table 2-12 — Error Codes for GetSyncData() 42
Table 2-13 — Arguments for ExchangeSyncData()..42
Table 2-14 — Error Codes for ExchangeSyncData()..42
Table 2-15 — Arguments for AddSyncPair() 44
Table 2-16 — Error Codes for AddSyncPair() 44
Table 2-17 — Arguments for ModifySyncPair()..45
Table 2-18 — Error Codes for ModifySyncPair() 45
Table 2-19 — Arguments for DeleteSyncPair() 46
Table 2-20 — Error Codes for DeleteSyncPair() 46
Table 2-21 — Arguments for StartSync()...47
Table 2-22 — Error Codes for StartSync()..48
Table 2-23 — Arguments for AbortSync()...48
Table 2-24 — Error Codes for AbortSync()..49
Table 2-25 — Arguments for GetChangeLog()..50
Table 2-26 — Error Codes for GetChangeLog()..50
Table 2-27 — Arguments for ResetChangeLog() 51
Table 2-28 — Error Codes for ResetChangeLog() 51
Table 2-29 — Arguments for ResetStatus() 51
Table 2-30 — Error Codes for ResetStatus() 52
Table 2-31 — Arguments for GetSyncStatus()..52
Table 2-32 — Error Codes for GetSyncStatus()..53
Table 2-33 — Common Error Codes 53
Table 2-34 — Actions for example sequence..56
Table A.1 — Content Synchronization-related Properties Overview 84
Table B.1 — Syncability of CDS object class...88
Table B.2 — Syncability of CDS Object property 90

INFORMATION TECHNOLOGY – UPNP DEVICE ARCHITECTURE –

Part 15-10: Content Synchronization Device Control Protocol – Synchronization Service

FOREWORD

1) ISO (International Organization for Standardization) and IEC (International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards. Their preparation is entrusted to technical committees; any ISO and IEC member body interested in the subject dealt with may participate in this preparatory work. International governmental and non-governmental organizations liaising with ISO and IEC also participate in this preparation.

2) In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

3) The formal decisions or agreements of IEC and ISO on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC and ISO member bodies.

4) IEC, ISO and ISO/IEC publications have the form of recommendations for international use and are accepted by IEC and ISO member bodies in that sense. While all reasonable efforts are made to ensure that the technical content of IEC, ISO and ISO/IEC publications is accurate, IEC or ISO cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.

5) In order to promote international uniformity, IEC and ISO member bodies undertake to apply IEC, ISO and ISO/IEC publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any ISO/IEC publication and the corresponding national or regional publication should be clearly indicated in the latter.

6) ISO and IEC provide no marking procedure to indicate their approval and cannot be rendered responsible for any equipment declared to be in conformity with an ISO/IEC publication.

7) All users should ensure that they have the latest edition of this publication.

8) No liability shall attach to IEC or ISO or its directors, employees, servants or agents including individual experts and members of their technical committees and IEC or ISO member bodies for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication of, use of, or reliance upon, this ISO/IEC publication or any other IEC, ISO or ISO/IEC publications.

9) Attention is drawn to the normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.

10) Attention is drawn to the possibility that some of the elements of this International Standard may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

International Standard ISO/IEC 29341-15-10 was prepared by UPnP Forum Steering committee[1], was adopted, under the fast track procedure, by subcommittee 25: Interconnection of information technology equipment, of ISO/IEC joint technical committee 1: Information technology.

The list of all currently available parts of the ISO/IEC 29341 series, under the general title *Information technology – UPnP device architecture*, can be found on the IEC web site.

This International Standard has been approved by vote of the member bodies, and the voting results may be obtained from the address given on the second title page.

[1] UPnP Forum Steering committee, UPnP Forum, 3855 SW 153rd Drive, Beaverton, Oregon 97006 USA. See also “Introduction”.

1 Overview and Scope

This service definition is compliant with the UPnP Device Architecture version *1.0*.

1.1 Introduction

Content Synchronization service enables two or more ContentDirectory services [CDS] to synchronize content with each other. This service also enables a UPnP control point to synchronize content with a ContentDirectory service. We refer this service as "CSS" or "ContentSync service" from hereon. If a CDS wants to support synchronization of objects and its resources with other CDSs, the implementation MUST enable this ContentSync service (CSS). CSS keeps change log as part of CDS object property that describe which CDS objects are added or modified or deleted since it has synchronized last. Since synchronization enables interaction between ContentSync services, each service has a Control Point (CP) functionality that invokes actions to other ContentSync service to achieve synchronization of contents with each other.

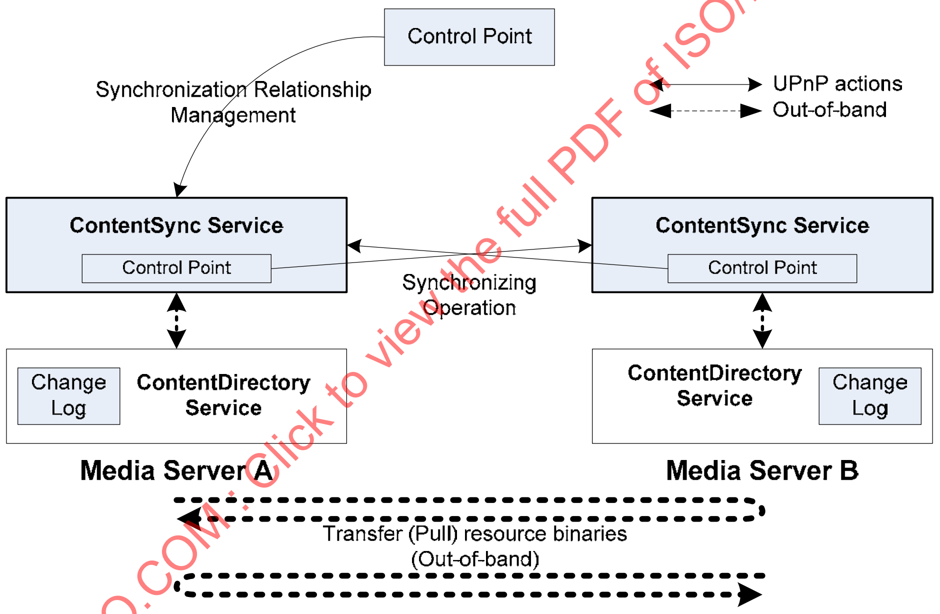

Figure 1 — Content Synchronization Model

Figure 1 shows how synchronization is accomplished between two CSSs. In the figure above, a stand alone control point is managing the synchronization between two CSSs. This includes management of content synchronization data structure (i.e., creating, browsing and deleting of synchronization data structure) and invocation of synchronization operation, etc. An embedded control point in the CSS has the role of performing the actual synchronization of objects which include retrieving the change log for objects that have changed, monitoring the status of the other CSS and updating the synchronization data structure when an object is successfully synchronized etc.

Figure 2 shows a high-level flow diagram of how Content Sync services, ContentDirectory services and Control Point interact with each other to achieve content synchronization..

Firstly, a stand-alone control point (controlled by a user) creates a synchronization relationship that describes which devices to participate in the synchronization, which objects are to be synchronized, and how to resolve conflicts and so on. When the control point creates a synchronization relationship, it MUST be responsible to define valid information for the CSS. If the synchronization relationship is successfully created, the CDS implementation that supports CSS MUST keep track of change log of the objects that are subject to synchronization. When a synchronization relationship is created between two devices, identical synchronization data structure information is maintained in both devices.

Once a synchronization relationship is created, a stand-alone control point can trigger a synchronization operation on either of CSSs. If the CSS is ready to synchronize (i.e. successfully respond to the trigger from the stand-alone control point), the embedded control point in the CSS retrieves change log from the other partner device.

After obtaining the change log, the CSS parses and interprets the change log. The CSS then updates the CDS by retrieving object information from the partner device based on the change log and the rule defined in this specification. In this step, the CSS notifies the CSS of the partner device whenever an object in the change log is dealt with regardless of success or failure. If successful update for an object is notified, the CSS implementation MUST clear the change log for that object and the CDS must keep track of new change log since this last synchronization.

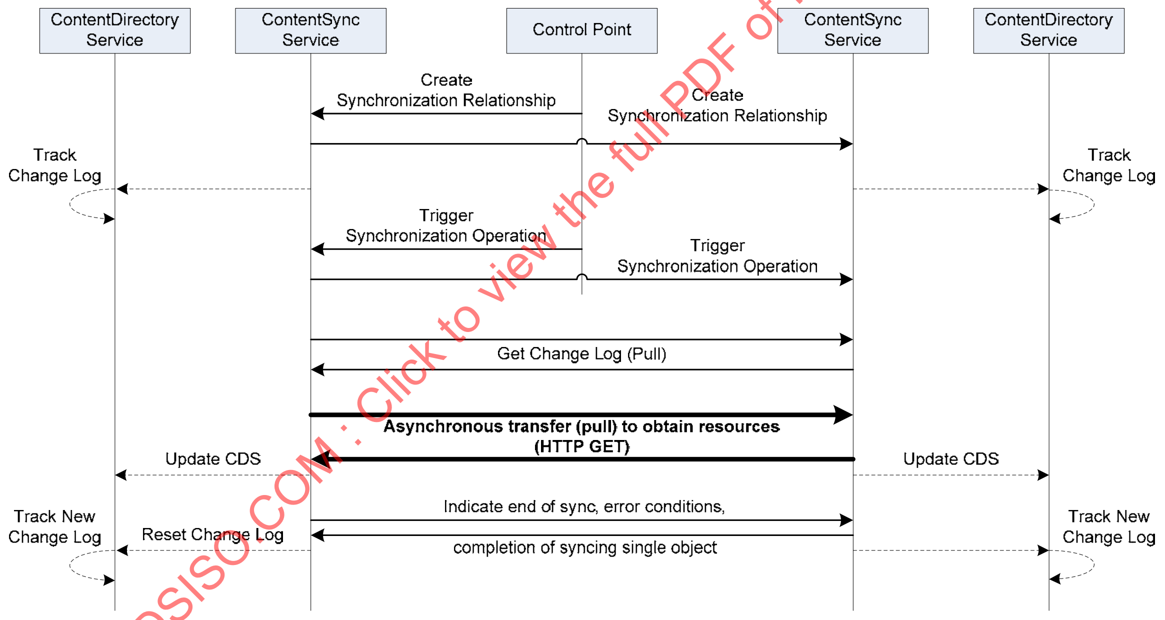
ContentDirectory Service
ContentSync Service
Control Point
ContentSync Service
ContentDirectory Service
Create Synchronization Relationship
Create Synchronization Relationship
Track Change Log
Track Change Log
Trigger Synchronization Operation
Trigger Synchronization Operation
Get Change Log (Pull)
Asynchronous transfer (pull) to obtain resources (HTTP GET)
Update CDS
Update CDS
Indicate end of sync, error conditions,
completion of syncing single object
Track New Change Log
Reset Change Log
Track New Change Log

Figure 2 — High-level Synchronization Flow Diagram

The ContentSync service also provides a functionality by which a control point can only track changes of objects that the control point is interested in. This functionality is helpful for unidirectional synchronization. Figure 3 shows such a scenario. In this scenario, a control point with its own local storage (not compliant to CDS) can synchronize with a CDS by its own local policy. In other words, the control point does not follow any policies that are defined in this specification. The control point creates synchronization relationship information on a CDS with its interest for the CDS to track some objects. The CDS keeps change log for the objects the control point is interested. Therefore, an embedded control point in the CSS is disabled for this type of synchronization. Subclause 2.5 explains in details how this kind of unidirectional synchronization can be achived.

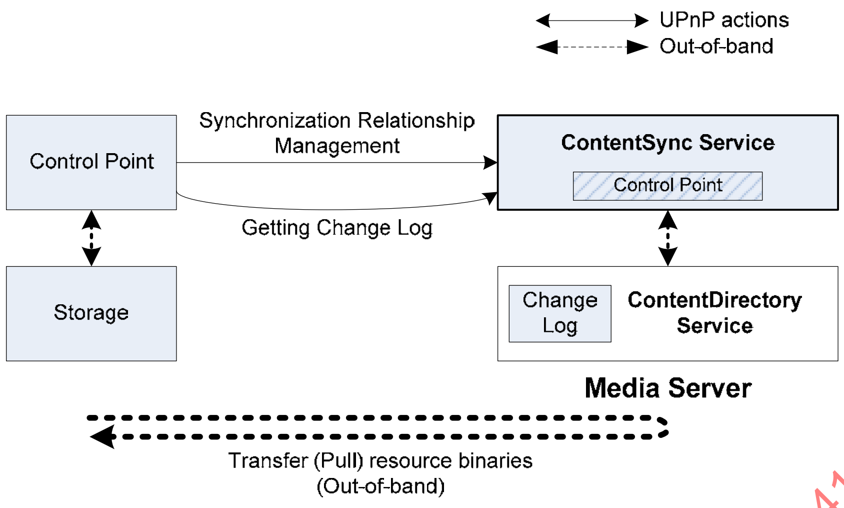

Figure 3 — Unidirectional Contents Synchronization

1.1.1 ContentSync Function

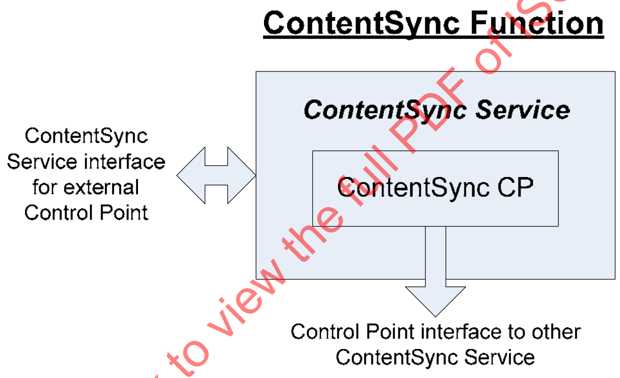
ContentSync Function
ContentSync Service
ContentSync Service interface for external Control Point
ContentSync CP
Control Point interface to other ContentSync Service

Figure 4 — ContentSync Function

The Content Sync function is an essential part of the Content Synchronization. This function is a combination of a ContentSync service and a Content Sync CP in a CSS as shown in Figure 4.

ContentSync Service:

ContentSync service is responsible for managing synchronization data structure and performing synchronization operation with a partner CSS.

ContentSync CP:

The ContentSync CP provides Control Point functionality that controls other ContentSync service running on the network.

The interface between the ContentSync CP and the ContentSync service is device-dependent and not defined by the UPnP ContentSync Service specifications.

1.1.2 Media Server Device and ContentDirectory Service

Since a ContentSync service provides the functionality to synchronize ContentDirectory service objects, ContentSync service implementation MUST appear together with ContentDirectory service implementation and MUST be also deployed on an UPnP Media Server device [MSD] that supports synchronization. Therefore, a Media Server implementation MUST expose an XML device description document which contains description of both ContentSync service and ContentDirectory service when the Media Server implementation supports synchronization of CDS objects.

The following device type identifies a Media Server device that is compliant with this specification:

urn:schemas-upnp-org:device:*MediaServer:2*

The following service type identifies a ContentDirectory service that is compliant with this specification:

urn:schemas-upnp-org:service:*ContentDirectory:2*

To enable synchronization of CDS objects, this specification imposes additional requirements on ContentDirectory:2 service specification. When supporting synchronization of CDS objects, these additional requirements MUST be implemented on top of ContentDirectory:2 service implementation. **See Annex A for the additional requirements on ContentDirectory:2 service specification (especially CDS properties of ContentDirectory:2 service).**

Additionally, since this specification adds extended properties to CDS, the AVCS XML schema [AVCS-XSD] for those properties is specified in this specification, not in UPnP AV. In other words, a CDS object expressed by original DIDL-Lite XML document MUST also refer to the AVCS XML schema when the new properties are added to the object. (The schema of the DIDL-Lite XML document does not have any reference to the AVCS XML schema). Note that the schema is informative only and hence the XML data types defined in this specification take precedence over all the XML schemas.

1.2 Notation

- In this document, features are described as Required, Recommended, or Optional as follows:

  The key words "MUST," "MUST NOT," "REQUIRED," "SHALL," "SHALL NOT," "SHOULD," "SHOULD NOT," "RECOMMENDED," "MAY," and "OPTIONAL" in this specification are to be interpreted as described in [RFC 2119].

  In addition, the following keywords are used in this specification:

  PROHIBITED – The definition or behavior is an absolute prohibition of this specification. Opposite of REQUIRED.

  CONDITIONALLY REQUIRED – The definition or behavior depends on a condition. If the specified condition is met, then the definition or behavior is REQUIRED, otherwise it is PROHIBITED.

  CONDITIONALLY OPTIONAL – The definition or behavior depends on a condition. If the specified condition is met, then the definition or behavior is OPTIONAL, otherwise it is PROHIBITED.

  These keywords are thus capitalized when used to unambiguously specify requirements over protocol and application features and behavior that affect the interoperability and security of implementations. When these words are not capitalized, they are meant in their natural-language sense.

- Strings that are to be taken literally are enclosed in "double quotes".
- Words that are emphasized are printed in *italic*.
- Keywords that are defined by the UPnP ContentSync and AV Working Committee are printed using the *forum* character style.
- Keywords that are defined by the UPnP Device Architecture are printed using the **arch** character style.
- A double colon delimiter, "::", signifies a hierarchical parent-child (parent::child) relationship between the two objects separated by the double colon. This delimiter is used

in multiple contexts, for example: Service::Action(), Action()::Argument, parentProperty::childProperty.

1.2.1 Data Types

This specification uses data type definitions from two different sources. The UPnP Device Architecture defined data types are used to define state variable and action argument data types.

For UPnP Device Architecture defined Boolean data types, it is strongly RECOMMENDED to use the value "**0**" for false, and the value "**1**" for true. However, when used as input arguments, the values "**false**", "**no**", "**true**", "**yes**" may also be encountered and MUST be accepted. Nevertheless, it is strongly RECOMMENDED that all state variables and output arguments be represented as "**0**" and "**1**".

For XML Schema defined Boolean data types, it is strongly RECOMMENDED to use the value "*0*" for false, and the value "*1*" for true. However, when used as input properties, the values "*false*", "*true*" may also be encountered and MUST be accepted. Nevertheless, it is strongly RECOMMENDED that all properties be represented as "*0*" and "*1*".

1.3 Vendor-defined Extensions

Whenever vendors create additional vendor-defined state variables, actions or properties, their assigned names and XML representation MUST follow the naming conventions and XML rules as specified in [DEVICE]

1.4 Namespace for ContentSync Service

All data types represented by XML document in this specification MUST use the following namespaces and XML schemas. Note that this schema is informative only and hence the XML data types defined in this specification take precedence over the XML schema.

Table 1-1 — Namespace Definitions

Standard Name-space Prefix	Namespace Name	Namespace Description	Normative Definition Document Reference
cs:	urn:schemas-upnp-org:cs Reference: http://www.upnp.org/schemas/cs/cs-v1-2007xxxx.xsd	Common data types for use in ContentSync schema	[CSS-XSD]
avcs:	urn:schemas-upnp-org:cs:avcs Reference: http://www.upnp.org/schemas/cs/avcs-v1-2007xxxx.xsd	Metadata for UPnP AV CDS	[AVCS-XSD]

1.5 References

[RFC 2119] – IETF RFC 2119, Key words for use in RFCs to Indicate Requirement Levels, S. Bradner, 1997.

[RFC 4122] – IETF RFC 4122, A Universally Unique Identifier (UUID) URN Namespace, P. Leach, et. al., 2005.

[CDS] – *ContentDirectory:2*, UPnP Forum, May 31, 2006.

[DIDL-LITE-XSD] – XML Schema for ContentDirectory:2 Structure and Metadata (DIDL-Lite), UPnP Forum, May 31, 2006.

[CSS-XSD] – *XML Schema for ContentSync Service:1*, UPnP Forum, July 26, 2007.

[AVCS-XSD] – XML Schema for additional CDS Object Properties of ContentSync Service:1, UPnP Forum, July 26, 2007.

[DEVICE] – *UPnP Device Architecture, version 1.0*, UPnP Forum, June 13, 2000.

[XML] – *Extensible Markup Language (XML) 1.0 (Third Edition)*, François Yergeau, Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, eds., W3C Recommendation, February 4, 2004.

[XML SCHEMA-2] – *XML Schema Part 2: Data Types, Second Edition*, Paul V. Biron, Ashok Malhotra, W3C Recommendation, 28 October 2004.

[MSD] – *MediaServer:2*, UPnP Forum, May 31, 2006.

2 Service Modeling Definitions

2.1 ServiceType

The following service type identifies a service that is compliant with this template:

urn:schemas-upnp-org:service:*ContentSync:1*.

2.2 Terms

2.2.1 Synchronization Object and Pair

A CDS object that is to be synchronized is called a synchronization object.

A synchronization pair represents a binding between a synchronization object in the local device and a synchronization object in the partner device. This binding information is stored in the *avcs:syncInfo* property of the synchronization objects. The *avcs:syncInfo* property for an object also keeps information related to which property or resource has been changed for that object since the object synchronized last with a remote object. This property MUST be updated whenever there is a change to that object. Therefore, any change to *avcs:syncInfo* property MUST not be perceived as object change. **See Annex A and Annex B for details on synchronization object property.** It is possible that an object that is new or yet to be synchronized does not have the corresponding remote object. In that case the remote object gets created in the partner device during the synchronization operation if specified by the policy. When creating a synchronization pair for an object, one of the three possible scenarios as shown in Figure 5 will occur.

- **Scenario 1:** an (local) object is paired with an existing remote object in the partner device.
- **Scenario 2:** the local object does not have a corresponding remote object in the partner device and the remote object gets created under an exsiting container object in the partner device which is designated by the control point. The existing container object here is called as *Remote Parent Object*.
- **Scenario 3:** This is similar to scenario 2, however the remote parent object under which the remote object will be created does not exist either and it gets created along with the remote object during the synchronization operation. In scenario 3, the remote parent object that will be created MUST be paired with the parent object of the local object which is called as *Virtual Remote Parent Object*.

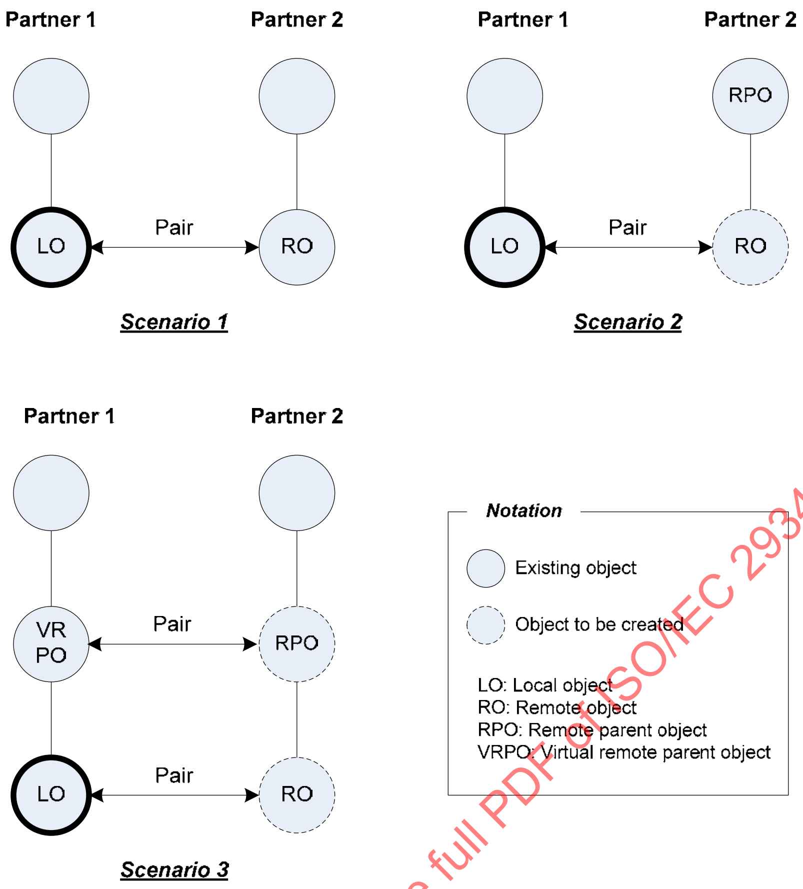

Figure 5 — Types of Synchronization Pair

The *avcs:syncInfo* property for an object can have multiple synchronization pair information if the object is paired with multiple remote objects in different devices. In such case, there are some restrictions that MUST be followed. **See 2.9.6 *AddSyncPair()* action for details.**

2.2.2 Synchronization Data Structure

A Synchronization Data Structure consists of the following information.

