Coordinator API Specification

Version 2.4 2 March 2016

Coordinator API Specification Version 2.4

Notice:

As of the date of publication, this document is a release candidate specification subject to DECE Member review and final adoption by vote of the Management Committee