- **Synchronization PairGroup** is the data structure that identifies a group of synchronization pairs where identical synchronization policy will be applied. The actual synchronization pair information describing which object in the local CDS is paired with an object in the partner CDS is contained in the object itself as part of object property.
- **Synchronization Partnership** is the data structure that describes a synchronization operation between two specific CDSs. These two CDSs are called partners. A synchronization partnership contains multiple synchronization pairGroups. A synchronization partnership contains policy information that is applicable to all the pairGroups contained within that partnership. If a pairGroup has its own policy information then the pairGroup policy overrides partnership policy for that specific pairGroup.
- **Synchronization Relationship** is the data structure that describes a synchronization operation between two or more CDSs. A synchronization relationship is composed of one or more synchronization partnerships and each partnership is composed of one or more synchronization pairGroups.

Figure 6 shows an example synchronization data structure with all its components.

The synchronization data structure allows an object in one device to synchronize with an object in another device. Every syncable object in CDS has synchronization pair information that describes how the object gets synchronized with another object. **See Clause 2.2.3 Synchronization Policy for more details.**

A synchronization relationship or a partnership or a pairGroup is identified by a unique ID. Regardless of disappearance/reappearance of this service on the network, the implementations that support ContentSync service implementations MUST maintain the same value for these IDs in the CDS over its life-time. The value once used MUST be never re-used. In order to make the value of this state variable globally unique, it must be generated using GUID as defined in [RFC 4122]. A GUID is 128 bits long, and can guarantee uniqueness across space and time.

Structurally, single synchronization relationship can have multiple partnerships by definition. However, this version of the specification allows only one partnership within a synchronization relationship as shown in Figure 6. But, multiple pairGroups within a partnership are allowed in this version of the specification. For example, the synchronization relationship, S2, is only effective one in the figure below.

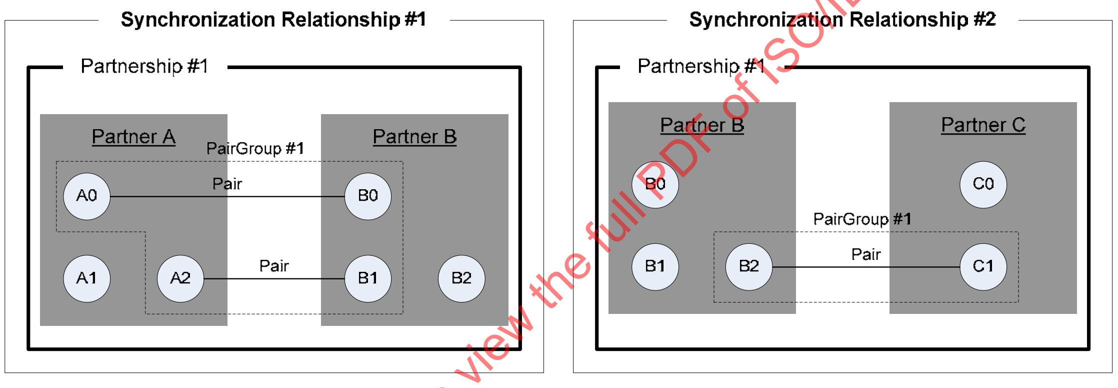

Figure 6 — Synchronization Data Structure

2.2.3 Synchronization Policy and Behavior

A synchronization policy indicates how synchronization partners that are involved in a synchronization relationship can exchange synchronization objects. In general, a synchronization policy indicates which device should provide metadata and resources to which device. There are four types of policies defined which is explained below:

2.2.3.1 "replace" synchronization policy

In "replace" synchronization policy, one of the synchronization partners becomes the source and the other becomes the sink. The purpose of "replace" synchronization is to make a sink identical to the source. That is, contents of the sink objects are replaced with the contents of the source. The terms source and sink are merely conceptual between the synchronization pair.

The behavior of "replace" synchronization policy is as follows.

- The object added to the source which does not have any corresponding object in the sink MUST be copied to the sink.
- Any modification, including deletion, to the existing objects in the source will be applied to the corresponding objects in the sink. To protect an object in the sink from deletion by synchronization, this object MUST be marked as deletion protected in the synchronization

policy. The protected object will not be deleted after synchronization, and will be excluded from the relationship.

Note1: There should not be any object in the sink in a relationship that does not have a corresponding object in the source.

Note2: Any changes in the sink are not useful, as these will be replaced by the source. To retain changes in an object in the sink, the object should be excluded from the relationship or it needs to be copied before synchronization.

2.2.3.2 “merge” synchronization policy

The “merge” synchronization policy defines that after synchronization, each partner will end up with a superset of synchronization objects of all the partners. In other words, the synchronization objects from all the partners will be merged according to the following rules:

- An object added to a synchronization partner that does not have any corresponding object in the other partner will be copied to the other partner.
- Any metadata or resources that are missing on either partner that missing data is copied to the other partner. If an object and its corresponding object have the same properties with different values, then the values of properties of the partner with higher precedence will be copied to the other partner.

2.2.3.3 “blend” synchronization policy

The “blend” synchronization policy defines that after synchronization, each partner will end up with a superset of synchronization objects of all the partners. In other words, the synchronization objects from all the partners will be blended according to the following rules:

- An object added to a synchronization partner that does not have any corresponding object in the other partner will be copied to the other partner.
- Any metadata or resources that are missing on either partner that missing data is copied to the other partner. If an object and its corresponding object have the same properties with different values, then the values are left as is on both partners.

2.2.3.4 “tracking” synchronization policy

A “tracking” synchronization policy is useful only when synchronizing between a CDS and a non-CDS device. The actual synchronization operation for this policy is out of the scope of this specification. In this policy, only the device having a CDS keeps track of the change log for synchronization objects. The device clears the change log by invocation of the *ResetChangeLog()* action and starts keeping new log from that point. The device stops keeping change log when the synchronization relationship is destroyed.

The behavior of “tracking” synchronization policy is as follows.

- A new object is automatically added to a synchronization pairGroup of its parent object if the parent object (container) of the newly added object is also a synchronization object and the *autoObjAdd* (automatic addition of new child object to synchronization pairGroup) option in the policy of the parent object is set to "1".
  - However, descendent objects except direct child object are not affected by the option above.

2.2.3.5 Deleting object from synchronization relationship

If a user wants to delete an object from the CDS permanently, the user MUST exclude that object from the relationship. The object may be created again by a synchronization operation if the object is just deleted and not excluded from the synchronization relationship. If an object is associated with multiple synchronization relationships, it is not permanently deleted until the object is excluded from all the synchronization relationships.

2.2.3.6 *policy* Data Format

The synchronization policy is included in the <*policy>* element in the *ContentSync XML document* which contains zero or more synchronization data structure (**See Clause 2.7.4 *A_ARG_TYPE_SyncData* state variable)** and in the *avcs:syncInfo::pair::policy* property in a CDS object if overriding policy is necessary (**See clause A.3 Content Synchronization-related Properties.**) The following example shows an XML fragment of the policy.

Example:

```
<policy>
 <syncType>replace</syncType>
 <priorityPartnerID>1</priorityPartnerID>
 <delProtection>1</delProtection>
<policy>
```

The (one and only) root element, *<policy>*, MUST contain zero or more elements, each of which represents a synchronization policy.

The following example shows a generalized “template” for the format of the *policy* XML document. Additional elements and/or attributes MAY be added to future versions of this specification. Furthermore, a 3rd-party vendor MAY add vendor-defined elements and/or attributes. However, by definition, this specification does not define the format and/or values for these 3rd-party elements. In order to eliminate element/attribute naming conflicts, the name of any vendor-defined element/attribute MUST follow the rules set forth in **Clause 1.3 “Vendor-defined Extensions”** All control points should gracefully ignore any element/attribute that it does not understand.

The following notation includes the *forum* character style to indicate names that are defined by the ContentSync Working Committee. Additionally, fields that need to be filled out by individual implementations are shown in the *vendor* character style.

```
<?xml version="1.0"?>
<policy
  xmlns="urn:schemas-upnp-org:cs"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:schemas-upnp-org:cs
   http://www.upnp.org/schemas/cs/cs-v1-20070XXXX.xsd">
 <syncType>synchronization policy type</syncType>syncDataUpdate
 <priorityPartnerID>Role of a partner</priorityPartnerID>
 <delProtection>Protect an object deletion</delProtection>
 <autoObjAdd>flag indicating automatic addition of new direct child to a
   pairGroup</autoObjAdd>
</policy>
```

`xml`
: OPTIONAL. Case sensitive.

`SyncChange`
: REQUIRED. MUST have “urn:schemas-upnp-org:av:css-event” (which is the UPnP ContentSync Working Committee Schema) as the value for the xmlns attribute that declares the default namespace; Contains all elements and attributes defined by the CDS Event schema as follows:

`syncType`
: REQUIRED. Indicates whether it is a “replace” or “merge” or "blend" or "tracking" synchronization.

`priorityPartnerID`
: OPTIONAL. Indicates the role of a partner. If present, in a “replace” synchronization policy, it identifies the partner that is a source device. If present, in a “merge” synchronization policy, it identifies the partner that takes precedence on conflict. In "blend" and "tracking" synchronization, it is not applicable, therefore it could be omitted.

The value of the element MUST be the partner@id in a synchronization data structure. **See 2.7.4.1 *A_ARG_TYPE_SyncData_Data* Format for details.** Because of the definition of <partner> element in the synchronization data structure, the vaule MUST be either "1" or "2".

`delProtection`
OPTIONAL. indicates whether an object will remain or not in the CDS hierarchy even after the deletion of the object by a synchronization operation. A default behavior is that an object will be deleted by synchronization. If this property is not appeared in the policy property of the *avcs:syncInfo::pair::policy property*, the default behavior MUST be applied.

`autoObjAdd`
OPTIONAL. indicates whether a new direct child object will be automatically added into the synchronization pairGroup of the parent object.

Although there are four types of synchronization polices defined above, some properties of an objects in a CDS MUST be dealt with apart from synchronization. For example, the values of *@id* and *@parentID* are dependent on local CDS and a CDS cannot assign a new value to *@id* and *@parentID* by copying these values from other objects. They MUST NOT be considered as syncable and the changes on them MUST be ignored during the synchronization operation. These properties that are independent of synchronization are listed in the **Annex B, "Syncablility of CDS object."**

2.2.3.7 Synchronization behavior

A synchronization operation updates metadata and resource(s) of a synchronization object using change log in order to keep same metadata and resources between two objects in two different CDSs. While synchronizing objects between two CDSs, the following behavior MUST be applied;

- There is a case where a CDS does not support some metadata that the other synchronization partner supports. In this situation, the CDS that does not support those metadata MUST ignore them.
- Synchronizing two objects under the *merge* synchronization policy, both of the objects will end up with a superset of all properties between these two objects. However, in case of a property that exists in both objects, the object without precedence will copy the value of the property from the object with precedence. The same rule applies to in case of multi-value property. For example, if an object has two properties of the same (i.e. multi-value property) and this object takes precedence while synchronizing with another object, then these properties will be copied to the other object after synchronization even if the other has less or more properties of the same than the object with precedence.
- In the case of the *res* property, the *res@avcs:resModified* property indicates whether the resource has been changed. Therefore, while synchronizing, an object without precedence MUST always retrieve the resource from the other object with precedence if the *res@avcs:resModified* property of the object with precedence has set to "1".
- For *replace* synchronization policy, when source object is deleted, the sink object MUST be deleted after synchronization.
  - Note that deleted object information SHOULD be provided in change log.
- For *merge* synchronization policy, when an object with precedence is deleted, it will be revived with all metadata and resource of the partner object after synchronization when change log from the partner device contains the partner object.

2.2.4 Minimally Complete Synchronization Relationship Data Structure

A minimally complete synchronization data structure defines exactly one synchronization relationship, exactly one partnership within synchronization relationship and exactly one pairGroup within that partnership. **See 2.7.4 *A_ARG_TYPE_SyncData* state variable for details.**

2.3 Synchronization Data Structure Management

This subclause describes how a synchronization data structure gets added, modified and deleted. The synchronization data structure is defined by the ***A_ARG_TYPE_SyncData*** state variable (**See Clause 2.7.4)** and any changes to the data structures get evented by the ***SyncChange*** state variable. **See 2.7.1, *SyncChange* state variable, and 2.8 "Eventing and Moderation" for details on how to send an event message for the synchronization data structure change.**

2.3.1 Synchronization Data Structure Addition

Since the same synchronization data structure is kept in the partner devices within a partnership, any addition to the existing synchronization data structure such as adding a new synchronization relationship or new a pairGroup within an existing partnership MUST abide by the following rules:

Note: a new partnership within an existing synchronization relationship is not allowed in this version of the specification.

- When adding a new synchronization relationship or adding a new pairGroup, the two partner devices MUST be in the network. When a partner leaves the network while adding a synchronization data structure, the first partner that receives this addition request MUST not update its synchronization data structure. Likewise, when a partner fails for some reason after receiving a successful response for addition from the second partner, added synchronization relationship or pairGroup in the second partner MUST be destroyed. To remove such stale data in the second partner, the second partner exchanges its own synchronization data structure with the first partner by invoking the UPnP action *ExchangeSyncData()* when the first partner comes back to the network..

2.3.2 Synchronization Data Structure Modification

Since the same synchronization data structure is kept in all the partner devices, any modifications to the existing synchronization data structure such as modification to an XML element in a synchronization relationship or in a partnership within an existing synchronization relationship or in a pairGroup within an existing partnership MUST follow the following rules:

- To modify a synchronization relationship or a partnership or a pairGroup, all partner devices MUST be in the network. When a partner leaves the network while modifying a synchronization data structure, the first partner that receives this modification request MUST not update its synchronization data structure..When a partner fails for some reason after receiving a successful response for modifications from the second partner, the modified synchronization relationship or pairGroup in the second partner MUST be destroyed. To remove such stale data in the second partner, the second partner exchanges its own synchronization data structure with the first partner by invoking the UPnP action *ExchangeSyncData()* when the first partner comes back to the network..The partner device can determine the staleness of its partnership or pairGroup data by comparing the *partnership@updateID* attribute and the *pairGroup@updateID* with the one in the other partner device, respectively. Upon creation of a synchronization data structure, all partner devices MUST keep the *partnership@updateID* and the *pairGroup@updateID* attributes that are increased by 1 whenever a change is made on the partnership or pairGroup that the partner belongs to. If the values of the *partnership@updateID* or *pairGroup@updateID* are different then the partnership information with higher value of *partnership@updateID* or *pairGroup@updateID* is up-to-date, and the partner with the lower value MUST update its partnership or pairGroup information with the one from the other partner. After update, the *partnership@updateID* and *pairGroup@updateID* values on both the partners become identical.
- The change of the partner device in a partnership is NOT allowed.
- A device which is currently processing a modification request MUST reject any subsequent modification requests on the same data structure or part of the data structure

that is the target of the current modification request which is in progress. When a device is performing a synchronization operation, any modification request on the associated data structure MUST be rejected.

2.3.3 Synchronization Data Structure Deletion

Any deletions to an existing data structure MUST follow the following rules:

- When a synchronization relationship or a partnership or a pairGroup is deleted from an existing data structure, the changed data structure after the deletion MUST be synchronized among all the partners.
- Any deletions in one of the partners are allowed. However, the deleted information MUST be synchronized when the other partner come to the network.
- When a synchronization relationship is deleted, all related information stored in the device such as partnership, pairGroup, synchronization pair and deleted object list relevant to the relationship MUST be deleted.
- After a partnership is deleted, all information associated with this partnership such as pairGroup, synchronization pair and deleted object list MUST be deleted.
- When a pairGroup is deleted, all related information stored in the device such as synchronization pair and deleted object list relevant to the pairGroup MUST be deleted.
- When the last pairGroup within an existing partnership is deleted, the partnership MUST be deleted as well because the synchronization data structure does not allow a partnership without at least one pairGroup.

2.4 Synchronization Operation (CDS to CDS)

A Synchronization operation is performed according to the policies described in the synchronization data structure and/or in the synchronization pair. Therefore, before synchronizing objects between two or more CDSs, the synchronization data structure MUST be created, if it does not exist, by describing the devices to be involved in the synchronization along with the policies to be applied and synchronization pair information for objects that are to be synchronized. **See 2.9.6 *AddSyncPair()* action for details on how to add synchronization pair information.**

Figure 7 and the texts below describe the sequence of steps during a synchronization operation:

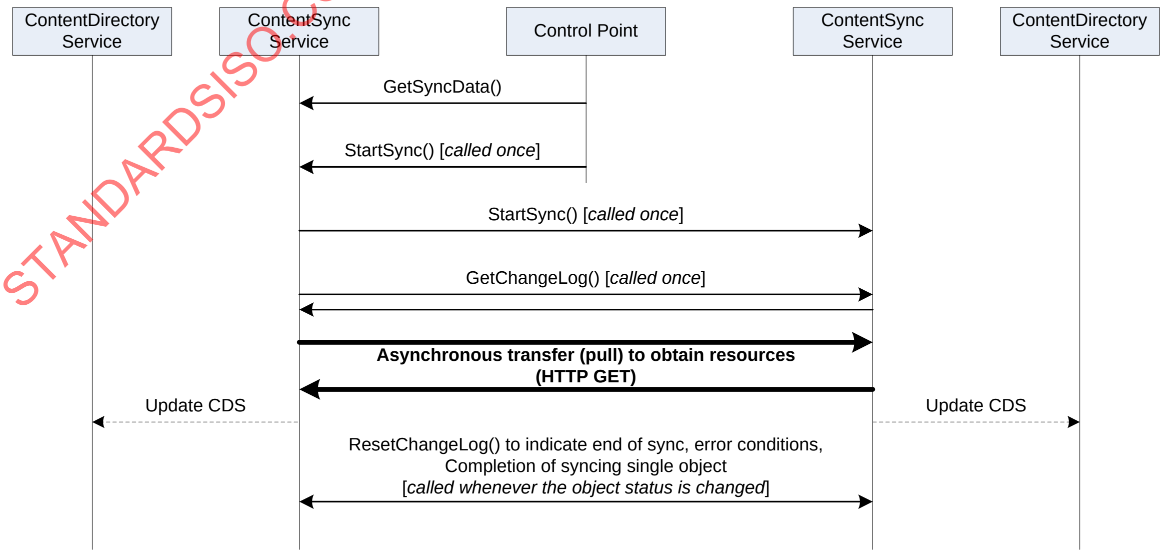
ContentDirectory Service
ContentSync Service
Control Point
ContentSync Service
ContentDirectory Service
GetSyncData()
StartSync() [called once]
StartSync() [called once]
GetChangeLog() [called once]
Asynchronous transfer (pull) to obtain resources (HTTP GET)
Update CDS
Update CDS
ResetChangeLog() to indicate end of sync, error conditions, Completion of syncing single object [called whenever the object status is changed]

Figure 7 — Interaction Diagram (Synchronization Operation)

a) A control point MAY invoke GetSyncData() action to retrieve existing synchronization data structure. The result of this action is a collection of synchronization relationships.

b) The control point triggers a synchronization operation by invoking the StartSync() action on one of the partner devices in the selected synchronization relationship. While invoking this action, the control point passes the target synchronization ID for identifying the synchronization relationship or partnership or pairGroup that is to be synchronized.

  1) The partner that receives the StartSync() action from a control point, MUST also trigger the synchronization operation on the other partner by invoking the StartSync() action on the partner.

c) Once the synchronization operation has been triggered successfully, the devices that are part of the synchronization operation perform the synchronization simultaneously. The subsequent process is as follows:

  1) Each device invokes GetChangeLog() action on the partner device to retrieve synchronization objects, which basically include the change log as objects are updated in the CDS.

  2) After receiving responses for the two actions above, each partner device parses and interprets the received DIDL-Lite XML document (Change log) to get synchronization object information. Since during the synchronization operation, some of the objects need to be created under a container object which itself needs to be created as well, the order how objects are to be synchronized should be handled very carefully. The synchronization operation MUST be done according to the following orders. **See Clause "2.10 Theory of Operation" for details on how individual object is synchronized.**

  Note: Each object in the change log MUST have one of the following XML elements;

    avcs:pair::remoteObjID
    or avcs:pair::remoteParentObjID
    or avcs:pair::virtualRemoteParentObjID

    - First, the device MUST synchronize objects (Scenario 1 pair) that have the avcs:pair::remoteObjID property.
    - Second, the device MUST synchronize objects (Scenario 2 pair) that have the avcs:pair::remoteParentObjID property.
      - The device creates new (local) object under the object that is identified by the value of the avcs:pair::remoteParentObjID property.
      - The partner device MUST replace the avcs:pair::remoteParentObjID property with the the avcs:pair::remoteObjID property and the value of which MUST be set to object@id of the newly created object. **(See Step 5 below how the partner device receives the information of the newly created object)**
    - Finally, the device MUST synchronize objects that have the avcs:pair::virtualRemoteParentObjID property. The device starts with the objects for which the value of the avcs:pair::virtualRemoteParentObjID property, is found in the avcs:pair::remoteObjID property of the objects. Once synchronized, the partner device MUST replace the avcs:pair::virtualRemoteParentObjID property with the avcs:pair::remoteObjID property and the value of which MUST be set to object@id of the newly created object. The device continues this process recursively until all objects are synchronized.

  3) Whenever a device obtains the DIDL-Lite XML fragment for each synchronization object, the device updates the local CDS in accordance with the synchronization policy as described in 2.2.3.

  4) Finally, the partner devices transfer resources using HTTP GET method as the transport protocol. Each partner device sends an event message whenever an object is processed. This event message includes the status of synchronization (i.e. SyncStatusUpdate state variable) which indicates whether the object is synchronized successfully or is failed to synchronize. When an object is synchronized successfully

the corresponding change log for that objects gets cleared and the CDS starts keeping log for new changes.

Note: Each CDS object MUST have at least one res property of which resource MUST be transferred using HTTP GET method if the object has a resource.

5) When a device receives the ResetChangeLog() action with objects that have synchronized, the device updates the avcs:pair property in the CDS object.

i) Each device MUST invoke the ResetChangeLog() action on the partner device to inform (acknowledge) the partner which remote objects have successfully synchronized with the local objects. A device can acknowledge multiple objects by a single action invocation. When invoking the ResetChangeLog() action, the device MUST provide the avcs:syncInfo@updateID for each object that is extracted from the result of the last GetChangeLog() action in order for the partner to decide what to reset in the change log. **See Clause 2.9.13 ResetChangeLog() how to reset the change log.**

d) After receiving an event message from the partner device that notifies the end of synchronization for all objects, the device releases all system resources that are involved in the synchronization operation.

2.5 Synchronization Operation (CDS to non CDS)

The synchronization between a CDS and a non-CDS (Control Point) is unidirectional and consists of the following steps:

a) A UPnP Control Point creates a new synchronization data structure containing a single synchronization relationship on the device with which the control point wants to synchronize. The process of creating such data structure is defined in 2.10.2.2. The following rules MUST be applied while creating a synchronization data structure for CDS-non CDS synchronization:

- Either <partner id="1"> or <partner id="2"> elements in the <partnership> element MUST be assigned to Non-CDS entity and its <deviceUDN> element and <serviceID> MUST have the empty string.
- If one of two partners is a non-CDS, adding another partnership is NOT allowed.
  - A partnership between CDS and Non-CDS can not be added into an existing synchronization relationship.

The following XML document shows an example format of the synchronization data structure for CDS and Non-CDS synchronization. (See Clause 2.7.4 **A_ARG_TYPE_SyncData** for details on a synchronization data structure)

```
<syncRelationship id="1cce93c2-6144-4093-9650-ae6c7ba28c91" active="1"
  xmlns="urn:schemas-upnp-org:cs"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:schemas-upnp-org:cs
   http://www.upnp.org/schemas/cs/cs-v1-20070XXXX.xsd">
 <title>ABC Electronic Program Guide</title>
 <partnership id="3fa8e9f8-ff21-47ee-90c8-7730793a613f" active="1">
  <partner id="1">
   <deviceUDN>e832a654-9c64-429b-9f34-8f55278f73a7</deviceUDN>
   <serviceID>AcmeContentSync-001</serviceID>
  </partner>
  <partner id="2">
   <deviceUDN></deviceUDN>
   <serviceID></serviceID>
  </partner>
  <policy>
   <syncType>tracking<syncType>
   <priorityPartnerID>1</priorityPartnerID>
   <autoObjAdd>1</autoObjAdd>
  </policy>
  <pairGroup id="bca02e62-e9d6-454c-b1b2-a52e199e02e7" active="1"/>
 </partnership>
</syncRelationship>
```

b) Once the synchronization data structure is created and pair information is added to the CDS objects, the CDS starts keeping track of changes to the objects.

c) A control point can retrieve the change log for all the objects that are part of the synchronization relationship by invoking the *GetChangeLog()* action defined in 2.9.11. After retrieving the change log, a control point can invoke the *ResetChangeLog()* action to instruct the CDS whether to continue accumulating the change log once it has been retrieved or starts keeping new log after flushing out the old logs.

d) Once the change log is retrieved, the control point compares the changes to objects with its internal database and updates its internal data base and hence essentially synchronizing with the CDS.

e) When a control point is no-longer interested in the change log for objects that are part of a synchronization relationship, the control point will delete the synchronization relationship by invoking the action *DeleteSyncData()* defined in 2.9.3.

f) Any resource transfer on this type of a synchronization relationship is out of scope of the specificatieon. Therefore, the *StartSync()* action invocation on the synchronization relationship containing a partnership between a CDS and a non-CDS MUST fail with an appropriate error code.

2.6 Garbage Collection

If a synchronization data structure is either inactive or has not been used for a long time for the purpose of synchronization, a CSS implementation can decide to remove that data structure and similarly the CDS implementation related to that CSS MUST remove all pair information associated with that data structure.

A synchronization pair that is not synchronized for a long time, an implementation can decide to remove that synchronization pair as well.

The future version of this specification will investigate to provide a standardized mechanism for garbage collection.

2.7 State Variables

Table 2-1 — State Variables

Variable Name	Req. or Opt. [a]	Data Type	Allowed Value [b]	Default Value	Eng. Units
SyncChange	*R*	**string**			
SyncStatusUpdate	*R*	**string**			
A_ARG_TYPE_ActionCaller	*R*	**string**			
A_ARG_TYPE_SyncData	*R*	**string**			
A_ARG_TYPE_SyncPair	*R*	**string**			
A_ARG_TYPE_SyncID	*R*	**string**			
A_ARG_TYPE_ObjectID	*R*	**string**			
A_ARG_TYPE_SyncStatus	*R*	**string**			
A_ARG_TYPE_ChangeLog	*R*	**string**			
A_ARG_TYPE_Index	*R*	**ui4**			
A_ARG_TYPE_Count	*R*	**ui4**			
A_ARG_TYPE_ResetObjectList	*R*	**string**			
Non-standard state variables implemented by an UPnP vendor go here.	*X*	*TBD*	*TBD*	*TBD*	*TBD*

Variable Name	Req. or Opt. [a]	Data Type	Allowed Value [b]	Default Value	Eng. Units

[a] R = Required, O = Optional, X = Non-standard

[b] Values listed in this column are required. To specify standard optional values or to delegate assignment of values to the vendor, you must reference a specific instance of an appropriate table below.

2.7.1 *SyncChange*

The *SyncChange* state variable contains an XML document identifying *all* changes that have occurred since the last time the *SyncChange* state variable was evented. Synchronization data structure change and synchronization object chage are evented in this version of the specification. See 2.8 for details. Individual events MUST be buffered and delivered in the order that they occurred with the most recent event corresponding to the last XML element within the *SyncChange XML Document* that is stored in the *SyncChange* state variable. Refer to 2.7.1.1, "*SyncChange* Data Format" and the "ContentSync service Event Schema" document for more details.

The *SyncChange* state variable is evented and moderated according to the GENA eventing mechanism as defined by the UPnP Device Architecture specification [DEVICE]. When multiple changes of object and synchronization data structure occur within the same moderation period (as determined by the implementation), each change MUST be accumulated in the *SyncChange* state variable and MUST be evented as a single event notification message when the current moderation period expires. After the event notification message has been sent to all subscribed control points, the value of the *SyncChange* state variable is reset when an update to the *SyncChange* state variable becomes necessary i.e. when the next event occurs. The resulting value is a fresh XML document that contains a single element that represents the update (i.e. it contains the first update event following the distribution of the previous event message to all subscribers). Subsequently, additional update elements are added to the XML document until the current moderation period ends and the current value of the *SyncChange* state variable (i.e. the current event message) is propagated to all event subscribers.

2.7.1.1 *SyncChange* Data Format

The optional XML header `<?xml version="1.0" ?>` is allowed. The (one and only) root element, <*SyncChange*>, MUST contain zero or more elements, each of which represents a change to a specific synchronization data structure.

The following example shows a generalized "template" for the format of the *SyncChange* state variable. Additional elements and/or attributes MAY be added to future versions of this specification. Furthermore, a 3rd-party vendor MAY add vendor-defined elements and/or attributes. However, by definition, this specification does not define the format and/or values for these 3rd-party elements. In order to eliminate element/attribute naming conflicts, the name of any vendor-defined element/attribute MUST follow the rules set forth in **Clause 1.3 "Vendor-defined Extensions"** All control points should gracefully ignore any element/attribute that it does not understand.

Note: The content of this state variable (i.e. the *SyncChange* XML document) MUST be properly escaped before it is sent to an event subscriber via GENA.

The following notation includes the *forum* character style to indicate names that are defined by the ContentSync Working Committee. Additionally, fields that need to be filled out by individual implementations are shown in the *vendor* character style.

```
<?xml version="1.0"?>
<SyncChange
  xmlns="urn:schemas-upnp-org:cs"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:schemas-upnp-org:cs
   http://www.upnp.org/schemas/cs/cs-v1-20070XXXX.xsd">
```

```
 <syncDataUpdate syncID="synchronization relationship or partnership or
   pairGroup ID of updated synchronization relationship"/>
 <syncObjUpdate objectID="object ID of updated synchronization object"/>
</SyncChange>
```

`xml`
OPTIONAL. Case sensitive.

`SyncChange`
REQUIRED. MUST have "urn:schemas-upnp-org:cs" (which is the UPnP ContentSync Working Committee Schema) as the value for the xmlns attribute that declares the default namespace; Contains all elements and attributes defined by the CDS Event schema as follows:

`syncDataUpdate`
OPTIONAL. Indicates that a synchronization relationship among the local device and other partner devices has been modified on line. If the device receives this state variable, the local device MUST browse updated relationship on the partner devices and update its local relationship information with the partner device's relationship information. This asynchronous update behavior only happens in case of deletion of the synchronization data structure. See 2.3.3 "Synchronization Data Structure Deletion" for details on the synchronization data structure update. The contents of this element MUST be the empty string. However, future versions of this specification may define specific values for this element. Consequently, control points must be prepared to gracefully ignore any element contents and/or element attributes that it does not understand. Contains all of the following attributes:

`syncID`
REQUIRED. xsd:string, Contains the *@id* attribute of the synchronization relationship or partnership or pairGroup that was added or modified.

`syncObjUpdate`
OPTIONAL. Indicates that a synchronization object has been modified since last synchronization operation. If the partner device receives this state variable, the local device can do synchronization operation immediately. The contents of this element MUST be the empty string. However, future versions of this specification may define specific values for this element. Consequently, control points must be prepared to gracefully ignore any element contents and/or element attributes that it does not understand. Contains all of the following attributes:

`objectID`
REQUIRED. xsd:string, Contains the *object@id* property of the synchronization object that was modified.

2.7.2 *SyncStatusUpdate*

This state variable is used for eventing purposes which allow a control point to receive meaningful event notifications whenever there is a update in synchronization operation involving a synchronization relationship. [CSS-XSD] defines the schema for the *SyncStatusUpdate XML Document* used in this state variable. The optional XML header =`<?xml version="1.0"?>` is allowed. One root element, `<SyncStatusUpdate>` has a list of one or more synchronization operation information structures representing currently ongoing synchronization operations. A synchronization operation information structure includes the status of the opreation. Other update elements MAY be added in the future CSS specifications as needed.

SyncStatusUpdate state variable is sent whenever a synchronization object is successfully synchronized or failed to synchronize during a synchronization operation. This state variable only contains new updates since the last time the state variable was evented. Once the update is sent, this update information is never sent again.

Example (before XML escaping):

```
<?xml version="1.0" encoding="utf-8">
<SyncStatusUpdate
  xmlns="urn:schemas-upnp-org:cs"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:schemas-upnp-org:cs
   http://www.upnp.org/schemas/cs/cs-v1-20070XXXX.xsd">
 <syncRelationship id="d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d">
  <status numberOfTotalObjects="50" numberOfCompletedObjects="47"
```

```
    numberOfFailedObjects="2">
   IN_PROGRESS_WITH_ERROR
  </status>
  <partnership id="a0e4d0a7-3378-4f17-8af2-3f7de3345dc6">
   <status numberOfTotalObjects="50" numberOfCompletedObjects="45"
     numberOfFailedObjects="2">
    IN_PROGRESS_WITH_ERROR
   </status>
   <pairGroup id="0ada9f4f-596f-4906-93d0-230f9df78a10">
    <status numberOfTotalObjects="25" numberOfCompletedObjects="23"
      numberOfFailedObjects="1">
     IN_PROGRESS_WITH_ERROR
    </status>
    <logEntry>
     <localObjectID>obj01</localObjectID>
     <remoteObjectID>robj07</remoteObjectID>
     <statusCode>001</statusCodes>
     <statusDescription>Succeeded completely</statusDescription>
    </logEntry>
   </pairGroup>
   <pairGroup id='70a74981-35f3-4262-84e8-ba0ec1794c0c'>
    <status numberOfTotalObjects="25" numberOfCompletedObjects="22"
      numberOfFailedObjects="1">
     IN_PROGRESS_WITH_ERROR
    </status>
    <logEntry>
     <localObjectID>obj03</localObjectID>
      <remoteObjectID>robj02</remoteObjectID>
      <statusCode>001</statusCodes>
      <statusDescription>Succeeded completely</statusDescription>
     </logEntry>
    </pairGroup>
  </partnership>
 </syncRelationship>
</SyncStatusUpdate>
```

The *SyncStatusUpdate* state variable MUST only be cleared just before adding the first update event that occurs after the last event message was sent.

A series of updates and the resulting eventing activity are illustrated in their temporal order in the example shown above.

2.7.2.1 *SyncStatusUpdate* Data Format

The optional XML header `<?xml version="1.0" ?>` is allowed. The (one and only) root element, <*SyncStatusUpdate*>, MUST contain zero or more elements, each of which represents a log of the synchronization object that is synchronized.

The following example shows a generalized "template" for the format of the *SyncStatusUpdate* state variable. Additional elements and/or attributes MAY be added to future versions of this specification. Furthermore, a 3rd-party vendor MAY add vendor-defined elements and/or attributes. However, by definition, this specification does not define the format and/or values for these 3rd-party elements. In order to eliminate element/attribute naming conflicts, the name of any vendor-defined element/attribute MUST follow the rules set forth in **Clause 1.3 "Vendor-defined Extensions"** All control points should gracefully ignore any element/attribute that it does not understand.

The following notation includes the *forum* character style to indicate names that are defined by the ContentSync Working Committee. Additionally, fields that need to be filled out by individual implementations are shown in the *vendor* character style.

```
<?xml version="1.0"?>
<SyncStatusUpdate
    xmlns="urn:schemas-upnp-org:cs"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:schemas-upnp-org:cs
    http://www.upnp.org/schemas/cs/cs-v1-20070XXXX.xsd">
```

```
  <syncRelationship id="synchronization relationship ID">
    <status numberOfTotalObjects="total number of synchronizing objects"
       numberOfCompletedObjects="number of synchronized objects"
       numberOfFailedObjects="number of synchronization-failed objects">
      synchronization status of this synchronization relationship
    </status>
    <partnership id="synchronization partnership ID"
      <status numberOfTotalObjects="total number of synchronizing objects"
         numberOfCompletedObjects="number of synchronized objects"
         numberOfFailedObjects="number of synchronization-failed objects">
        synchronization status of this synchronization partnership
      </status>
      <pairGroup id="synchronization pairGroup ID"
        <status numberOfTotalObjects="total number of synchronizing objects"
           numberOfCompletedObjects="number of synchronized objects
             within this pairGroup"
           numberOfFailedObjects="number of synchronization-failed objects">
          synchronization status of this synchronization pairGroup
        </status>
        <logEntry>
          <localObjID>local object ID</localObjID>
          <remoteObjID>remote object ID</remoteObjID>
          <statusCode>synchronization status codes</statusCode>
          <statusDesc>synchronization status description</statusDesc>
        </logEntry>
      </pairGroup>
    </partnership>
  </syncRelationship>
</SyncStatusUpdate>
```

`xml`

OPTIONAL. Case sensitive.

`SyncStatusUpdate`

REQUIRED. MUST have “urn:schemas-upnp-org:cs” (which is the UPnP ContentSync Working Committee Schema) as the value for the xmlns attribute that declares the default namespace; Contains all elements and attributes defined by the CSS schema as follows:

`syncRelationship`

OPTIONAL. a wrapper element that holds the synchronization operation information associated with a synchronization relationship. This element can appear multiple times to contain multiple synchronization relationships in the XML document.

`@id`

REQUIRED. xsd:string, contains an identifier to distinguish synchronization relationship from other synchronization relationships.

`status`

REQUIRED. xsd:string, indicates the status of a synchronization operation of the synchronization relationship identified by @id attribute above. This element MUST assume one of the following enumerated values:

IN_PROGRESS: The operation is in progress without any errors.

IN_PROGRESS_WITH_ERROR: The operation is in progress where some objects are failed to synchronization.

COMPLETED: The operation is completed.

COMPLETED_WITH_ERROR: The operation is finished, but some objects are failed to synchronization.

STOPPED: The operation is stopped by any reasons.

TEMPORARILY_STOPPED: The operation is temporarily stopped by any reason. This operation could be resumed at any time.

`@numberOfTotalObjects`

REQUIRED. xsd:unsignedInt, contains the total number of synchronization objects that are in the change log.

`@numberOfCompletedObjects`

REQUIRED. xsd:unsignedInt, contains the total number of objects that are successfully imported into the local CDS.

`@numberOfFailedObjects`
REQUIRED. xsd:unsignedInt, contains the number of objects that are failed to be imported into the local CDS.

`partnership`
REQUIRED. xsd:string, a wrapper element that holds the synchronization operation information associated with a synchronization partnership.

`@id`
REQUIRED. xsd:string, contains an identifier to distinguish a partnership from other partnerships in a synchronization relationship.

`status`
REQUIRED. xsd:string, indicates the status of a synchronization operation of the synchronization partnership identified by @id attribute above. This element MUST assume one of the following enumerated values:

IN_PROGRESS: The operation is in progress without any errors.

IN_PROGRESS_WITH_ERROR: The operation is in progress where some objects are failed to synchronization.

COMPLETED: The operation is completed.

COMPLETED_WITH_ERROR: The operation is finished, but some objects are failed to synchronization.

STOPPED: The operation is stopped by any reasons.

TEMPORARILY_STOPPED: The operation is temporarily stopped by any reason. This operation could be resumed at any time.

`@numberOfTotalObjects`
REQUIRED. xsd:unsignedInt, contains the total number of synchronization objects that are found in the change log from a partner device.

`@numberOfCompletedObjects`
REQUIRED. xsd:unsignedInt, contains the total number of objects that are successfully imported into the local CDS.

`@numberOfFailedObjects`
REQUIRED. xsd:unsignedInt, contains the number of objects that are failed to be imported into the local CDS.

`pairGroup`
REQUIRED. xsd:string, a wrapper element that holds the synchronization operation information associated with a synchronization pairGroup.

`@id`
REQUIRED. xsd:string, contains an identifier to distinguish a pairGroup from other pairGroups in a synchronization partnership.

`status`
REQUIRED. xsd:string, indicates the status of a synchronization operation of the synchronization pairGroup identified by @id attribute above. This element MUST assume one of the following enumerated values:

IN_PROGRESS: The operation is in progress without any errors.

IN_PROGRESS_WITH_ERROR: The operation is in progress where some objects are failed to synchronize.

COMPLETED: The operation is completed.

COMPLETED_WITH_ERROR: The operation is finished, but some objects are failed to synchronize.

STOPPED: The operation is stopped by any reasons.

TEMPORARILY_STOPPED: The operation is temporarily stopped for any reason. This operation could be resumed at any time.

`@numberOfTotalObjects`

REQUIRED. xsd:unsignedInt, contains the total number of synchronization objects that are found in the change log from a partner device.

`@numberOfCompletedObjects`

REQUIRED. xsd:unsignedInt, contains the total number of objects that are successfully imported into the local CDS.

`@numberOfFailedObjects`

REQUIRED. xsd:unsignedInt, contains the number of objects that are failed to be imported into the local CDS.

`logEntry`

REQUIRED. xsd:string, contains the result of synchronization operation for each synchronization object.

`localObjID`

REQUIRED. xsd:string, identifies a CDS object that resides on the local CDS of the device issuing the event. When the *SyncStatusUpdate XML document* is sent to the partner as an event message, this localObjID element is perceived as a remote object by the partner.

`remoteObjID`

REQUIRED. xsd:string, identifies a CDS object in the partner CDS that is paired with an object in the local CDS for synchronization. When the partner receives the *SyncStatusUpdate XML document* as an event message, this remoteObjID element is perceived as a local object at the partner device.

`statusCode`

REQUIRED. xsd:unsignedInt, indicates pre-defined status codes for a synchronization operation for the logEntry element. The table below defines the status codes to identify various synchronization conditions. This status list can be extended in the future by vendors. The status codes are grouped into separate categories and labeled as 1xx, 2xx, 3xx and 4xx, where each group represents the nature of status; such as: success status, general errors, media errors, system error and synchronization errors, respectively. The grouping of status codes allows a control point to be able to understand the nature of status when an unknown status code (that is: extended specification or vendor extended) is encountered. For example, for an unknown error labeled as 2xx, it can be interpreted by the control point as 200.

Table 2-2 — Status Codes of Synchronization Operation

Value	R/O	Description
Non-positive	N/A	These error codes are reserved for future use. Control points should gracefully ignore any non-positive error codes.
001-099	N/A	Non-Error Group
001	R	Success – Synchronization of an object is succeeded.
002	O	Partial Success – Synchronization of an object is succeeded, but some DIDL-Lite properties are missing due to device capability.
003	R	Not Accepted - the object is not accepted due to device capability
004-099	N/A	Reserved
100-199	N/A	General Error Code Group - arbitrary errors, which do not belong to other groups.
100	R	General Problem –a problem is confirmed, but no specific reason can be identified.
101	O	Disabled Sync Operation- the synchronization operation is disabled by the user.
102	O	The destination for the new object is not specified.
102-149	N/A	Reserved for future General Error Codes.
150-199	N/A	Reserved for vendor-defined General Error Codes.
200-299	N/A	Media Error Code Group - arbitrary media related errors.
200	O	General Media Problem–some trouble related to media is detected. Checking the media to resolve it.
201	O	Insufficient Disk Space–storage of the sync device (i.e., HDD or Flash Memory, etc…) does not have enough available space to complete the synchronization.
202	O	Storage Low Space - the storage of sync device has low available space and the synchronization process may fail. The criteria to determine “low space” is vendor dependent and may be independent from the size of the sync contents to synchronize.
203-249	N/A	Reserved for future Media Error Codes.
250-299	N/A	Reserved for vendor-defined Media Error Codes.
300-399	N/A	System Error Code Group - arbitrary system related error.
300	O	General System Problem –a problem related to the system is detected. It may affect all synchronization processes in the sync-enabled Content Directory service.
301	O	Insufficient Memory- the system does not have enough system memory to complete the synchronization processes.
302	O	Insufficient Processing - the system does not have enough CPU power to execute the designated synchronization processes.
303	O	Low Memory - the system has low available memory and the designated synchronization process may fail. The criteria to determine “low memory” is vendor dependent and may be independent from the size of the sync content to synchronize.
304	O	Low Processing - the system has low available CPU power and the designated synchronization process may fail. The criteria to determine “low processing” is vendor dependent and may be independent from the size of the sync content to synchronize.
305-349	N/A	Reserved for future System Error Codes.
350-399	N/A	Reserved for vendor-defined System Error Codes.
400-499	N/A	Content Error Code Group - arbitrary errors related to the content to be synchronized.
400	O	General Content Problem –a problem related to the content is detected. It may be associated with the content that is being synchronized.

Value	R/O	Description
401	O	No Sync Content–the necessary content is missing from the sync devices.
402	O	Content Write Protect - write access to the recording content is prohibited.
403	O	Synchronization Loser –there are other synchronizing process with the same contents(i.e., CDS objects) at the same period, and the current synchronization process is superseded by the conflicting synchronization process.
404	O	Content Locked- the originally sync content has been preempted by another synchronization process.
405	O	Invalid XML – xml document format for content metadata is not valid.
404-449	N/A	Reserved for future Content Error Codes.
450-499	N/A	Reserved for vendor-defined Content Error Codes.
500 and above	N/A	Reserved for future new category information extensions.

`statusDesc`
REQUIRED. xsd:string, expresses readable error status of the synchronization operation for this logEntry element.

2.7.3 A_ARG_TYPE_ActionCaller

This state variable is introduced to provide type information for the ActionCaller argument in various actions. The ActionCaller argument identifies the caller of an action. If the caller is a control point embedded in a UPnP device then the value MUST be the device's UDN. Otherwise the value MUST be set to the empty string indicating that the caller is a stand-alone control point.

2.7.4 A_ARG_TYPE_SyncData

This state variable is introduced to provide type information for various arguments that contain different parts of a synchronization data structure to be used in various actions. The A_ARG_TYPE_SyncData state variable MUST contain one of the following types of synchronization-related XML fragments:

- Synchronization relationship data: Represents synchronization relationship level information.
- Partnership data: Represents partnership level information for a given synchronization relationship. This fragment MUST NOT contain any synchronization or pairGroup level information.
- Pairgroup data: Represents pairGroup level information in a synchronization relationship for a given partnership. This fragment MUST NOT contain any synchronization or partnership level information.

All instances of this data type MUST comply with the [CSS-XSD] schema.

Note that since the ContentSync format of an argument of data type A_ARG_TYPE_SyncData is an XML document, it needs to be escaped (using the normal XML rules: [XML] Clause 2.4 Character Data and Markup) before embedding in a SOAP response message.

The example below shows synchronization data structure for synchronization between two CDSs:

Example:

```
<?xml version="1.0" encoding="utf-8">
<ContentSync xmlns="urn:schemas-upnp-org:cs"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:schemas-upnp-org:cs
   http://www.upnp.org/schemas/cs/cs-v1-20070XXXX.xsd">
 <syncRelationship id="d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d" active="1">
  <title>Sync between My iPod, My PMP and Home Media Server</title>
  <partnership id="a0e4d0a7-3378-4f17-8af2-3f7de3345dc6" active="1"
    updateID="0">
   <partner id="1">
    <deviceUDN>343bd2a2-189b-40c0-8eb5-ea91ea730402</deviceUDN>
    <serviceID>service_ID_A</serviceID>
   </partner>
   <partner id="2">
    <deviceUDN>05de2732-5df5-4c48-922b-12f73473f0e9</deviceUDN>
    <serviceID>service_ID_B</serviceID>
   </partner>
   <policy>
    <syncType>merge<syncType>
    <priorityPartnerID>1</priorityPartnerID>
   </policy>
   <pairGroup id="ba8e57de-7f66-4102-ae4b-31b96c86f173" active="1">
    <policy>
     <syncType>replace</syncType>
     <priorityPartnerID>1</priorityPartnerID>
    </policy>
   </pairGroup>
   <pairGroup id="0ada9f4f-596f-4906-93d0-230f9df78a10" active="1">
    <policy>
     <syncType>replace</syncType>
     <priorityPartnerID>2</priorityPartnerID>
    </policy>
   </pairGroup>
   <!-- More pairGroups can go here -->
  </partnership>
  <partnership id="864074ec-dad5-4d2c-b5c6-41e3e6f53b79" active="1"
    updateID="0">
   <partner id="1">
    <deviceUDN>343bd2a2-189b-40c0-8eb5-ea91ea730402</deviceUDN>
    <serviceID>service_ID_A</serviceID>
   </partner>
   <partner id="2">
    <deviceUDN>e832a654-9c64-429b-9f34-8f55278f73a7</deviceUDN>
    <serviceID>service_ID_C</serviceID>
   </partner>
   <policy>
    <syncType>merge</syncType>
    <priorityPartnerID>2</priorityPartnerID>
   </policy>
   <pairGroup id="265193c0-0b07-4f33-979c-f4701a98a1d9" active="1"/>
  </partnership>
  <!-- More partnerships can go here -->
 </syncRelationship>
 <syncRelationship id="e884c276-c489-44f0-bcec-332450dab074" active="1">
  <title>Sync between My PMP and Home Media Server</title>
  <partnership id="1ab3fef4-777e-496a-82ed-d2580cdafa75" active="1"
    updateID="0">
   <partner id="1">
    <deviceUDN>343bd2a2-189b-40c0-8eb5-ea91ea730402</deviceUDN>
    <serviceID>service_ID_A</serviceID>
   </partner>
   <partner id="2">
    <deviceUDN>ef7c6650-5748-4cc7-9cde-6a5b8719615f</deviceUDN>
    <serviceID>service_ID_D</serviceID>
   </partner>
   <policy>
    <syncType>replace</syncType>
    <priorityPartnerID>2</priorityPartnerID>
   </policy>
   <pairGroup id="c1bc5bd7-0207-4226-beee-b528fe63a919" active="1"/>
  </partnership>
 </syncRelationship>
</ContentSync>
```

The XML document example above contains multiple synchronization relationships and multiple partnerships in the first synchronization relationship. However, only one partnership is allowed in a synchronization relationship in this version of the specification.

A synchronization relationship (and its *syncRelationship* data structure) is identified by a globally unique *syncRelationship@id* element. A synchronization relationship is composed of one or more partnerships (see below). A synchronization relationship can be in an active or inactive state. An active state means that the synchronization relationship participates in a synchronization operation whereas an inactive synchronization relationship does not participate in a synchronization operation. The active state of a synchronization relationship is expressed by the *syncRelationship@active* element.

A partnership identifies two specific partner devices containing content that is synchronized during a synchronization operation. A partnership exists only between two sync partner devices. The partner devices are identified by their respective UDN values. Each partnership is identified by a globally unique *partnership@id* element.

Each partnership consists of one or more synchronization PairGroups. A PairGroup identifies a set of synchronization pairs where identical synchronization policies are applied. Each object that belongs to a pair and is associated with a pairGroup includes a *avcs:syncInfo::pair* property that contains a reference to that PairGroup via the PairGroup's id. .

Within every synchronization data structure (relationship, partnership, and PairGroup) a default policy is defined such that all dependent structures inherit that policy unless the dependent structure specifies a policy on its own.For example if policies are defined for a PairGroup and for a pair under that pairGroup, the pair policy will override the pairGroup policy. Similarly, if policies are defined for a partnership and a pairGroup under that partnership, then pairGroup policy will override partnership policy.

The *partnership@updateID* element can be used to determine whether locally cached partnership information has become stale. The *partnership@updateID* element value is increased by one whenever the partnership information is modified. See the action *ModifySyncData()* for more details.

The synchronization data structure for a given synchronization relationship MUST be identical in all devices that are referenced within that synchronization relationship before performing any synchronization operation.

2.7.4.1 A_ARG_TYPE_SyncData **Data Format**

The optional XML header `<?xml version="1.0" ?>` is allowed. The (one and only) root element, *<ContentSync>*, MUST contain zero or more elements, each of which represents a synchronization data structure.

The following example shows a generalized "template" for the format of the *A_ARG_TYPE_SyncData* state variable. Additional elements and/or attributes MAY be added to future versions of this specification. Furthermore, a 3rd-party vendor MAY add vendor-defined elements and/or attributes. However, by definition, this specification does not define the format and/or values for these 3rd-party elements. In order to eliminate element/attribute naming conflicts, the name of any vendor-defined element/attribute MUST follow the rules set forth in **Clause 1.3 "Vendor-defined Extensions".** All control points should gracefully ignore any element/attribute that it does not understand.

The following notation includes the *forum* character style to indicate names that are defined by the ContentSync Working Committee. Additionally, fields that need to be filled out by individual implementations are shown in the *vendor* character style.

```
<?xml version="1.0"?>
<ContentSync
```

```
  xmlns="urn:schemas-upnp-org:cs"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:schemas-upnp-org:cs
   http://www.upnp.org/schemas/cs/cs-v1-20070XXXX.xsd">
 <syncRelationship id="synchronization relationship ID"
   active="flag indicates whether a relationship is enabled or disabled"
   systemUpdateID="system update ID of the CDS at the time of change">
  <title>title of this synchronization relationship</title>
  <partnership id="synchronization partnership ID"
    active="flag indicates whether a partnership is enabled or disabled"
    updateID="uniquely assigned ID when a partnership is updated">
   <partner id="1">
    <deviceUDN>device UDN of the first partner</deviceUDN>
    <serviceID>ID of a service of the first partner</serviceID>
   </partner>
   <partner id="2">
    <deviceUDN>device UDN of the second partner</deviceUDN>
    <serviceID>ID of a service of the second partner</serviceID>
   </partner>
   <policy>synchronization policy in a partnership level</policy>
   <pairGroup id="synchronization pairGroup ID"
     active="flag indicates whether a pairGroup is enabled or disabled">
    <policy>synchronization policy in a pairGroup level</policy>
   </pairGroup>
  </partnership>
 </syncRelationship>
</ContentSync>
```

`xml`

OPTIONAL. Case sensitive.

`ContentSync`

REQUIRED. MUST have “urn:schemas-upnp-org:cs” (which is the UPnP ContentSync Working Committee Schema) as the value for the xmlns attribute that declares the default namespace; Contains all elements and attributes defined by the CSS schema as follows:

`syncRelationship`

OPTIONAL. a wrapper element that holds the information associated with a synchronization relationship. This element can appear multiple times to contain multiple synchronization relationship in the XML document.

`@id`

REQUIRED. xsd:string, contains an identifier to distinguish a synchronization relationship from other synchronization relationships. The value of this attribute MUST be generated using GUID as defined in RFC 4122. A GUID is 128 bits long and can guarantee uniqueness across space and time.

`@active`

REQUIRED. xsd:boolean, Indicates whether a synchronization relationship is enabled. To indicate a synchronization relationship is currently disabled, the syncRelationship@active attribute MUST be set to false (“0”). Attempting to synchronize a disabled synchronization relationship MUST result in an error. Each synchronization partner MUST keep its local Change Log even though the synchronization relationship is disabled. Enabling is accomplished by setting the active attribute to true (“1”). If a relationship is disabled then all partnerships under this relationship will be treated as disabled regardless of the setting of the active flags of those partnerships.

`@systemUpdateID`

REQUIRED. xsd:string, Indicates *systemUpdateID* property of the CDS at the time of change

`title`

REQUIRED. xsd:string, contains a user-friendly name for the synchronization relationship.

`partnership`

REQUIRED. xsd:string, indicates which two devices in a synchronization relationship are partnered together. Sub-properties of the partnership element describe detailed information of the partnership. This element MUST appear under the <syncRelationship> element.

`@id`

REQUIRED. xsd:string, contains an identifier to distinguish a partnership from other partnerships in a synchronization relationship. The value of the partnership@id attribute

MUST be generated using GUID as defined in RFC 4122. A GUID is 128 bits long and can guarantee uniqueness across space and time.

`@active`

REQUIRED. xsd:boolean, indicates whether a partsnerhip is enabled. To indicate that a partnership is currently disabled, the partnership@active MUST be set to false (“0”). Attempting to synchronize a disabled partnership MUST result in an error. Each synchronization partner MUST keep its local Change Log even though the partnership is disabled. Enabling is accomplished by setting the active attribute to true (“1”). If a partnership is disabled then all pairGroups under this partnership will be treated as disabled regardsless of the setting of the active flags of those pairGroups.

`@updateID`

REQUIRED. xsd:string, a counter that increases its value whenever there is a change in this partnership. The value of the change must be increased by 1. This attribute is used to prevent updates with stale data.

`partner`

REQUIRED. xsd:string, indicates individual device information that is involved in a partnership.

`@id`

REQUIRED. xsd:string, contains the unique ID to identify a partner in a partnership. The value of partner@id attribute is static, “1” or “2”. “1” represents the first partner and “2” represents the second partner.

`deviceUDN`

REQUIRED. xsd:string, contains the UDN of the device that provides ContentSync service.

`serviceID`

REQUIRED. xsd:string, contains the service ID of the CSS of the partner device.

`policy`

REQUIRED. xsd:string, indicates how to synchronize objects that are involved in a synchronization partnership. **See 2.2.3 “Synchronization Policy and Behavior” for policy definition and format.**

`pairGroup`

REQUIRED. xsd:string, indicates pairGroup information within a partnership. This element can appears multiple times under the <partnership> element.

`@id`

REQUIRED. xsd:string, uniquely identifies a pairGroup within a partnership to distinguish it from other pairGroups in a synchronization partnership. The value of this attribute MUST be generated using GUID as defined in RFC 4122. A GUID is 128 bits long and can guarantee uniqueness across space and time.

`@active`

REQUIRED. xsd:string, indicates whether a pairGroup is enabled. To indicate that a pairGroup is currently disabled, the *pairGroup@active* MUST be set to false (“0”). Attempting to synchronize a disabled pairGroup MUST result in an error. Each synchronization partner MUST keep its local Change Log even though the pairGroup is disabled. Enabling is accomplished by setting the active attribute to true (“1”).

`@updateID`

REQUIRED. xsd:string, a counter that increases its value whenever there is a change in this pairGroup.

`policy`

REQUIRED. xsd:string, indicates how to synchronize objects that are involved in a synchronization pairGroup. **See 2.2.3 “Synchronization Policy and Behavior” for policy definition and format.**

2.7.5 A_ARG_TYPE_SyncPair

This state variable is introduced to provide type information for various arguments that contain a synchronization pair for a CDS object to be used in various actions.

The following illustrates a typical example of the A_ARG_TYPE_SyncPair state variable

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<syncInfo updateID="3" xmlns="urn:schemas-upnp-org:cs:avcs"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:schemas-upnp-org:cs:avcs
   http://www.upnp.org/schemas/cs/avcs-v1-20070XXXX.xsd">
 <pair
   syncRelationshipID="d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d"
   partnershipID="a0e4d0a7-3378-4f17-8af2-3f7de3345dc6"
   pairGroupID="ba8e57de-7f66-4102-ae4b-31b96c86f173">
  <remoteObjID>B1</remoteObjID>
<policy>
 <syncType>replace</syncType>
 <priorityPartnerID>1</priorityPartnerID>
</policy>
  <status>MODIFIED</status>
 </pair>
 <pair
   syncRelationshipID="e884c276-c489-44f0-bcec-332450dab074"
   partnershipID="1ab3fef4-777e-496a-82ed-d2580cdafa75"
   pairGroupID="c1bc5bd7-0207-4226-beee-b528fe63a919">
<remoteParentObjID>B2</remoteParentObjID>
<status>NEW</status>
 </pair>
</syncInfo>
```

Since XML elements in this state variable are CDS object property, see Annex A for details of pair information. Also, see [CSS-XSD] for a schema of the ***A_ARG_TYPE_SyncPair*** state variable.

2.7.6 A_ARG_TYPE_SyncID

This state variable is introduced to provide type information for various action arguments that uniquely identify a synchronization relationship, or a partnership or a pairGroup. The value of this variable MUST be generated using GUID as defined in RFC 4122. A GUID is 128 bits long and can guarantee uniqueness across space and time.

2.7.7 A_ARG_TYPE_ObjectID

This state variable is introduced to provide type information for various action arguments that uniquely identify a CDS object. The format of the *A_ARG_TYPE_ObjectID* state variable MUST follow the definition of the ***A_ARG_TYPE_ObjectID*** in the ContentDirectory:2 Service specification and the definition of the *@id* property in Annex A.

2.7.8 A_ARG_TYPE_SyncStatus

This state variable is introduced to provide type information for the *SyncStatus* argument in the *GetSyncStatus()* action which contains a list of zero or more synchronization operation information structures representing both currently ongoing and the previous synchronization operation. A synchronization operation information structure includes the status of the opreation. All instances of this data type MUST comply with the *SyncStatus XML document* schema. **See 2.7.2 *SyncStatusUpdate* state variable for details.**

Note that since the *SyncStatus* format of an argument of data type *A_ARG_TYPE_SyncStatus* is an XML document, it needs to be escaped (using the normal

XML rules: [XML] Clause 2.4 Character Data and Markup) before embedding in a SOAP response message.

This value of this state variable is identical to the value of the *SyncStatusUpdate* state variable except that the *SyncStatusUpdate* only contains the status of currently ongoing synchronization operations. The *A_ARG_TYPE_SyncStatus* state variable contains the status of both currently ongoing synchronization operations and the status of the last synchronization operation.

When a synchronization operation is invoked on a relationship level, the status information of the last synchronization operation of that specific synchronization relationship including the status of the last synchronization operations of all partnerships within that relationship and the status of the last synchronization operations of all pairGroups within each partnership MUST be cleared. When a synchronization operation is invoked on a partnership level then the status information of the last synchronization operation for that specific partnership including the status of last synchronization operations of all pairGroups within that partnership MUST be cleared. When a synchronization operation is invoked on a pairGroup level then the status information of the last synchronization operation for that specific pairGroup MUST be cleared.

2.7.9 *A_ARG_TYPE_ChangeLog*

This state variable is introduced to provide type information for the *ChangeLog* argument in the *GetChangeLog()* action. The structure of the *ChangeLog* argument is a DIDL-Lite XML Document.

A change log is a list of CDS objects represented by DIDL-Lite XML document with extension in this specification. The change log contains the CDS objects which have changed since the last synchronization operation. When the change log is returned as a reponse of the *GetChangeLog()* action, it contains only the changed CDS objects which are bound to a specific synchronization relationship or partnership or pairGroup.

- Optional XML declaration `<?xml version="1.0" ?>`
- `<DIDL-Lite>` is the root element.
- `<container>` is the element representing objects of class *container* and all its derived classes, which has been changed since the last synchronization operation.
- `<item>` is the element representing objects of class *item* and all its derived classes, which has been changed since the last synchronization operation.
- Elements in the Dublin Core (dc) and UPnP (upnp) namespaces represent object metadata.
- See the DIDL-Lite schema [DIDL-LITE-XSD] for more details on the structure. The available properties and their names are described in Annex B, "AV Working Committee Extended Properties" in the ContentDirectory:2 service [CDS].

Note that since the value of *ChangeLog* is XML, it needs to be escaped (using the normal XML rules: [XML] Clause 2.4 Character Data and Markup) before embedding in a SOAP response message.

For objects that are deleted outside of synchronization operation, the change log SHOULD provide deleted objects information with only *item* or *container*, and its *avcs:syncInfo* properties if and only if a synchronization policy is *replace*. If an object which is deleted is part of multiple synchronization pairs, then deleted information MUST be kept until all synchronization pairs are synchronized. The following gives an example of deleted objects information in a change log.

Example: (The namesapce declaration is omitted)

```
<item id="A3" parentID="A2">
```

```
 <avcs:syncInfo updateID="1">
  <avcs:pair
    syncRelationshipID="d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d"
    partnershipID="a0e4d0a7-3378-4f17-8af2-3f7de3345dc6"
    pairGroupID="a8e57de-7f66-4102-ae4b-31b96c86f173">
    <avcs:remoteObjID>B3</avcs:remoteObjID>
    <avcs:status>DELETED</avcs:status>
  </avcs:pair>
 </avcs:syncInfo>
</item>
```

2.7.10 *A_ARG_TYPE_Index*

This state variable is introduced to provide type information for an argument in various actions. Arguments specify an offset into an arbitrary list of objects (change log). A value of 0 represents the first CDS object in the change log.

2.7.11 *A_ARG_TYPE_Count*

This state variable is introduced to provide type information for an argument in various actions. Arguments specify an ordinal number of arbitrary objects.

2.7.12 *A_ARG_TYPE_ResetObjectList*

This state variable is introduced to provide type information for an argument that contains a list of synchronization objects of which the change log will be cleaned.

The structure of the argument of data type *A_ARG_TYPE_ResetObjectList* is an XML document (See [CSS-XSD]):

Note that since *A_ARG_TYPE_ResetObjectList* is an XML document, it needs to be escaped (using the normal XML rules: [XML] Clause 2.4 Character Data and Markup) before embedding in a SOAP response message.

Example:

```
<ResetObjectList xmlns="urn:schemas-upnp-org:cs
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:schemas-upnp-org:cs
   http://www.upnp.org/schemas/cs/cs-v1-20070XXXX.xsd">
 <object id="A1" remoteObjID="32" updateID="2"/>
 <object id="A72" remoteObjID="9547" updateID="4"/>
</ResetObjectList>
```

2.7.12.1 *A_ARG_TYPE_ResetObjectList* Data Format

The optional XML header `<?xml version="1.0" ?>` is allowed. The (one and only) root element, *<ResetObjectList>*, MUST contain zero or more elements, each of which identifies a synchronization object.

The following example shows a generalized "template" for the format of the *A_ARG_TYPE_ResetObjectList* state variable. Additional elements and/or attributes MAY be added to future versions of this specification. Furthermore, a 3rd-party vendor MAY add vendor-defined elements and/or attributes. However, by definition, this specification does not define the format and/or values for these 3rd-party elements. In order to eliminate element/attribute naming conflicts, the name of any vendor-defined element/attribute MUST follow the rules set forth in **Clause 1.3 "Vendor-defined Extensions".** All control points should gracefully ignore any element/attribute that it does not understand.

The following notation includes the *forum* character style to indicate names that are defined by the ContentSync Working Committee. Additionally, fields that need to be filled out by individual implementations are shown in the *vendor* character style.

```
<?xml version="1.0"?>
<ResetObjectList
  xmlns="urn:schemas-upnp-org:cs"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="rn:schemas-upnp-org:cs
   http://www.upnp.org/schemas/cs/cs-v1-20070XXXX.xsd">
 <objectID id="object ID"
   remoteObjID="object ID of a partner paired with this object"
   updateID="uniquely assigned ID when the object is changed">
 </objectID>
</ResetObjectList>
```

`xml`
OPTIONAL. Case sensitive.

`ResetObjectList`
REQUIRED. MUST have "urn:schemas-upnp-org:cs" (which is the UPnP ContentSync WC Schema) as the value for the xmlns attribute that declares the default namespace; Contains all elements and attributes defined by the CSS schema as follows:

`objectID`
OPTIONAL. xsd:string, contains object@id property of the CDS which identifies the object of which changed log will be cleaned.

`@id`
REQUIRED. xsd:string, contains the ID of the object of which change log to be reset.

`@remoteObjID`
REQUIRED. xsd:string, contains the object ID in a partner, which is paired with the local object.

`@updateID`
REQUIRED. xsd:unsignedInt, contains the value of avcs:syncInfo@updatedID which was retrieved by the GetChangelog() action before.

2.8 Eventing and Moderation

Table 2-3 — Event Moderation

Variable Name	Evented	Moderated Event	Max Event Rate	Logical Combination	Min Delta per Event
SyncChange	*YES*	*YES*	0.2 sec		
SyncStatusUpdate	*YES*	*YES*	0.2 sec		

The *SyncStatusUpdate* state variable is evented and moderated. When multiple updates occur between moderation periods, the *SyncStatusUpdate* state variable accumulates all updates within that period and sends an event message at the end of the moderation period that contains all of the accumulated events. The *SyncStatusUpdate* state variable MUST only be cleared just before adding the first update event that occurs after the last event message was sent.

2.9 Actions

Immediately following this table is detailed information about these actions, including short descriptions of the actions, the effects of the actions on state variables, and error codes defined by the actions.

Table 2-4 — Actions

Name	Req. or Opt. [a]
AddSyncData()	R
ModifySyncData()	R
DeleteSyncData()	R
GetSyncData()	R
ExchangeSyncData()	R
AddSyncPair()	R
ModifySyncPair()	R
DeleteSyncPair()	R
StartSync()	O
AbortSync()	O
GetChangeLog()	R
ResetChangeLog()	R
ResetStatus()	R
GetSyncStatus()	O
Non-standard actions implemented by an UPnP vendor go here.	X

[a] R = Required, O = Optional, X = Non-standard

2.9.1 *AddSyncData()*

This action creates either a new synchronization relationship template, or a new partnership template within an existing relationship or a new pairGroup template within an existing partnership.

When creating a new synchronization relationship by invoking the *AddSyncData()* action, the *SyncData* input argument MUST contain a minimally complete synchronization relationship data structure. The control point can add additional pairGroups for the partnership by invoking the *AddSyncData()* action where the *SyncData* input argument contain the pairGroup data structure. **See Clause "2.7.4 *A_ARG_TYPE_SyncData* state variable" for details.**

The *ActionCaller* argument identifies the *deviceUDN* of the caller. **See Clause "2.7.3 *A_ARG_TYPE_ActionCaller* state variable" for details.**

If the *ActionCaller* argument specifies a *deviceUDN* then this action is invoked by a partner device and the caller does not need to disseminate synchronization data structure (*SyncData*) to the partner device. However, if the *ActionCaller* argument is *null*, the device MUST disseminate the synchronization data structure (*SyncData*) to the partner device, by invoking the *AddSyncData()* action on the partner .

When creating a new synchronization relationship, the partner device involved in the relationship MUST be in the network. See "Clause 2.3.1 Synchronization Data Structure Addition" for detailed rules.

When a device receives the *AddSyncData()* action from a stand-alone control point to create a new synchronization relationship, the device generates three IDs to identify the synchronization relationship, the partnership and the pairGroup for the minimally complete synchronization data structure. The generated IDs conform to the requirement of *A_ARG_TYPE_SyncID* state variable.

When a stand-alone control point is adding a synchronization relationship, the value of the *SyncID* input argument MUST be set to the empty string. While adding a partnership, the *SyncID* argument will contain the *SyncID* of an existing synchronization relationship where the partnership information will be added. Likewise, while adding a pairGroup, the SyncID argument will contain the SyncID of an existing partnership where the pairGroup information will be added.

The SyncDataResult output argument returns the synchronization relationship data structure containing the newly added data specified by the SyncData argument. In the case of adding a pairGroup to an existing synchronization relationship, the SyncDataResult argument will return the whole synchronization relationship data structure which contains that pairGroup.

2.9.1.1 Arguments

Table 2-5 — Arguments for AddSyncData()

Argument	Direction	Related State Variable
ActionCaller	IN	A_ARG_TYPE_ActionCaller
SyncID	IN	A_ARG_TYPE_SyncID
SyncData	IN	A_ARG_TYPE_SyncData
SyncDataResult	OUT	A_ARG_TYPE_SyncData

2.9.1.2 Dependency on State

None.

2.9.1.3 Effect on State

None.

2.9.1.4 Errors

Table 2-6 — Error Codes for AddSyncData()

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture clause on Control.
500-599	TBD	See UPnP Device Architecture clause on Control.
600-699	TBD	See UPnP Device Architecture clause on Control.
701	No such sync data	The AddSyncData() request failed because the specified SyncID argument is invalid.
702	Invalid XML	The AddSyncData() request failed because the specified SyncData argument
703	Invalid action caller	The AddSyncData() request failed because the action caller is a part of the sync data.
704	Partner Timeout	The AddSyncData() request failed because the sync data structure could not be exchanged due to time out of the partner device.
705	Partner not online	The AddSyncData() request failed because partner device is not in the network.

2.9.2 ModifySyncData()

This action modifies either a synchronization relationship, or a partnership within an existing relationship or a pairGroup within an existing partnership.

To modify synchronization relationship level information, all partner devices involved in the relationship MUST be in the network. **See "Clause 2.3.2 Synchronization Data Structure Modification" for detailed rules.**

To modify a synchronization data structure, SyncID argument that identifies which synchronization data structure is being modified MUST be specified.

To maintain identical synchronization relationship, partnership and pairGroup information on all partner devices, the device that receives this action MUST invoke the ModifySyncData() action on the partner device by including identical synchronization relationship, partnership or pairGroup data structure in the SyncData action argument.

The ActionCaller argument identifies the deviceUDN of the caller. **See Clause "2.7.3 A_ARG_TYPE_ActionCaller state variable" for details**.

If the ActionCaller argument specifies a deviceUDN then this action is invoked by a partner device and the caller does not need to disseminate synchronization data structure (SyncData) to the partner device. However, if the ActionCaller argument is empty string, the device MUST disseminate the synchronization data structure (SyncData) to the partner device, by invoking the ModifySyncData() action on the partner .

To prevent updating synchronization data structure by stale data, the SyncData input argument MUST contain the @updateID attribute of a partnership or pairGroup when the partnership or pairGroup level is modified.

If the modification would result in a synchronization relationship that is no longer valid, the ModifySyncData() action MUST fail without any change and return an appropriate error code.

2.9.2.1 Arguments

Table 2-7 — Arguments for ModifySyncData()

Argument	Direction	Related State Variable
ActionCaller	IN	A_ARG_TYPE_ActionCaller
SyncID	IN	A_ARG_TYPE_SyncID
SyncData	IN	A_ARG_TYPE_SyncData

2.9.2.2 Dependency on State

None.

2.9.2.3 Effect on State

None.

2.9.2.4 Errors

Table 2-8 — Error Codes for *ModifySyncData()*

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture clause on Control.
500-599	TBD	See UPnP Device Architecture clause on Control.
600-699	TBD	See UPnP Device Architecture clause on Control.
701	No such sync data	The *ModifySyncData()* request failed because the specified *SyncID* argument is invalid.
702	Invalid XML	The *ModifySyncData()* request failed because the specified *SyncData* argument
703	Invalid action caller	The *ModifySyncData()* request failed because the action caller is a part of the sync data.
704	Partner Timeout	The *ModifySyncData()* request failed because the sync data structure could not be exchanged due to time out of the partner device.
705	Partner not online	The *ModifySyncData()* request failed because partner device is not in the network.
706	Update in-progress	The *ModifySyncData()* request failed because another action request is still being processed.
707	Stale data	The *ModifySyncData()* request failed because the sync data is stale.

2.9.3 *DeleteSyncData()*

This action deletes either a synchronization relationship, or a partnership within an existing synchronization relationship or a pairGroup within an existing partnership. The *SyncID* argument of the action *DeleteSyncData()* identifies the synchronization relationship or the partnership or the pairGroup to be deleted.

The *ActionCaller* argument identifies the *deviceUDN* of the caller. **See Clause "2.7.3 *A_ARG_TYPE_ActionCaller* state variable" for details**.

If the *ActionCaller* argument specifies a *deviceUDN* then this action is invoked by a partner device and the caller does not need to inform the partner device of the deletion. However, if the *ActionCaller* argument is *null*, the device MUST inform the partner device of the deletion, by invoking the *DeleteSyncData()* action on the partner.

A deletion of a partnership or a pairGroup is allowed even when one of the partner devices is not in the network. In this case, the other partner device gets updated synchronization data structure by invoking the *ExchangeSyncData()* action before performing any synchronization operation when the device rejoins the network. **See "Clause 2.3.3 Synchronization Data Structure Deletion" for detailed rules.**

When the last pairGroup within an existing partnership is deleted, the partnership MUST be deleted as well because the synchronization data structure does not allow a partnership without at least one pairGroup.

Likewise, when the last partnership within an existing relationship is deleted, the relationship MUST be deleted, as well.

2.9.3.1 Arguments

Table 2-9 — Arguments for *DeleteSyncData()*

Argument	Direction	Related State Variable
ActionCaller	*IN*	*A_ARG_TYPE_ActionCaller*
SyncID	*IN*	*A_ARG_TYPE_SyncID*

2.9.3.2 Dependency on State

None.

2.9.3.3 Effect on State

None.

2.9.3.4 Errors

Table 2-10 — Error Codes for *DeleteSyncData()*

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture clause on Control.
500-599	TBD	See UPnP Device Architecture clause on Control.
600-699	TBD	See UPnP Device Architecture clause on Control.
701	No such sync data	The *DeleteSyncData()* request failed because the specified *SyncID* argument is invalid.
703	Invalid action caller	The *DeleteSyncData()* request failed because the action caller is a part of the sync data.

2.9.4 *GetSyncData()*

This action returns the synchronization data structure identified by the *SyncID* input argument. If the value of the action argument *SyncID* identifies a synchronization relationship then the *SyncData* output argument contains the entire synchronization data structure for that synchronization relationship including all partnerships within that relationship and all pairGroups for each partnership contained within that relationship. If the value of the action argument *SyncID* identifies a partnership then the *SyncData* output argument contains the synchronization data structure for that partnership including all pairGroups contained within that partnership. If the value of the action argument *SyncID* identifies a pairGroup then the *SyncData* output argument contains the synchronization data structure for the identified pairGroup. If the value of the action argument *SyncID* is the empty string then the *SyncData* output argument contains the synchronization data structure for all synchronization relationships.

2.9.4.1 Arguments

Table 2-11 — Arguments for *GetSyncData()*

Argument	Direction	Related State Variable
SyncID	*IN*	*A_ARG_TYPE_SyncID*
SyncData	*OUT*	*A_ARG_TYPE_SyncData*

2.9.4.2 Dependency on State

None.

2.9.4.3 Effect on State

None.

2.9.4.4 Errors

Table 2-12 — Error Codes for *GetSyncData()*

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture clause on Control.
500-599	TBD	See UPnP Device Architecture clause on Control.
600-699	TBD	See UPnP Device Architecture clause on Control.
701	No such sync data	The *GetSyncData()* request failed because the specified *SyncID* argument is invalid.

2.9.5 *ExchangeSyncData()*

This action exchanges a synchronization data structure between two partner devices. When the partner device joins the network, the device MUST evaluate whether or not the synchronization data structure is stale by exchanging its own synchronization data structure with other partner devices that are also in the network.

The *LocalSyncData* input argument contains the synchronization data structure for the local device. The partner device's synchronization data structure is returned in the *RemoteSyncData* output argument as response to the *ExchangeSyncData()* action.

The *RemoteSyncData* output argument MUST contain the synchronization data structure that is updated with the *LocalSyncData* input argument. It means that the partner device MUST do the update operation before responding to the the *ExchangeSyncData()* action.

2.9.5.1 Arguments

Table 2-13 — Arguments for *ExchangeSyncData()*

Argument	Direction	Related State Variable
LocalSyncData	*IN*	*A_ARG_TYPE_SyncData*
RemoteSyncData	*OUT*	*A_ARG_TYPE_SyncData*

2.9.5.2 Dependency on State

None.

2.9.5.3 Effect on State

None.

2.9.5.4 Errors

Table 2-14 — Error Codes for *ExchangeSyncData()*

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture clause on Control.
500-599	TBD	See UPnP Device Architecture clause on Control.
600-699	TBD	See UPnP Device Architecture clause on Control.

2.9.6 *AddSyncPair()*

This action adds synchronization pair information into a CDS object. See Clause "2.2.1 Synchronization Object and Pair" definition for details.

The *ActionCaller* input argument identifies the *deviceUDN* of the caller. **See Clause "2.7.3 *A_ARG_TYPE_ActionCaller* state variable" for details**. If the *ActionCaller* argument specifies a *deviceUDN* then this action is invoked by a partner device and the caller does not need to disseminate pair information to the partner device. However, if the *ActionCaller* argument is *null* then the device that receives this action MUST invoke the *AddSyncPair()* action on the partner device to maintain identical pair information on the partner.

The *ObjectID* input argument of the action identifies the object to which the pairGroup information is being added.

The *SyncPair* input argument includes an XML fragment containing the pair information. See "Clause A.3 Content Synchronization-related Properties" for details.

There are three possible scenarios that may occur while invoking this action while adding pairGroup information for an object:

- Both objects that are part of the pair already exist (Scenario 1):

  The *object@id* value of the remote object MUST be included in the *avcs:pair::remoteObjID* element.
- Object on only one of the partner exists (Scenario 2 and Scenario 3):
  - The partner device does not have a corresponding partner object for the pair. The object will be created on the partner device during the first synchronization operation. The rules to create an object on the partner device are as follows:
    - If the parent container object under which the new object item will be created exists in the partner device, the *avcs:pair::remoteParentObjID* element MUST include the *object@id* value of the parent container object.
    - If the parent container object does not exist in the partner and the container object under which the new object item will be located is to be created from the local container object, the *avcs:pair::virtualRemoteParentObjID* element MUST include the *object@id* value of the local container (i.e. this container object will be the parent object in the partner device.) This parent container object MUST have an pair in the same partnership as well. During the synchronization operation, if a local device determines that no corresponding object exist in the local device for the pair, the device MUST create a new object in the local device and MUST update the *avcs:pair::RemoteObjID* by assigning the value of the object ID of the newly created object. The device then MUST delete the *avcs:pair::remoteParentObjID* or the *avcs:pair::virtualRemoteParentObjID*.

A result of the *AddSyncPair()* action is that the *avcs:pair* property is added into *avcs:syncInfo* property of the CDS object. If this is the first synchronization pair for this object the *avcs:syncInfo* property MUST be created first and then the *avcs:pair* property is added into it.

An object can be part of multiple pairGroups within a single synchronization relationship. But, the following rules apply in such cases:

- If the synchronization policy is 'replace', the object that is a source is allowed to have multiple pairs.
- If the synchronization policy is 'blend', the object with precedence is allowed to have multiple pairs.
- If the synchronization policy is 'merge', only single pairGroup is allowed.

However, the rules above are not applied between multiple synchronization relationships.

The *avcs:pair::policy* property overrides any policy that are specified in the upper level hierarchy of the synchronization relationship structure.

2.9.6.1 Arguments

Table 2-15 — Arguments for *AddSyncPair()*

Argument	Direction	Related State Variable
ActionCaller	*IN*	*A_ARG_TYPE_ActionCaller*
ObjectID	*IN*	*A_ARG_TYPE_ObjectID*
SyncPair	*IN*	*A_ARG_TYPE_SyncPair*

2.9.6.2 Dependency on State

None.

2.9.6.3 Effect on State

None.

2.9.6.4 Errors

Table 2-16 — Error Codes for *AddSyncPair()*

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture clause on Control.
500-599	TBD	See UPnP Device Architecture clause on Control.
600-699	TBD	See UPnP Device Architecture clause on Control.
703	Invalid action caller	The *AddSyncPair()* request failed because the action caller is a part of the sync data.
704	Partner Timeout	The *AddSyncPair()* request failed because the sync data structure could not be add due to time out of the partner device.
705	Partner not online	The *AddSyncPair()* request failed because partner device is not in the network.
708	Invalid object	The *AddSyncPair()* request failed because the specified *ObjectID* argument is invalid.
709	Invalid pair	The *AddSyncPair()* request failed because the specified *SyncPair* argument is invalid.

2.9.7 *ModifySyncPair()*

The *ModifySyncPair()* action modifies the synchronization pair property for a CDS object.

This modification includes only the policy information. All other modifications are not allowed.

The *ActionCaller* argument identifies the *deviceUDN* of the caller. **See Clause "2.7.3 *A_ARG_TYPE_ActionCaller* state variable" for details**. If the *ActionCaller* argument specifies a *deviceUDN* then this action is invoked by a partner device and the caller does not need to disseminate pairGroup information to the partner device. However, if the *ActionCaller* argument is *null* then the action is called by a stand-alone control point and to maintain identical pair information on the partner device, the device MUST disseminate the pair information included in the *SyncPair* to the partner device, by invoking the *ModifySyncPair()* action on the partner.

The *ObjectID* argument identifies the object whose pair information is to be modified. The *SyncPair* input argument includes an XML fragment containing the pair information to be added.

The *SyncPair* input argument includes an XML fragment containing the pair information. In the *SyncPair* argument, *avcs:pair@syncRelationshipID*, *avcs:pair@partnershipID* and *avcs:pair@pairGroupID* MUST be specified and valid. A *SyncPair* also includes either a *avcs:pair::remoteObject* or a *avcs:pair::remoteParentObjID* or a *avcs::virtualRemoteParentObjID* MUST be specified. **See "Clause A.3 Content Synchronization-related Properties" for details.**

2.9.7.1 Arguments

Table 2-17 — Arguments for *ModifySyncPair()*

Argument	Direction	Related State Variable
ActionCaller	*IN*	*A_ARG_TYPE_ActionCaller*
ObjectID	*IN*	*A_ARG_TYPE_ObjectID*
SyncPair	*IN*	*A_ARG_TYPE_SyncPair*

2.9.7.2 Dependency on State

None.

2.9.7.3 Effect on State

None.

2.9.7.4 Errors

Table 2-18 — Error Codes for *ModifySyncPair()*

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture clause on Control.
500-599	TBD	See UPnP Device Architecture clause on Control.
600-699	TBD	See UPnP Device Architecture clause on Control.
703	Invalid action caller	The *ModifySyncPair()* request failed because the action caller is a part of the sync data.
704	Partner Timeout	The *ModifySyncPair()* request failed because the sync data structure could not be modified due to time out of the partner device.
705	Partner not online	The *ModifySyncPair()* request failed because partner device is not in the network.
708	Invalid object	The *ModifySyncPair()* request failed because the specified *ObjectID* argument is invalid.
709	Invalid pair	The *ModifySyncPair()* request failed because the specified *SyncPair* argument is invalid.

2.9.8 *DeleteSyncPair()*

The *DeleteSyncPair()* action sets the value of the *avcs:syncInfo::pair::status* property of a synchronizing object to "*EXCLUDED*".

The *ActionCaller* argument identifies the *deviceUDN* of the caller. **See Clause "2.7.3 *A_ARG_TYPE_ActionCaller* state variable" for details.** If the *ActionCaller* argument specifies a *deviceUDN* then this action is invoked by a partner device and the caller does not need to disseminate pair information to the partner device. However, if the ActionCaller argument is *null* then the action is called by a stand-alone control point and to maintain identical pair information on the partner device, the device MUST disseminate the pair information included in the SyncPair to the partner device, by invoking the DeleteSyncPair() action on the partner.

The ObjectID argument identifies the CDS object in which avcs:syncInfo::pair::status property is to be set.

The SyncID input argument identifies the target of the deletion. If the SyncID identifies a synchronization relationship then all pairs that are associated with the relationship MUST be deleted. . If the SyncID identifies a synchronization partnership, then all pairs that are associated with the partnership MUST be deleted. . If the SyncID identifies a synchronization pairGroup, then all pairs associated with the pairGroup MUSt be deleted.

Once the status property is set to "*EXCLUDED*", the avcs:pair property of the object is deleted during the next synchronization operation and thereby the object is permanently excluded from the synchronization relationship.

2.9.8.1 Arguments

Table 2-19 — Arguments for DeleteSyncPair()

Argument	Direction	Related State Variable
ActionCaller	IN	A_ARG_TYPE_ActionCaller
ObjectID	IN	A_ARG_TYPE_ObjectID
SyncID	IN	A_ARG_TYPE_SyncID

2.9.8.2 Dependency on State

None.

2.9.8.3 Effect on State

None.

2.9.8.4 Errors

Table 2-20 — Error Codes for DeleteSyncPair()

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture clause on Control.
500-599	TBD	See UPnP Device Architecture clause on Control.
600-699	TBD	See UPnP Device Architecture clause on Control.
701	No such sync data	The DeleteSyncPair() request failed because the specified SyncID argument is invalid.
703	Invalid action caller	The DeleteSyncPair() request failed because the action caller is a part of the sync data.
704	Partner Timeout	The DeleteSyncPair() request failed because the sync data structure could not be modified due to time out of the partner device.
705	Partner not online	The DeleteSyncPair() request failed because partner device is not in the network.
708	No such object	The DeleteSyncPair() request failed because the specified ObjectID argument is invalid.

2.9.9 *StartSync()*

The *StartSync()* action triggers a synchronization operation which is performed asynchronously in other words, the action MAY return to the caller before the synchronization operation completes (or even before the synchronization starts). The status of the synchronization operation can be monitored through the eventing of the *SyncStatusUpdate* state variable or via the *GetSyncStatus()* action. The *SyncStatusUpdate* state variable contains incremental synchronization status information which is evented and the *GetSyncStatus()* action returns the value of the *SyncStatus* state variable which contains the accumulation of all synchronization status information from when the synchronization operation was started.

When the *StartSync()* action is invoked, the device prepares itself for the synchronization operation e.g., locking internal data structure and returns to the caller. After returning to the caller, the device asynchronously performs the synchronization operation. The caller of the action may leave the network anytime without effecting the synchronization operation.

The *ActionCaller* argument identifies the *deviceUDN* of the caller. **See Clause "2.7.3 *A_ARG_TYPE_ActionCaller* state variable" for details**. The *ActionCaller* argument also determines the behavior of the *StartSync()* action. If the *ActionCaller* argument is *the empty string* then the device MUST invoke the *StartSync()* action on the partner(s) to trigger synchronization operations on the partner(s). The list of partners can be determined from the synchronization relationship that contains the specified *SyncID.* However, If the value of *ActionCaller* argument is set to a *deviceUDN*, then the device MUST NOT invoke the *StartSync()* action on the partner(s).

The *SyncID* action argument identifies what is to be synchronized. If the *SyncID* argument identifies a pairGroup then only that specific pairGroup MUST be synchronized. If the *SyncID* argument identifies a partnership then all pairGroups within that specific partnership MUST be synchronized. If the *SyncID* argument identifies a synchronization relationship then all pairGroups within each partnership contained within that specific relationship MUST be synchronized.

2.9.9.1 Arguments

Table 2-21 — Arguments for *StartSync()*

Argument	Direction	Related State Variable
ActionCaller	*IN*	*A_ARG_TYPE_ActionCaller*
SyncID	*IN*	*A_ARG_TYPE_SyncID*

2.9.9.2 Dependency on State

None.

2.9.9.3 Effect on State

None.

2.9.9.4 Errors

Table 2-22 — Error Codes for *StartSync()*

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture clause on Control.
500-599	TBD	See UPnP Device Architecture clause on Control.
600-699	TBD	See UPnP Device Architecture clause on Control.
701	No such sync data	The *StartSync()* request failed because the specified *SyncID* argument is invalid.
703	Invalid action caller	The *StartSync()* request failed because the action caller is a part of the sync data.
704	Partner Timeout	The *StartSync()* request failed because the sync operation could not progress due to time out of the partner device.
705	Partner not online	The *StartSync()* request failed because partner device is not in the network.
710	Inactive state	The *StartSync()* request failed because the specified *SyncID* argument is not active.
711	Sync operation in-progress	The *StartSync()* request failed because the sync operation of the specified sync data is in-progress.
712	Invalid Sync operation invocation	The *StartSync()* request failed because the relationship contain a non-CDS partner.

2.9.10 *AbortSync()*

This action cancels an active synchronization operation which is being performed asynchronously.

The *ActionCaller* argument identifies the *deviceUDN* of the caller. **See Clause "2.7.3 *A_ARG_TYPE_ActionCaller* state variable" for details**. The *ActionCaller* argument also determines the behavior of the *AbortSync()* action. If the *ActionCaller* argument is *the empty string* then the device MUST invoke the *AbortSync()* action on the partner(s) to abort synchronization operations on the partner(s). The list of partners can be determined from the synchronization relationship that contains the specified *SyncID.* However, If the value of *ActionCaller* argument is set to a *deviceUDN*, then the device MUST NOT invoke the *AbortSync()* action on the partner(s).

The *SyncID* action argument identifies which on-going synchronization operation is to be aborted. The *SyncID* argument is the same as was used to start the synchronization operation via the *StartSync()* action.

When a device aborts a synchronization operation, the CDS MUST be left in a fully consistent state. When aborting an implementation MUST NOT roll back any changes that already have been exposed on the network. Consequently, the resulting Content Directory database will reflect either the state prior to the synchronization or partial synchronization.

2.9.10.1 Arguments

Table 2-23 — Arguments for *AbortSync()*

Argument	Direction	Related State Variable
ActionCaller	*IN*	*A_ARG_TYPE_ActionCaller*
SyncID	*IN*	*A_ARG_TYPE_SyncID*

2.9.10.2 Dependency on State

None.

2.9.10.3 Effect on State

None.

2.9.10.4 Errors

Table 2-24 — Error Codes for *AbortSync()*

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture clause on Control.
500-599	TBD	See UPnP Device Architecture clause on Control.
600-699	TBD	See UPnP Device Architecture clause on Control.
701	No such sync data	The *AbortSync()* request failed because the specified *SyncID* argument is invalid.
703	Invalid action caller	The *AbortSync()* request failed because the action caller is a part of the sync data.
704	Partner Timeout	The *AbortSync()* request failed because the sync operation could not progress due to time out of the partner device.
705	Partner not online	The *AbortSync()* request failed because partner device is not in the network.

2.9.11 *GetChangeLog()*

2.9.12

This action allows a caller to get all the objects that have changed since the last synchronization operation. The *SyncID* identifies a synchronization relationship or a partnership or a pairGroup for which changed objects to be retrieved.

StartingIndex is zero-based offset to enumerate changed objects associated with *SyncID*.

RequestedCount is requested number of entries under the change log associated with *SyncID*. *RequestedCount* = 0 indicates request all entries.

A CDS MUST keep track of all objects since the last synchronization operation to provide response to the *GetChangeLog()* action. The deleted objects information can be deleted when the synchronization relationship, partnership or pairGroup that the objects belong to are deleted.

The *Result* output argument includes all the objects that have changed since the last synchronization operation. The format of the *Result* argument is represented by the *A_ARG_TYPE_ChangeLog*.

NumberReturned is number of objects returned in the *Result* argument.

TotalMatches MUST be set to the total number of objects in the changed log specified for the *GetChangeLog()* action (independent of the starting index specified by the *StartingIndex* argument). If the ContentSync service implementation cannot timely compute the value of *TotalMatches*, but there are matching objects that have been found by the ContentSync service implementation, then the *GetChangeLog()* action MUST successfully return with the *TotalMatches* argument set to zero and the *NumberReturned* argument indicating the number of returned objects. If the ContentSync service implementation cannot timely compute the value of *TotalMatches*, and there are no matching objects found, then the *GetChangeLog()* action MUST return error code 712.

2.9.12.1 Arguments

Table 2-25 — Arguments for *GetChangeLog()*

Argument	Direction	Related State Variable
SyncID	*IN*	*A_ARG_TYPE_SyncID*
StartingIndex	*IN*	*A_ARG_TYPE_Index*
RequestedCount	*IN*	*A_ARG_TYPE_Count*
Result	*OUT*	*A_ARG_TYPE_ChangeLog*
NumberReturned	*OUT*	*A_ARG_TYPE_Count*
TotalMatches	*OUT*	*A_ARG_TYPE_Count*

2.9.12.2 Dependency on State

None.

2.9.12.3 Effect on State

None.

2.9.12.4 Errors

Table 2-26 — Error Codes for *GetChangeLog()*

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture clause on Control.
500-599	TBD	See UPnP Device Architecture clause on Control.
600-699	TBD	See UPnP Device Architecture clause on Control.
701	No such sync data	The *GetChangeLog()* request failed because the specified *SyncID* argument is invalid.
720	Cannot process the request	The *GetChangeLog()* request failed because the ContentSync service is unable to compute, in the time allotted, the total number of objects that are a match for the synchronization ID and is additionally unable to return, in the time allotted, any objects that match the synchronization ID.

2.9.13 *ResetChangeLog()*

This action allows a caller to clear the existing change log of synchronization objects and start keeping new logs.

The *SyncID* identifies a pairGroup for which the change log of all objects included in the identified synchronization data structure will be cleared.

The *ObjectIDs* argument identifies individual objects for which the change logs will be cleared and contains one or more CDS object IDs. In other word, the device MUST clear the change logs of multiple objects within the action processing period if the *ObjectIDs* argument has multiple object IDs. When all synchronization objects that are involved with a pairGroup or a partnership or a relationship are to be cleared, this input argument should have the value of "*". Otherwise, this argument should have a list of individual *object@id*. In order to clear the change log of the objects that are involved in different pairGroups, the *SyncID* should be empty string.

By comparing the *@updateID* in the *ObjectIDs* input argument and *avcs:pair@updateID* of the CDS object, the device can determine whether the CDS object has changed. A caller keeps this *@updateID* value until it invokes the *ResetChangeLog()* action. Whenever the property of a synchronization object is changed, the CDS MUST increase the @updateID property of the object by 1. The device sets the avcs:pair:status value as "MODIFIED" if two values (@updateID in the ObjectIDs input argument and avcs:pair@updateID of the CDS object) are different from each other. Otherwise, the value of avcs:pair:status property is reset to “SYNC’ED”. This process removes the necessity for the device to lock object properties while it is synchronizing.

2.9.13.1 Arguments

Table 2-27 — Arguments for ResetChangeLog()

Argument	Direction	Related State Variable
SyncID	IN	A_ARG_TYPE_SyncID
ObjectIDs	IN	A_ARG_TYPE_ResetObjectList

2.9.13.2 Dependency on State

None.

2.9.13.3 Effect on State

None.

2.9.13.4 Errors

Table 2-28 — Error Codes for ResetChangeLog()

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture clause on Control.
500-599	TBD	See UPnP Device Architecture clause on Control.
600-699	TBD	See UPnP Device Architecture clause on Control.
701	No such sync data	The ResetChangeLog() request failed because the specified SyncIDs argument is invalid.

2.9.14 ResetStatus()

2.9.15

This action allows a caller to reset status of synchronization pairs that are bound to a synchronization relationship or partnership or pairGroup regardless of current status of the pair. In other words, the action changes the value of status of a synchronization pair to "NEW" similar to as a newly created synchronization pair.

This action is only effective when the policy defined for a synchronization relationship or partnership or pairGroup is a *tracking* policy.

The SyncID identifies a synchronization relationship, or a partnership or a pairGroup for which the status of change log of all objects included in the identified synchronization data structure will be reseted.

2.9.15.1 Arguments

Table 2-29 — Arguments for ResetStatus()

Argument	Direction	Related State Variable
SyncID	IN	A_ARG_TYPE_SyncID

2.9.15.2 Dependency on State

None.

2.9.15.3 Effect on State

None.

2.9.15.4 Errors

Table 2-30 — Error Codes for *ResetStatus()*

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture clause on Control.
500-599	TBD	See UPnP Device Architecture clause on Control.
600-699	TBD	See UPnP Device Architecture clause on Control.
701	Invalid *SyncID*	The *ResetStatus()* request failed because the specified *SyncID* argument is invalid.

2.9.16 *GetSyncStatus()*

This action returns the status of the current synchronization operation identified by the *SyncID*. If the synchronization operation is completed, then invocation of this action returns the status of the last synchronization operation. Therefore, the device MUST keep status of the last synchronization operation until the next synchronization operation starts.

If the action argument *SyncID* identifies a synchronization relationship then the *SyncStatus* output argument contains the current value of the *A_ARG_TYPE_SyncStatus* state variable. If the action argument *SyncID* identifies a partnership then the *SyncStatus* output argument contains the status information of that specific partnership contained in the *A_ARG_TYPE_SyncStatus* state variable. If the action argument *SyncID* identifies a pairGroup then the *SyncStatus* output argument contains the status information of that specific pairGroup contained in the *A_ARG_TYPE_SyncStatus* state variable.

2.9.16.1 Arguments

Table 2-31 — Arguments for *GetSyncStatus()*

Argument	Direction	Related State Variable
SyncID	*IN*	*A_ARG_TYPE_SyncID*
SyncStatus	*OUT*	*A_ARG_TYPE_SyncStatus*

2.9.16.2 Dependency on State

None.

2.9.16.3 Effect on State

None.

2.9.16.4 Errors

Table 2-32 — Error Codes for *GetSyncStatus()*

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture clause on Control.
500-599	TBD	See UPnP Device Architecture clause on Control.
600-699	TBD	See UPnP Device Architecture clause on Control.
701	No such sync data	The *StartSync()* request failed because the specified *SyncID* argument is invalid.

2.9.17 Non-Standard Actions Implemented by a UPnP Vendor

To facilitate certification, non-standard actions implemented by UPnP vendors should be included in this service template. The UPnP Device Architecture lists naming requirements for non-standard actions (see the clause on Description).

2.9.18 Common Error Codes

The following table lists error codes common to actions for this service type. If a given action results in multiple errors, the most specific error MUST be returned.

Table 2-33 — Common Error Codes

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture clause on Control.
500-599	TBD	See UPnP Device Architecture clause on Control.
600-699	TBD	See UPnP Device Architecture clause on Control.
701	No such sync data	The action request failed because the specified sync data is invalid.
702	Invalid XML	The action failed because given XML fragment violates the XML schema
703	Invalid action caller	The action failed because the action caller is a part of the sync data.
704	Partner Timeout	The action failed because the sync data structure could not be modified due to time out of the partner device.
705	Partner not online	The action failed because partner device is not in the network.
706	Update in-progress	The action failed because another action request is still being processed.
707	Stale data	The action failed because the sync data is stale.
708	No such object	The action failed because the specified object is invalid.
709	Invalid pair	The action failed because the specified sync pair is invalid.
710	Inactive state	The action failed because given sync data is inactive.
711	Sync operation in-progress	The action failed because the sync operation of the specified sync data is in-progress.
712	Cannot process the request	The action failed because the ContentSync service was unable to complete the necessary computations in the time allotted.

2.10 Theory of Operation

2.10.1 Introduction

This clause shows several scenarios to illustrate the various actions supported by the ContentSync service. These include synchronization relationship creation and deletion, transferring synchronization data structures in preparation for a synchronization operation, performing a synchronization operation and terminating a synchronization operation.

2.10.2 CDS Synchronization

In order to synchronize objects between two CDSs, a control point must first establish a synchronization relationship and then create synchronization pair(s) that determines which CDS object(s) will be synchronized. After creating the synchronization relationship and its synchronization pair(s), the control point can execute the synchronization operation. Once the initial synchronization operation is successfully done (to establish the basis of a common set of objects between the partners), two CDSs maintain same synchronization pair(s) information for the CDS synchronization object. When an object is changed after the synchronization, the CDS implementation must recognize which object is changed and then provide information about that object as a change log when the next synchronization is triggered by the control point.

The following shows the example sequence of the lifetime of synchronization.

Example Sequence

- Synchronization setup
  - Creation of minimally complete synchronization data structure
  - Creation of synchronization pair.
    - Creation of policy for an synchronization pair (i.e. A3) that overrides partnership policy
- Trigger first synchronization operation
- Update Objects in partner 1
- Update (delete) Objects in partner 2
- Trigger subsequent synchronization operation
- Update synchronization data structure
- Synchronize synchronization data structure

2.10.2.1 Synchronization Setup

In order to demonstrate how the ContentSync service works, let us consider the following logical structure of two separate CDSs. The content in Partner 1 includes two music items and one container. The content in Partner 2 includes only a single music item. The logical directory hierarchies for each CDS are presented as follows:

[Partner 1 CDS hierarchy]

- Name="Content", ID="A0"
  - Name="Would - Alice In Chains.wma", ID="A1", Size="90000"
  - Name="My Music", ID="A2"
    - Name="Chloe Dancer - Mother Love Bone.mp3", ID="A3", Size="200000"

[Partner 2 CDS hierarchy]

- Name="My Multimedia Content", ID="B0"

- Name="Alice In Chains", ID="B1", Size="90000"
- Name="Wonder - Tell Me", ID="B4", Size="500000"

Figure 8 shows a visual representation of two CDS hierarchies above and which object is to be synchronized (see a solid line between objects) in order to provide better understanding of the example.

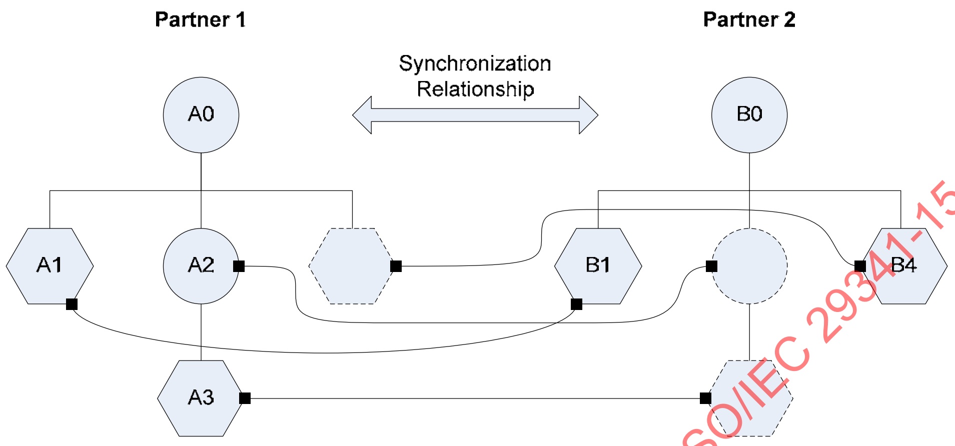

Figure 8 — Synchronization Relationship between two CDSs

Let us consider that object A1 and B1 are to be synchronized with each other and object A2 is to be synchronized under object B0 and object A3 is to synchronize under a new container object which will be created while synchronizing object A2 and object B4 is to be synchronized under object A0. In order to perform this synchronization operation, a control point first MUST create a new synchronization relationship and then, create new synchronization pairs for each of the three objects that are to be synchronized. Metadata and resources for all items in this example are expected to synchronize (i.e., each item has *avcs:pair* and *res@avcs:syncAllowed* properties.). Let us also assume that the example will create only single partnership and single pairGroup in the synchronization relationship.

The following subclauses describe how a synchronization relationship between two devices is established and synchronization objects in these two devices are synchronized with each other. Table 2-34 summarizes which action will be invoked for each step of sequence in this example.

Table 2-34 — Actions for example sequence

Task	Related Action	Task Result
Creating a synchronization data structure	*AddSyncData()*	1 pairGroup within 1 partnership within 1 synchronization relationship
Creating a synchronization pair	*AddSyncPair()*	4 synchronization pairs as shown below. Control point invokes invokes *AddSyncPair()* on Partner 1 for pairs 1, 2, and 3. Control point invokes *AddSyncPair()* on partner 2 for 4. 1. A1-B1 pair 2. A2-new object (B2) (that will be created under the container object (B0)) 3. A3-new object (B3) (that will be created under the object B2) 4. B4-new object (A4) (that will be created under the container object (A0))
Initiating a synchronization operation	*StartSync()*	Patners start the operation
Performing a synchronization operation	*GetChangeLog()*	Updated CDS
Resetting a change log	*ResetChangeLog()*	Reset a change log of an object

2.10.2.2 Creating a Complete Synchronization Data Structure

To create a synchronization relationship, the control point first invokes the *AddSyncData()* action with a minimally complete synchronization data structure on one of the synchronization partner devices. It is immaterial which parter of the relationship that the control point invokes the *AddSyncData()* action on, however, for this example it will be assumed to be Partner 1.

Request: (Control point to Partner 1)

```
AddSyncData("", "", "
<syncRelationship id="" active="1" xmlns="urn:schemas-upnp-org:cs"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:schemas-upnp-org:cs
   http://www.upnp.org/schemas/cs/cs-v1-20070XXXX.xsd">
 <title>Sync between My MP3P and Home Media Server</title>
 <partnership id="" active="1">
  <partner id="1">
   <deviceUDN>343bd2a2-189b-40c0-8eb5-ea91ea730402</deviceUDN>
   <serviceID>service_ID_A</serviceID>
  </partner>
  <partner id="2">
   <deviceUDN>05de2732-5df5-4c48-922b-12f73473f0e9</deviceUDN>
   <serviceID>service_ID_B</serviceID>
  </partner>
  <policy>
   <syncType>merge<syncType>
   <priorityPartnerID>1</priorityPartnerID>
  </policy>
  <pairGroup id="" active="1"></pairGroup>
 </partnership>
</syncRelationship>
");
```

Upon receiving this action invocation, Partner 1 will initialize its internal data structures for the relationship and then Partner 1 invokes the *AddSyncData()* action on Partner 2 with its *deviceUDN* as the *ActionCaller* argument in order to propagate the synchronization data structure to Partner 2. The difference between the call from the control point to Partner 1 to initialize the relationship and the call between Partner 1 and Partner 2 to propagate this information is that the latter invocation of *AddSyncData()* includes device UDN as the first argument of the action and the values of synchronization relationship, partnership and pairGroup ID are not the empty strings. Receiving this call, Partner 2 understands by the initialized values of the synchronization relationship, partnership and pairGroup IDs and by the presence of a *deviceUDN* as the *ActionCaller* argument that this is an initialized synchronization relationship and it does not need to propagate the information further.

Request: (Partner 1 to Partner 2)

```
AddSyncData("343bd2a2-189b-40c0-8eb5-ea91ea730402", "", "
<syncRelationship id="d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d" active="1"
  xmlns="urn:schemas-upnp-org:cs"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:schemas-upnp-org:cs
   http://www.upnp.org/schemas/cs/cs-v1-20070XXXX.xsd">
 <title>Sync between My MP3P and Home Media Server</title>
 <partnership id="a0e4d0a7-3378-4f17-8af2-3f7de3345dc6" active="1">
  <partner id="1">
   <deviceUDN>343bd2a2-189b-40c0-8eb5-ea91ea730402</deviceUDN>
   <serviceID>service_ID_A</serviceID>
  </partner>
  <partner id="2">
   <deviceUDN>05de2732-5df5-4c48-922b-12f73473f0e9</deviceUDN>
   <serviceID>service_ID_B</serviceID>
  </partner>
  <policy>
   <syncType>merge<syncType>
   <priorityPartnerID>1</priorityPartnerID>
  </policy>
  <pairGroup id="ba8e57de-7f66-4102-ae4b-31b96c86f173" active="1"/>
 </partnership>
</syncRelationship>
");
```

At this point, Partner 2 intializes its internal data structures with the relationship and returns a success or failure response to Partner 1.

Response: (Partner 2 to Partner 1)

```
AddSyncData("
<syncRelationship id="d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d" active="1"
  xmlns="urn:schemas-upnp-org:cs"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:schemas-upnp-org:cs
   http://www.upnp.org/schemas/cs/cs-v1-20070XXXX.xsd">
 <title>Sync between My MP3P and Home Media Server</title>
 <partnership id="a0e4d0a7-3378-4f17-8af2-3f7de3345dc6" active="1">
  <partner id="1">
   <deviceUDN>343bd2a2-189b-40c0-8eb5-ea91ea730402</deviceUDN>
   <serviceID>service_ID_A</serviceID>
  </partner>
  <partner id="2">
   <deviceUDN>05de2732-5df5-4c48-922b-12f73473f0e9</deviceUDN>
   <serviceID>service_ID_B</serviceID>
  </partner>
  <policy>
   <syncType>merge<syncType>
   <priorityPartnerID>1</priorityPartnerID>
  </policy>
  <pairGroup id="ba8e57de-7f66-4102-ae4b-31b96c86f173" active="1"/>
 </partnership>
</syncRelationship>
");
```

Only after receiving the response from the call to Partner 2 can Partner 1 responsd to the original *AddSyncData()* action.

Response: (Partner 1 to Control Point)

```
AddSyncData("
<syncRelationship id="d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d" active="1"
```

```
  xmlns="urn:schemas-upnp-org:cs"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:schemas-upnp-org:cs
   http://www.upnp.org/schemas/cs/cs-v1-20070XXXX.xsd">
 <title>Sync between My MP3P and Home Media Server</title>
 <partnership id="a0e4d0a7-3378-4f17-8af2-3f7de3345dc6" active="1">
  <partner id="1">
   <deviceUDN>343bd2a2-189b-40c0-8eb5-ea91ea730402</deviceUDN>
   <serviceID>service_ID_A</serviceID>
  </partner>
  <partner id="2">
   <deviceUDN>05de2732-5df5-4c48-922b-12f73473f0e9</deviceUDN>
   <serviceID>service_ID_B</serviceID>
  </partner>
  <policy>
   <syncType>merge<syncType>
   <priorityPartnerID>1</priorityPartnerID>
  </policy>
  <pairGroup id="ba8e57de-7f66-4102-ae4b-31b96c86f173" active="1"/>
 </partnership>
</syncRelationship>
");
```

2.10.2.3 Creating a Synchronization Pair

After creating the synchronization relationship, the control point creates the necessary synchronization object pairs and associates the pairs with the synchronization relationship. In this example, object A1 and object B1 are to be synchronized with each other and objects A2 and A3 are to be synchronized with the objects that will be created during the synchronization operation.

Creating the pair between object A1 and B1

In the next step of the process, the control point establishes the synchronization pairs that will be used as part of the synchronization relationship. In order to do this, the control point invokes the *AddSyncPair()* action, on one of the Partners. It must specify the local ID of the object on the partner that it is invoking as a parameter. The object ID on the other partner is carried as an element of the sync pair data structure. Note that the pair between A1 and B1 is Scenario 1 pairing according to the definition of a synchronization pair, where both objects currently exist within their respective CDS.. **See 2.2.1 Synchronization Object and Pair for details.**

Request: (Control Point to Partner 1)

```
AddSyncPair("", "A1", "
<avcs:pair xmlns:avcs="urn:schemas-upnp-org:cs:avcs"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:schemas-upnp-org:cs:avcs
   http://www.upnp.org/schemas/cs/avcs-v1-2007xxxx.xsd"
  syncRelationshipID="d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d"
  partnershipID="a0e4d0a7-3378-4f17-8af2-3f7de3345dc6"
  pairGroupID="ba8e57de-7f66-4102-ae4b-31b96c86f173">
 <avcs:remoteObjID>B1</avcs:remoteObjID>
</avcs:pair>
");
```

Note: a control can know 3 input IDs for the *avcs:pair* property by invoking the GetSyncData() action of which purpose is to retrieve a synchronization data structure kept by the device.

Upon receiving the invocation of the *AddSyncPair()* action, the Partner must initialize its internal data structures and propagate the call to the other partner in the relationship. In order to create a pair, the partner 1 device which received the *AddSyncPair()* action invokes the *AddSyncPair()* action on the partner 2 based-on the information described above. The *AddSyncPair()* action call to propagate the call from the control point includes device UDN as the first argument of the action. **See clause 2.9.6 for details on *AddSyncPair()* action.**

Request: (Partner 1 to Partner 2)

```
AddSyncPair("343bd2a2-189b-40c0-8eb5-ea91ea730402", "B1", "
<avcs:pair xmlns:avcs="urn:schemas-upnp-org:cs:avcs"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:schemas-upnp-org:cs:avcs
   http://www.upnp.org/schemas/cs/avcs-v1-2007xxxx.xsd"
  syncRelationshipID="d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d"
  partnershipID="a0e4d0a7-3378-4f17-8af2-3f7de3345dc6"
  pairGroupID="ba8e57de-7f66-4102-ae4b-31b96c86f173">
 <avcs:remoteObjID>A1</avcs:remoteObjID>
</avcs:pair>
");
```

Response: (Partner 2 to Partner 1)

```
AddSyncPair("")
```

After receiving the second *AddSyncPair()* action response, the partner 1 responds to the first *AddSyncPair()* action.

Response: (Parnter 1 to Control Point)

```
AddSyncPair();
```

Creating the pair for object A2

In this case, only one *AddSyncPair()* action must be invoked on the partner 1 because there is no corresponding remote object on partner 2. In other words, the partner 1 does not invoke the second *AddSyncPair()* action to propagate the pair information. Instead of the *remoteObjID* property, object A2 includes the *remoteParentObjID* property.

Request: (Control Point to Partner 1)

```
AddSyncPair("", "A2", "
<avcs:pair xmlns:avcs="urn:schemas-upnp-org:cs:avcs"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:schemas-upnp-org:cs:avcs
   http://www.upnp.org/schemas/cs/avcs-v1-2007xxxx.xsd"
  syncRelationshipID="d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d"
  partnershipID="a0e4d0a7-3378-4f17-8af2-3f7de3345dc6"
  pairGroupID="ba8e57de-7f66-4102-ae4b-31b96c86f173">
 <remoteParentObjID>B0</avcs:remoteParentObjID>
</avcs:pair>
");
```

Response: (Parnter 1 to Control Point)

```
AddSyncPair();
```

Creating the pair for object A3 with the policy overriding

In this case, the *AddSyncPair()* action is invoked on the partner 1 only because the partner 2 does not contain the object to be synchronized with. In other words, the partner 1 does not invoke the second *AddSyncPair()* action to propagate pair information. Instead of the *remoteObjID* property, object A2 includes the *virtualRemoteParentObjID* property.

Request: (Control Point to Parnter 1)

```
AddSyncPair("", "A3", "
<avcs:pair xmlns:avcs="urn:schemas-upnp-org:cs:avcs"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:schemas-upnp-org:cs:avcs
   http://www.upnp.org/schemas/cs/avcs-v1-2007xxxx.xsd"
  syncRelationshipID="d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d"
  partnershipID="a0e4d0a7-3378-4f17-8af2-3f7de3345dc6"
  pairGroupID="ba8e57de-7f66-4102-ae4b-31b96c86f173">
 <avcs:virtualRemoteParentObjID>A2</avcs:virtualRemoteParentObjID>
 <avcs:policy>
  <avcs:syncType>replace</avcs:syncType>
  <avcs:priorityPartnerID>1</avcs:priorityPartnerID>
 </avcs:policy>
</avcs:pair>
");
```

Response: (Partner 1 to Control Point)

```
AddSyncPair();
```

Creating the pair for object B4

Similar to creating the pair for object A2, only one *AddSyncPair()* action must be invoked on the partner 2 because there is no corresponding remote object on partner 1.

Request: (Control Point to Partner 2)

```
AddSyncPair("", "B4", "
<avcs:pair xmlns:avcs="urn:schemas-upnp-org:cs:avcs"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:schemas-upnp-org:cs:avcs
  http://www.upnp.org/schemas/cs/avcs-v1-2007xxxx.xsd"
  syncRelationshipID="d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d"
  partnershipID="a0e4d0a7-3378-4f17-8af2-3f7de3345dc6"
  pairGroupID="ba8e57de-7f66-4102-ae4b-31b96c86f173">
 <avcs:remoteParentObjID>A0</avcs:remoteParentObjID>
</avcs:pair>
");
```

Response: (Parnter 2 to Control Point)

```
AddSyncPair();
```

By one *AddSyncData()* action and four *AddSyncPair()* action calls, the synchronization data structure for this example is established. After creating the synchronization data structure, a control point can trigger a synchronization operation with relationship ID or partnership ID or pairing ID at any time.

2.10.2.4 Synchronizing CDS

In order to synchronize two CDSs, a control point invokes the *StartSync()* action on either of the two partners as shown below. In this example, a synchronization relationship ID is used to trigger the CDS to start the synchronization operation.

Request: (Control Point to Partner 1)

```
StartSync("", "d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d");
```

The device that receives this action invokes *StartSync()* action on Partner 2 subsequently.

Request: (Partner 1 to Partner 2)

```
StartSync("343bd2a2-189b-40c0-8eb5-ea91ea730402", "d8c9fa13-d79b-4a0c-999b-
6ae2ff91a46d");
```

Response: (Parnter 2 to Partner 1)

```
StartSync();
```

After receiving the second action response, the partner 1 responds to the first *StartSync()* action.

Response: (Partner 1 to Control Point)

```
StartSync();
```

Getting Change Log:

After the partner devices respond to the *StartSync()* action successfully, the partner devices perform the synchronization operation simultaneously. To get synchronization objects as a change log, the embedded control points in the partners invoke *GetChangeLog()* action which is shown below.

Since during the first synchronization operation, some of the objects need to be created under a container object which itself needs to be created as well, the order how objects are to be synchronized should be handled very carefully.

The partner 1 gathers the *DIDL-Lite XML document* as a change log as shown below.

Request: (Partner 1 to Partner 2)

```
GetChangeLog("d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d", 0, 0);
```

Response: (Partner 2 to Partner 1)

```
GetChangeLog("
<?xml version="1.0" encoding="UTF-8"?>
<DIDL-Lite xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns="urn:schemas-upnp-org:metadata-1-0/DIDL-Lite/"
  xmlns:upnp="urn:schemas-upnp-org:metadata-1-0/upnp/"
  xmlns:avcs="urn:schemas-upnp-org:cs:avcs"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:schemas-upnp-org:metadata-1-0/DIDL-Lite/
  http://upnp.org/standardizeddcps/schemas/avwc/didl-lite/2.00
  urn:schemas-upnp-org:metadata-1-0/upnp/
  http://upnp.org/standardizeddcps/schemas/avwc/upnp/2.00
  urn:schemas-upnp-org:cs:avcs
  http://www.upnp.org/schemas/cs/avcs-v1-2007xxxx.xsd">
 <item id="B1" parentID="B0" restricted="1">
  <dc:title>Alice In Chains</dc:title>
  <upnp:class>object.item.audioItem.musicTrack</upnp:class>
  <res protocolInfo="http-get:*:audio/x-ms-wma:*" size="90000"
    avcs:syncAllowed="ALL" avcs:resModified="0">
   http://10.0.0.2/getcontent.asp?id=1
  </res>
  <avcs:syncable/>
  <avcs:syncInfo updateID="0">
   <avcs:pair
    syncRelationshipID="d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d"
    partnershipID="a0e4d0a7-3378-4f17-8af2-3f7de3345dc6"
    pairGroupID="ba8e57de-7f66-4102-ae4b-31b96c86f173">
    <avcs:remoteObjID>A1</avcs:remoteObjID>
    <avcs:status>NEW</avcs:status>
   </avcs:pair>
  </avcs:syncInfo>
 </item>
 <item id="B4" parentID="B0" restricted="1">
  <dc:title>Wonder - Tell Me</dc:title>
  <upnp:class>object.item.audioItem.musicTrack</upnp:class>
  <res protocolInfo="http-get:*:audio/mpeg:*" size="500000"
    avcs:syncAllowed="ALL" avcs:resModified="0">
   http://10.0.0.2/getcontent.asp?id=4
  </res>
  <avcs:syncable/>
  <avcs:syncInfo updateID="0">
   <avcs:pair
```

```
      syncRelationshipID="d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d"
      partnershipID="a0e4d0a7-3378-4f17-8af2-3f7de3345dc6"
      pairGroupID="ba8e57de-7f66-4102-ae4b-31b96c86f173">
     <avcs:remoteParentObjID>A0</avcs:remoteParentObjID>
     <avcs:status>NEW</avcs:status>
    </avcs:pair>
   </avcs:syncInfo>
  </item>
</DIDL-Lite>
", 1, 1);
```

When partner 1 processes the received change log, it does not update object A1 as the object in the partner 2 does not have any new properties that can be added to the object A1 in the partner 1. In this example, object A1 in partner 1 has the precedence over object B1 in partner 2 as defined in the synchronization policy. Therefore, the *dc:title* property of the partner 1 in this example is not updated by synchronization. In addition, there are no corresponding objects for object A2 and A3 in the change log because the corresponding objects will be created in the partner 2 by synchronization.

For object B4 in partner 2, partner 1 creates a new object under the container object A0 that is specified in the *remoteParentObjID* element of the *avcs:pair* in the change log as partner 1 does not have a corresponding pair object for object B4. While creating the new object, partner 1 accepts all properties and a resource from object B4. Let us assume that the newly created object has A4 as the value of *object@id* property. This object A4 is now associated with object B4, which means that the value of the *remoteObjID* element is B4 as shown below:

```
<item id="A4" parentID="A0" restricted="1">
 <dc:title>Wonder - Tell Me</dc:title>
 <upnp:class>object.item.audioItem.musicTrack</upnp:class>
 <res protocolInfo="http-get:*:audio/mpeg:*" size="500000"
   avcs:syncAllowed="ALL" avcs:resModified="0">
  http://10.0.0.1/getcontent.asp?id=4
 </res>
 <avcs:syncable/>
 <avcs:syncInfo updateID="0">
  <avcs:pair
    syncRelationshipID="d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d"
    partnershipID="a0e4d0a7-3378-4f17-8af2-3f7de3345dc6"
    pairGroupID="ba8e57de-7f66-4102-ae4b-31b96c86f173">
   <avcs:remoteObjID>B4</avcs:remoteObjID>
   <avcs:status>SYNC'ED</avcs:status>
  </avcs:pair>
 </avcs:syncInfo>
</item>
```

After finishing the update, partner 1 sends an event message to notify the status of the operation. The following is an example of an event message for object A1 and A4 in the partner 1.

GENA Message: (Partner 1 to Partner 2)

```
<?xml version="1.0" encoding="utf-8">
<SyncStatusUpdate xmlns="urn:schemas-upnp-org:cs"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:schemas-upnp-org:cs
   http://www.upnp.org/schemas/cs/cs-v1-20070XXXX.xsd">
 <syncRelationship id="d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d">
  <status numberOfTotalObjects="2" numberOfCompletedObjects="2"
    numberOfFailedObjects="0">
   COMPLETED_ALL
  </status>
  <partnership id="a0e4d0a7-3378-4f17-8af2-3f7de3345dc6">
   <status numberOfTotalObjects="2" numberOfCompletedObjects="2"
     numberOfFailedObjects="0">
    COMPLETED_ALL
   </status>
   <pairGroup id="ba8e57de-7f66-4102-ae4b-31b96c86f173">
    <status numberOfTotalObjects="2" numberOfCompletedObjects="2"
      numberOfFailedObjects="0">
     COMPLETED_ALL
    </status>
    <logEntry>
     <localObjectID>A1</localObjectID>
     <remoteObjectID>B1</remoteObjectID>
     <statusCode>001</statusCodes>
     <statusDescription>Succeeded completely</statusDescription>
    </logEntry>
    <logEntry>
     <localObjectID>A4</localObjectID>
     <remoteObjectID>B4</remoteObjectID>
     <statusCode>001</statusCodes>
     <statusDescription>Succeeded completely</statusDescription>
    </logEntry>
   </pairGroup>
  </partnership>
 </syncRelationship>
</SyncStatusUpdate>
```

The GENA event message above is sent for the entire change log entry regardless of synchronization status. However, the partner device explictly invokes the ResetChangeLog() action to Partner 2 in order to inform that an individual object in change log is successfully synchronized.

Request: (Partner 1 to Partner 2)

```
ResetChangeLog("d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d", "
<ResetObjectList xmlns="urn:schemas-upnp-org:cs"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:schemas-upnp-org:cs
   http://www.upnp.org/schemas/cs/cs-v1-20070XXXX.xsd">
 <object id="B1" remoteObjID="A1" updateID="0"/>
 <object id="B4" remoteObjID="A4" updateID="0"/>
</ResetObjectList>
");
```

Response: (Partner 2 to Partner 1)

```
ResetChangeLog();
```

When the partner 2 receives the ResetChangeLog() action, it can now change the value of the avcs:status property of the object B1 and B4 to "SYNC'ED"

The partner 2 gathers the *DIDL-Lite XML document* as change log as shown below:

Request: (Partner 2 to Partner 1)

GetChangeLog("d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d", 0, 0);

Response: (Partner 1 to Partner 2)

```
GetChangeLog("
<?xml version="1.0" encoding="UTF-8"?>
<DIDL-Lite xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns="urn:schemas-upnp-org:metadata-1-0/DIDL-Lite/"
  xmlns:upnp="urn:schemas-upnp-org:metadata-1-0/upnp/"
  xmlns:avcs="urn:schemas-upnp-org:cs:avcs"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:schemas-upnp-org:metadata-1-0/DIDL-Lite/
  http://upnp.org/standardizeddcps/schemas/avwc/didl-lite/2.00
  urn:schemas-upnp-org:metadata-1-0/upnp/
  http://upnp.org/standardizeddcps/schemas/avwc/upnp/2.00
  urn:schemas-upnp-org:cs:avcs
  http://www.upnp.org/schemas/cs/avcs-v1-2007xxxx.xsd">
 <item id="A1" parentID="0" restricted="1">
  <dc:title>Would - Alice In Chains.wma</dc:title>
  <upnp:class>object.item.audioItem.musicTrack</upnp:class>
  <res protocolInfo="http-get:*:audio/x-ms-wma:*"
    size="90000"avcs:syncAllowed="ALL" avcs:resModified="0">
   http://10.0.0.1/getcontent.asp?id=A1
  </res>
  <avcs:syncable/>
  <avcs:syncInfo updateID="0">
   <avcs:pair
     syncRelationshipID="d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d"
     partnershipID="a0e4d0a7-3378-4f17-8af2-3f7de3345dc6"
     pairGroupID="ba8e57de-7f66-4102-ae4b-31b96c86f173">
    <avcs:remoteObjID>B1</avcs:remoteObjID>
    <avcs:status>NEW</avcs:status>
   </avcs:pair>
  </avcs:syncInfo>
 </item>
 <item id="A2" parentID="A0" restricted="1">
  <dc:title>My Music</dc:title>
  <upnp:class>object.container.album</upnp:class>
  <avcs:syncable/>
  <avcs:syncInfo updateID="0">
   <avcs:pair
     syncRelationshipID="d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d"
     partnershipID="a0e4d0a7-3378-4f17-8af2-3f7de3345dc6"
     pairGroupID="ba8e57de-7f66-4102-ae4b-31b96c86f173">
    <avcs:remoteParentObjID>B0</avcs:RemoteParentObjID>
    <avcs:status>NEW</avcs:status>
   </avcs:pair>
  </avcs:syncInfo>
 </item>
 <item id="A3" parentID="A2" restricted="1">
  <dc:title>Chloe Dancer - Mother Love Bone.mp3</dc:title>
  <upnp:class>object.item.audioItem.musicTrack</upnp:class>
  <res protocolInfo="http-get:*:audio/mpeg:*" size="200000"
    avcs:syncAllowed="ALL" avcs:resModified="0">
   http://10.0.0.1/getcontent.asp?id=A3
  </res>
  <avcs:syncable/>
  <avcs:syncInfo updateID="0">
   <avcs:pair
     syncRelationshipID="d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d"
     partnershipID="a0e4d0a7-3378-4f17-8af2-3f7de3345dc6"
     pairGroupID="ba8e57de-7f66-4102-ae4b-31b96c86f173">
    <avcs:virtualRemoteParentObjID>A2</avcs:virtualRemoteParentObjID>
    <avcs:policy>
     <avcs:syncType>replace</avcs:syncType>
     <avcs:priorityPartnerID>1</avcs:priorityPartnerID>
    </avcs:policy>
    <avcs:status>NEW</avcs:status>
   </avcs:pair>
  </avcs:syncInfo>
 </item>
</DIDL-Lite>
", 3, 3)
```

When partner 2 processes the received change log, it updates its object B1 with *dc:title* and downloads a resource from partner 1 because the object in partner 1 is new to partner 2 and the object in partner 2 does not take precedence by the synchronization policy.

For object A2 in partner 1, partner 2 creates a new object under the container object B0 that is specified in the *remoteParentObjID* element of the *avcs:pair* in the change log as partner 2 does not have a corresponding pair object for object A2. While creating the new object, partner 2 accepts all properties and a resource from object A2. Let us assume that the newly created object has B2 as the value of *@objectID* property. This object B2 is now associated with object A2, which means that the value of the *remoteObjID* element is A2 as shown below:

```
<item id="B2" parentID="B0" restricted="1">
```

```
 <dc:title>My Music</dc:title>
 <upnp:class>object.container.album</upnp:class>
 <avcs:syncable/>
 <avcs:syncInfo updateID="0">
  <avcs:pair
    syncRelationshipID="d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d"
    partnershipID="a0e4d0a7-3378-4f17-8af2-3f7de3345dc6"
    pairGroupID="ba8e57de-7f66-4102-ae4b-31b96c86f173">
  <avcs:remoteObjID>A2</avcs:remoteObjID>
  <avcs:status>SYNC'ED</avcs:status>
  </avcs:pair>
 </avcs:syncInfo>
</item>
```

For object A3 in partner 1, partner 2 creates a new object under the newly-created container object (i.e. B2) during processing of object A2 as shown above. Let us assume that newly created object has B3 as the value of the *@objectID* property. The *virtualRemoteParentObjID>* element of the *avcs:pair* in the change log is replaced with the *remoteObjID* element when creating this new object. This object B3 is now associated with object A3, which means that the value of the *remoteObjID* element is A3 as shown below.

```
<item id="B3" parentID="B2" restricted="1">
 <dc:title>Chloe Dancer - Mother Love Bone.mp3</dc:title>
 <upnp:class>object.item.audioItem.musicTrack</upnp:class>
 <res protocolInfo="http-get:*:audio/mpeg:*"
   size="200000"avcs:syncAllowed="ALL" avcs:resModified="0">
  http://10.0.0.2/getcontent.asp?id=B3
 </res>
 <avcs:syncable/>
 <avcs:syncInfo updateID="0">
  <avcs:pair
    syncRelationshipID="d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d"
    partnershipID="a0e4d0a7-3378-4f17-8af2-3f7de3345dc6"
    pairGroupID="ba8e57de-7f66-4102-ae4b-31b96c86f173">
   <avcs:remoteObjID>A3</avcs:remoteObjID>
   <avcs:policy>
    <avcs:syncType>replace</avcs:syncType>
    <avcs:priorityPartnerID>1</avcs:priorityPartnerID>
   </avcs:policy>
   <avcs:status>SYNC'ED</avcs:status>
  </avcs:pair>
 </avcs:syncInfo>
</item>
```

After finishing updates for each object, partner 2 sends an event message to notify the status of each update operation. The following is an example of an event message that is sent to partner 1. Here, we assume that the first 2 events are sent within a single moderation time and the third event message will be sent during the next moderation time.

The partner 2 sends a GENA message during the first moderation time as shown below:

GENA Message: (Partner 2 to Partner 1)

```
<?xml version="1.0" encoding="utf-8">
<SyncStatusUpdate xmlns="urn:schemas-upnp-org:cs"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:schemas-upnp-org:cs
   http://www.upnp.org/schemas/cs/cs-v1-20070XXXX.xsd">
 <syncRelationship id="d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d">
  <status numberOfTotalObjects="3" numberOfCompletedObjects="2"
    numberOfFailedObjects="0">
   IN_PROGRESS
  </status>
  <partnership id="a0e4d0a7-3378-4f17-8af2-3f7de3345dc6">
   <status numberOfTotalObjects="3" numberOfCompletedObjects="2"
     numberOfFailedObjects="0">
    IN_PROGRESS
   </status>
```

```
    <pairGroup id="ba8e57de-7f66-4102-ae4b-31b96c86f173">
     <status numberOfTotalObjects="3" numberOfCompletedObjects="2"
       numberOfFailedObjects="0">
      IN_PROGRESS
     </status>
     <logEntry>
      <localObjectID>B1</localObjectID>
      <remoteObjectID>A1</remoteObjectID>
      <statusCode>001</statusCodes>
      <statusDescription>Succeeded completely</statusDescription>
     </logEntry>
     <logEntry>
      <localObjectID>B2</localObjectID>
      <remoteObjectID>A2</remoteObjectID>
      <statusCode>001</statusCodes>
      <statusDescription>Succeeded completely</statusDescription>
     </logEntry>
    </pairGroup>
   </partnership>
  </syncRelationship>
</SyncStatusUpdate>
```

The partner 2 sends a GENA event message for the second moderation time as shown below:

GENA Message: (Partner 2 to Partner 1)

```
<?xml version="1.0" encoding="utf-8">
<SyncStatusUpdate xmlns="urn:schemas-upnp-org:cs"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:schemas-upnp-org:cs
   http://www.upnp.org/schemas/cs/cs-v1-20070XXXX.xsd">
 <syncRelationship id="d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d">
  <status numberOfTotalObjects="3" numberOfCompletedObjects="3"
    numberOfFailedObjects="0">
   COMPLETED_ALL
  </status>
  <partnership id="a0e4d0a7-3378-4f17-8af2-3f7de3345dc6">
   <status numberOfTotalObjects="3" numberOfCompletedObjects="3"
     numberOfFailedObjects="0">
    COMPLETED_ALL
   </status>
   <pairGroup id="ba8e57de-7f66-4102-ae4b-31b96c86f173">
    <status numberOfTotalObjects="3" numberOfCompletedObjects="3"
      numberOfFailedObjects="0">
     COMPLETED_ALL
    </status>
    <logEntry>
     <localObjectID>B3</localObjectID>
     <remoteObjectID>A3</remoteObjectID>
     <statusCode>001</statusCodes>
     <statusDescription>Succeeded completely</statusDescription>
    </logEntry>
   </pairGroup>
  </partnership>
 </syncRelationship>
</SyncStatusUpdate>
```

After updating the CDS of Parter 2, it explictly invokes the *ResetChangeLog()* action to Partner 1 in order to inform that individual object in change log is successfully synchronized.

Request: (Partner 1 to Partner 2)

```
ResetChangeLog("d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d", "
<ResetObjectList xmlns="urn:schemas-upnp-org:cs"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:schemas-upnp-org:cs
   http://www.upnp.org/schemas/cs/cs-v1-20070XXXX.xsd">
 <object id="A1" remoteObjID="B1" updateID="0"/>
 <object id="A2" remoteObjID="B2" updateID="0"/>
 <object id="A3" remoteObjID="B3" updateID="0"/>
 </ResetObjectList>
");
```

Response: (Partner 2 to Partner 1)

ResetChangeLog();

When the partner 1 receives the *ResetChangeLog()* action, it can now change the value of the *avcs:status* property of the object A1, A2 and A3 to "SYNC'ED", repectively. In addition, the *avcs:remoteParentObjID* property of the object A2 and the *avcs:virtualRemoteParentObjID* property of the object A3 are replaced with the *avcs:remoteObjID* property since the Partner 2 notifies those objects are successfully synchronized.

After completing the synchronization operation, two CDSs MUST show the following hierarchies.

CDS hierarchy of the partner 1:

```
<?xml version="1.0" encoding="UTF-8"?>
<DIDL-Lite xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns="urn:schemas-upnp-org:metadata-1-0/DIDL-Lite/"
  xmlns:upnp="urn:schemas-upnp-org:metadata-1-0/upnp/"
  xmlns:avcs="urn:schemas-upnp-org:cs:avcs"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:schemas-upnp-org:metadata-1-0/DIDL-Lite/
  http://upnp.org/standardizeddcps/schemas/avwc/didl-lite/2.00
  urn:schemas-upnp-org:metadata-1-0/upnp/
  http://upnp.org/standardizeddcps/schemas/avwc/upnp/2.00
  urn:schemas-upnp-org:cs:avcs
  http://www.upnp.org/cs/avcs-v1-2007xxxx.xsd">
<item id="A1" parentID="A0" restricted="1">
  <dc:title>Would - Alice In Chains.wma</dc:title>
  <upnp:class>object.item.audioItem.musicTrack</upnp:class>
  <res protocolInfo="http-get:*:audio/x-ms-wma:*" size="90000"
    avcs:syncAllowed="ALL" avcs:resModified="0">
   http://10.0.0.1/getcontent.asp?id=A1
  </res>
  <avcs:syncable/>
  <avcs:syncInfo updateID="0">
   <avcs:pair
     syncRelationshipID="d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d"
     partnershipID="a0e4d0a7-3378-4f17-8af2-3f7de3345dc6"
     pairGroupID="ba8e57de-7f66-4102-ae4b-31b96c86f173">
    <avcs:remoteObjID>B1</avcs:remoteObjID>
    <avcs:status>SYNC'ED</avcs:status>
   </avcs:pair>
  </avcs:syncInfo>
 </item>
 <item id="A2" parentID="A0" restricted="1">
  <dc:title>My Music</dc:title>
  <upnp:class>object.container.album</upnp:class>
  <avcs:syncable/>
  <avcs:syncInfo updateID="0">
   <avcs:pair
     syncRelationshipID="d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d"
     partnershipID="a0e4d0a7-3378-4f17-8af2-3f7de3345dc6"
     pairGroupID="ba8e57de-7f66-4102-ae4b-31b96c86f173">
    <avcs:remoteObjID>B2</avcs:remoteObjID>
    <avcs:status>SYNC'ED</avcs:status>
   </avcs:pair>
  </avcs:syncInfo>
 </item>
 <item id="A3" parentID="A2" restricted="1">
  <dc:title>Chloe Dancer - Mother Love Bone.mp3</dc:title>
  <upnp:class>object.item.audioItem.musicTrack</upnp:class>
  <res protocolInfo="http-get:*:audio/mpeg:*" size="200000"
```

```
    avcs:syncAllowed="ALL" avcs:resModified="0">
   http://10.0.0.1/getcontent.asp?id=A3
  </res>
  <avcs:syncable/>
  <avcs:syncInfo updateID="0">
   <avcs:pair
     syncRelationshipID="d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d"
     partnershipID="a0e4d0a7-3378-4f17-8af2-3f7de3345dc6"
     pairGroupID="ba8e57de-7f66-4102-ae4b-31b96c86f173">
    <avcs:remoteObjID>B3</avcs:remoteObjID>
    <avcs:policy>
     <avcs:syncType>replace</avcs:syncType>
     <avcs:priorityPartnerID>1</avcs:priorityPartnerID>
    </avcs:policy>
    <avcs:status>SYNC'ED</avcs:status>
   </avcs:pair>
  </avcs:syncInfo>
 </item>
 <item id="A4" parentID="A0" restricted="1">
  <dc:title>Wonder - Tell Me</dc:title>
  <upnp:class>object.item.audioItem.musicTrack</upnp:class>
  <res protocolInfo="http-get:*:audio/mpeg:*" size="500000"
    avcs:syncAllowed="ALL" avcs:resModified="0">
   http://10.0.0.1/getcontent.asp?id=4
  </res>
  <avcs:syncable/>
  <avcs:syncInfo updateID="0">
   <avcs:pair
     syncRelationshipID="d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d"
     partnershipID="a0e4d0a7-3378-4f17-8af2-3f7de3345dc6"
     pairGroupID="ba8e57de-7f66-4102-ae4b-31b96c86f173">
    <avcs:remoteObjID>B4</avcs:remoteObjID>
    <avcs:status>SYNC'ED</avcs:status>
   </avcs:pair>
  </avcs:syncInfo>
 </item>
</DIDL-Lite>
```

CDS hierarchy of the partner 2:

```
<?xml version="1.0" encoding="UTF-8"?>
<DIDL-Lite xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns="urn:schemas-upnp-org:metadata-1-0/DIDL-Lite/"
  xmlns:upnp="urn:schemas-upnp-org:metadata-1-0/upnp/"
  xmlns:avcs="urn:schemas-upnp-org:cs:avcs"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:schemas-upnp-org:metadata-1-0/DIDL-Lite/
  http://upnp.org/standardizeddcps/schemas/avwc/didl-lite/2.00
  urn:schemas-upnp-org:metadata-1-0/upnp/
  http://upnp.org/standardizeddcps/schemas/avwc/upnp/2.00
  urn:schemas-upnp-org:cs:avcs
  http://www.upnp.org/schemas/cs/avcs-v1-2007xxxx.xsd">
 <item id="B1" parentID="B0" restricted="1">
  <dc:title>Would - Alice In Chains.wma</dc:title>
  <upnp:class>object.item.audioItem.musicTrack</upnp:class>
  <res protocolInfo="http-get:*:audio/x-ms-wma:*" size="90000"
    avcs:syncAllowed="ALL" avcs:resModified="0">
   http://10.0.0.2/getcontent.asp?id=B1
  </res>
  <avcs:syncable/>
  <avcs:syncInfo updateID="0">
   <avcs:pair
     syncRelationshipID="d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d"
     partnershipID="a0e4d0a7-3378-4f17-8af2-3f7de3345dc6"
     pairGroupID="ba8e57de-7f66-4102-ae4b-31b96c86f173">
    <avcs:remoteObjID>A1</avcs:remoteObjID>
    <avcs:status>SYNC'ED</avcs:status>
   </avcs:pair>
  </avcs:syncInfo>
 </item>
 <item id="B2" parentID="B0" restricted="1">
  <dc:title>My Music</dc:title>
  <upnp:class>object.container.album</upnp:class>
  <avcs:syncable/>
  <avcs:syncInfo updateID="0">
   <avcs:pair
     syncRelationshipID="d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d"
     partnershipID="a0e4d0a7-3378-4f17-8af2-3f7de3345dc6"
     pairGroupID="ba8e57de-7f66-4102-ae4b-31b96c86f173">
    <avcs:remoteObjID>A2</avcs:remoteObjID>
    <avcs:status>SYNC'ED</avcs:status>
   </avcs:pair>
  </avcs:syncInfo>
 </item>
 <item id="B3" parentID="B2" restricted="1">
  <dc:title>Chloe Dancer - Mother Love Bone.mp3</dc:title>
  <upnp:class>object.item.audioItem.musicTrack</upnp:class>
  <res protocolInfo="http-get:*:audio/mpeg:*" size="200000"
    avcs:syncAllowed="ALL" avcs:resModified="0">
   http://10.0.0.2/getcontent.asp?id=B3
  </res>
  <avcs:syncable/>
  <avcs:syncInfo updateID="0">
   <avcs:pair
     syncRelationshipID="d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d"
     partnershipID="a0e4d0a7-3378-4f17-8af2-3f7de3345dc6"
     pairGroupID="ba8e57de-7f66-4102-ae4b-31b96c86f173">
    <avcs:remoteObjID>A3</avcs:remoteObjID>
    <avcs:policy>
     <avcs:syncType>replace</avcs:syncType>
     <avcs:priorityPartnerID>1</avcs:priorityPartnerID>
    </avcs:policy>
    <avcs:status>SYNC'ED</avcs:status>
   </avcs:pair>
  </avcs:syncInfo>
 </item>
 <item id="B4" parentID="B0" restricted="1">
  <dc:title>Wonder - Tell Me</dc:title>
  <upnp:class>object.item.audioItem.musicTrack</upnp:class>
  <res protocolInfo="http-get:*:audio/mpeg:*" size="500000"
    avcs:syncAllowed="ALL" avcs:resModified="0">
   http://10.0.0.2/getcontent.asp?id=4
  </res>
  <avcs:syncable/>
  <avcs:syncInfo updateID="0">
   <avcs:pair
     syncRelationshipID="d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d"
     partnershipID="a0e4d0a7-3378-4f17-8af2-3f7de3345dc6"
     pairGroupID="ba8e57de-7f66-4102-ae4b-31b96c86f173">
    <avcs:remoteObjID>A4</avcs:remoteObjID>
    <avcs:status>SYNC'ED</avcs:status>
   </avcs:pair>
  </avcs:syncInfo>
 </item>
</DIDL-Lite>
```

2.10.2.5 Next Synchronization after Changing Objects

2.10.2.6

Objects changed:

If an object is changed since the last synchronization, the DIDL-Lite object keeps track of which property is changed. In this example, the dc:title property of object A1 in partner 1 is changed and object A3 is deleted. To find out the changed object, the embedded control point in the partner device uses the GetChangeLog( ) action. To trigger synchronization operation, the procedure as described in Clause above needs to be followed.

Getting Change Log:

The partner 1 gets no changed CDS objects as there is no change in objects on the partner 2 in this example.

However, when the partner 2 calls the GetChangeLog() action on the partner 1, it will get the following changed CDS objects..

Request: (Partner 2 to Partner 1)

```
GetChangeLog("d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d", 0, 0);
```

Response: (Partner 1 to Partner 2)

```
GetChangeLog("
<?xml version="1.0" encoding="UTF-8"?>
<DIDL-Lite xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns="urn:schemas-upnp-org:metadata-1-0/DIDL-Lite/"
  xmlns:upnp="urn:schemas-upnp-org:metadata-1-0/upnp/"
  xmlns:avcs="urn:schemas-upnp-org:cs:avcs"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:schemas-upnp-org:metadata-1-0/DIDL-Lite/
  http://upnp.org/standardizeddcps/schemas/avwc/didl-lite/2.00
  urn:schemas-upnp-org:metadata-1-0/upnp/
  http://upnp.org/standardizeddcps/schemas/avwc/upnp/2.00
  urn:schemas-upnp-org:cs:avcs
  http://www.upnp.org/schemas/cs/avcs-v1-2007xxxx.xsd">
 <item id="A1" parentID="0" restricted="1">
  <dc:title>Alice In Chains(Live)</dc:title>
  <upnp:class>object.item.audioItem.musicTrack</upnp:class>
  <res protocolInfo="http-get:*:audio/x-ms-wma:*"
    size="90000"avcs:syncAllowed="ALL" avcs:resModified="0">
   http://10.0.0.1/getcontent.asp?id=A1
  </res>
  <avcs:syncable/>
  <avcs:syncInfo updateID="1">
   <avcs:pair
     syncRelationshipID="d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d"
     partnershipID="a0e4d0a7-3378-4f17-8af2-3f7de3345dc6"
     pairGroupID="ba8e57de-7f66-4102-ae4b-31b96c86f173">
    <avcs:remoteObjID>A1</avcs:remoteObjID>
    <avcs:status>MODIFIED</avcs:status>
   </avcs:pair>
  </avcs:syncInfo>
 </item>
 <item id="A3" parentID="A2">
  <avcs:syncInfo updateID="1">
   <avcs:pair
     syncRelationshipID="d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d"
     partnershipID="a0e4d0a7-3378-4f17-8af2-3f7de3345dc6"
     pairGroupID="a8e57de-7f66-4102-ae4b-31b96c86f173">
    <avcs:remoteObjID>B3</avcs:remoteObjID>
    <avcs:status>DELETED</avcs:status>
   </avcs:pair>
  </avcs:syncInfo>
 </item>
</DIDL-Lite>
", 2, 2)
```

Therefore, partner 2 updates the <dc:title> property for object B1 and destroys object B3 based on the rules defined by the synchronization policy.

To confirm that the objects in the received change log are successfully synchronized, the Partner 2 invokes the GetChangeLog() action with object A1 and A3 information.

Request: (Partner 2 to Partner 1)

```
ResetChangeLog("d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d", "
<ResetObjectList xmlns="urn:schemas-upnp-org:cs"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:schemas-upnp-org:cs
   http://www.upnp.org/schemas/cs/cs-v1-20070XXXX.xsd">
 <object id="A1" remoteObjID="B1" updateID="0"/>
 <object id="A3" remoteObjID="B3" updateID="0"/>
</ResetObjectList>
");
```

Response: (Partner 1 to Partner 2)

```
ResetChangeLog("");
```

When the Partner 1 receives the *GetChangeLog()* action above, it change the value of the *avcs:status* of the object A1 and A3 to "SYNC'ED" repectively.

2.10.2.7 Modifications of a Synchronization Data Structure

To explain how to update a synchronization data structure, assume that there are two synchronization partners in a synchronization relationship as described in the XML document below.

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentSync xmlns="urn:schemas-upnp-org:cs"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:schemas-upnp-org:cs
   http://www.upnp.org/schemas/cs/cs-v1-20070XXXX.xsd">
 <syncRelationship id="d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d" active="1">
  <title>Sync between My iPod and Home Media Server</title>
  <partnership id="a0e4d0a7-3378-4f17-8af2-3f7de3345dc6" active="1"
    updateID="0">
   <partner id="1">
    <deviceUDN>343bd2a2-189b-40c0-8eb5-ea91ea730402</deviceUDN>
    <serviceID>service_ID_A</serviceID>
   </partner>
   <partner id="2">
    <deviceUDN>05de2732-5df5-4c48-922b-12f73473f0e9</deviceUDN>
    <serviceID>service_ID_B</serviceID>
   </partner>
   <policy>
    <syncType>merge</syncType>
    <priorityPartnerID>1</priorityPartnerID>
   </policy>
   <pairGroup id="ba8e57de-7f66-4102-ae4b-31b96c86f173" active="1"/>
  </partnership>
 </syncRelationship>
</ContentSync>
```

The following example demonstrates how to update the partnership in the synchronization relationship by the *ModifySyncData()* action:

Request: (Control Point to Partner 1)

```
ModifySyncData("", "a0e4d0a7-3378-4f17-8af2-3f7de3345dc6",
<partnership updateID="0" xmlns="urn:schemas-upnp-org:cs"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:schemas-upnp-org:cs
   http://www.upnp.org/schemas/cs/cs-v1-20070XXXX.xsd">
 <policy>
  <syncType>merge</syncType>
  <precedence>2</precedence>
 </policy>
</partnership>
");
```

The partner1 that receives the *ModifySyncData()* action invokes the *ModifySyncData()* action on the partner 2 to maintain identical information for the synchronization relationship on both partner devices.

Request: (Partner 1 to Partner 2)

```
ModifySyncData("343bd2a2-189b-40c0-8eb5-ea91ea730402", "a0e4d0a7-3378-4f17-8af2-
3f7de3345dc6",
<partnership updateID="0" xmlns="urn:schemas-upnp-org:cs"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:schemas-upnp-org:cs
   http://www.upnp.org/schemas/cs/cs-v1-20070XXXX.xsd">
 <policy>
  <syncType>merge</syncType>
  <precedence>2</precedence>
 </policy>
</partnership>
");
```

Response: (Partner 2 to Partner 1)

```
ModifySyncData();
```

After receiving the second action response, the partner 1 responds to the first *ModifySyncData()* action.

Response: (Partner 1 to Control Point)

```
ModifySyncData();
```

When the update is successfully processed, the updated synchronization relationship MUST be shown as below. (*partnership@updateID* is increased by 1.)

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentSync xmlns="urn:schemas-upnp-org:cs"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:schemas-upnp-org:cs
   http://www.upnp.org/schemas/cs/cs-v1-20070XXXX.xsd">
 <syncRelationship id="d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d" active="1">
  <title>Sync between My iPod and Home Media Server</title>
  <partnership id="a0e4d0a7-3378-4f17-8af2-3f7de3345dc6" active="1"
    updateID="1">
   <partner id="1">
    <deviceUDN>343bd2a2-189b-40c0-8eb5-ea91ea730402</deviceUDN>
    <serviceID>service_ID_A</serviceID>
   </partner>
   <partner id="2">
    <deviceUDN>05de2732-5df5-4c48-922b-12f73473f0e9</deviceUDN>
    <serviceID>service_ID_B</serviceID>
   </partner>
   <policy>
    <syncType>merge</syncType>
    <priorityPartnerID>2</priorityPartnerID>
   </policy>
   <pairGroup id="ba8e57de-7f66-4102-ae4b-31b96c86f173" active="1"/>
  </partnership>
 </syncRelationship>
</ContentSync>
```

2.10.3 Synchronization of a Reference Object

To explain how to synchronize a reference object, let us consider the following logical structure of two separate CDSs as shown below. Each of the CDS exposes physical directory structure like a PC file system. The content in the partner 1 includes two items each of which is a music item. One of the music items is a reference item. The content in the partner 2 does not include anything.

[Partner 1CDS hierarchy]

- Name=“Content”, ID=”0”
  - Name=“Would - Alice In Chains.wma”, ID=”A1”, Size=“90000”

- Name=”Music Playlist”, ID=”A2”
  - Name=“Would - Alice In Chains.wma”, ID=”A3”, refID=”A1”, Artist=”Mary” Size=“200000”

[Partner 2CDS hierarchy]

- Name=“My Multimedia Content”, ID=”0”

Figure 9 shows the visual Hierarchy of two CDSs. Object A3 refers to object A1 and has an additional property such as the *upnp:artist* property.

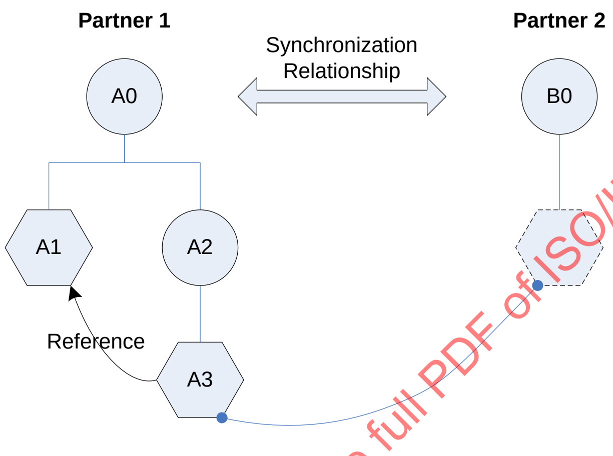

Figure 9 — Synchronization Relationship between two CDSs

Creating a Pair of a Reference Object:

Firstly, let us assume that the synchronization relationship is setup. To synchronize a reference object, a control point should make a synchronization pair and associate the pair with the synchronization relationship. There is no difference in making an pair for a regular object such as audio or video item and a reference object. Therefore, the control point invokes the *AddSyncPair()* as shown below. However, this action does not invoke subsequent *AddSyncPair()* action on the partner 2 device since there is no corresponding remote object in the partner 2.

Request: (Control Point to Partner 1)

```
AddSyncPair("", "A3", "
<avcs:pair xmlns:avcs="urn:schemas-upnp-org:cs:avcs"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:schemas-upnp-org:cs:avcs
  http://www.upnp.org/schemas/cs/avcs-v1-2007xxxx.xsd"
  syncRelationshipID="d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d"
  partnershipID="a0e4d0a7-3378-4f17-8af2-3f7de3345dc6"
  pairGroupID="ba8e57de-7f66-4102-ae4b-31b96c86f173">
 <avcs:remoteParentObjID>B0</avcs:remoteParentObjID>
 <avcs:policy>
  <avcs:syncType>replace</avcs:syncType>
  <avcs:priorityPartnerID>1</avcs:priorityPartnerID>
 </avcs:policy>
</avcs:pair>
");
```

Response: (Partner 1 to Control Point)

```
AddSyncPair();
```

Once the pair information is created, the *DIDL-Lite XML document* for object A3 is shown as below:

```
<?xml version="1.0" encoding="UTF-8"?>
<DIDL-Lite xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns="urn:schemas-upnp-org:metadata-1-0/DIDL-Lite/"
  xmlns:upnp="urn:schemas-upnp-org:metadata-1-0/upnp/"
  xmlns:avcs="urn:schemas-upnp-org:cs:avcs"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:schemas-upnp-org:metadata-1-0/DIDL-Lite/
  http://upnp.org/standardizeddcps/schemas/avwc/didl-lite/2.00
  urn:schemas-upnp-org:metadata-1-0/upnp/
  http://upnp.org/standardizeddcps/schemas/avwc/upnp/2.00
  urn:schemas-upnp-org:cs:avcs
  http://www.upnp.org/schemas/cs/avcs-v1-2007xxxx.xsd">
 <item id="A3" parentID="A2" restricted="1">
  <dc:title>Alice In Chains</dc:title>
  <upnp:artist>Mary</upnp:artist>
  <upnp:class>object.item.audioItem.musicTrack</upnp:class>
  <res protocolInfo="http-get:*:audio/x-ms-wma:*"
    size="90000" avcs:syncAllowed="ALL" avcs:resModified="0">
   http://10.0.0.2/getcontent.asp?id=1
  </res>
  <avcs:syncable/>
  <avcs:syncInfo>
   <avcs:pair
     syncRelationshipID="d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d"
     partnershipID="a0e4d0a7-3378-4f17-8af2-3f7de3345dc6"
     pairGroupID="ba8e57de-7f66-4102-ae4b-31b96c86f173">
    <avcs:remoteParentObjID>0</avcs:remoteParentObjID>
    <avcs:status>NEW</avcs:status>
   </avcs:pair>
  </avcs:syncInfo>
 </item>
</DIDL-Lite>
```

Getting Change Log:

Once the partners are triggered to start the synchronization operation, each partner gets the synchronization object by invoking *GetChangeLog()* actions.

In this example, the partner 1 will not get anything. However, the partner 2 will get the DIDL-Lite XML document for object A3 as shown below.

Request: (Partner 2 to Partner 1)

```
GetChangeLog("d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d", 0, 0);
```

Response: (Partner 1 to Partner 2)

```
GetChangeLog(
<?xml version="1.0" encoding="UTF-8"?>
<DIDL-Lite xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns="urn:schemas-upnp-org:metadata-1-0/DIDL-Lite/"
  xmlns:upnp="urn:schemas-upnp-org:metadata-1-0/upnp/"
  xmlns:avcs="urn:schemas-upnp-org:cs:avcs"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:schemas-upnp-org:metadata-1-0/DIDL-Lite/
  http://upnp.org/standardizeddcps/schemas/avwc/didl-lite/2.00
  urn:schemas-upnp-org:metadata-1-0/upnp/
  http://upnp.org/standardizeddcps/schemas/avwc/upnp/2.00
  urn:schemas-upnp-org:cs:avcs
  http://www.upnp.org/schemas/cs/avcs-v1-2007xxxx.xsd">
 <item id="A3" parentID="A2" restricted="1">
  <dc:title>Alice In Chains</dc:title>
  <upnp:artist>Mary</upnp:artist>
  <upnp:class>object.item.audioItem.musicTrack</upnp:class>
  <res protocolInfo="http-get:*:audio/x-ms-wma:*"
    size="90000"avcs:syncAllowed="ALL" avcs:resModified="0">
   http://10.0.0.2/getcontent.asp?id=1
  </res>
  <avcs:syncable/>
  <avcs:syncInfo updateID="0">
   <avcs:pair
     syncRelationshipID="d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d"
     partnershipID="a0e4d0a7-3378-4f17-8af2-3f7de3345dc6"
     pairGroupID="ba8e57de-7f66-4102-ae4b-31b96c86f173">
   <avcs:remoteParentObjID>B0</avcs:remoteParentObjID>
   <avcs:status>NEW</avcs:status>
   </avcs:pair>
  </avcs:syncInfo>
 </item>
</DIDL-Lite>
", 1, 1)
```

Then, the partner 2 creates the new object that is paired with object A3. After creating the new object, the partner 2 sends an event message to notify the status of the operation. The following is an example of the event message for object B1 that is newly created in the partner 2.

GENA Message: (Partner 2 to Partner 1)

```
<?xml version="1.0" encoding="utf-8">
<SyncStatusUpdate xmlns="urn:schemas-upnp-org:cs"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:schemas-upnp-org:cs
   http://www.upnp.org/schemas/cs/cs-v1-20070XXXX.xsd">
 <syncRelationship id="d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d">
  <status numberOfTotalObjects="1" numberOfCompletedObjects="1"
    numberOfFailedObjects="0">
   COMPLETED_ALL
  </status>
  <partnership id="a0e4d0a7-3378-4f17-8af2-3f7de3345dc6">
   <status numberOfTotalObjects="1" numberOfCompletedObjects="1"
     numberOfFailedObjects="0">
    COMPLETED_ALL
   </status>
   <pairGroup id="pr001">
    <status numberOfTotalObjects="1" numberOfCompletedObjects="1"
      numberOfFailedObjects="0">
     COMPLETED_ALL
    </status>
    <logEntry>
     <localObjectID>B1</localObjectID>
     <remoteObjectID>A3</remoteObjectID>
     <statusCode>001</statusCodes>
     <statusDescription>Succeeded completely</statusDescription>
    </logEntry>
   </pairGroup>
  </partnership>
 </syncRelationship>
</SyncStatusUpdate>
```

To confirm that the object A1 in the received change log are successfully synchronized, the Partner 2 invokes the GetChangeLog() action.

Request: (Partner 2 to Partner 1)

```
ResetChangeLog("d8c9fa13-d79b-4a0c-999b-6ae2ff91a46d", "
<ResetObjectList xmlns="urn:schemas-upnp-org:cs"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:schemas-upnp-org:cs
   http://www.upnp.org/schemas/cs/cs-v1-20070XXXX.xsd">
 <object id="A3" remoteObjID="B1" updateID="0"/>
</ResetObjectList>
```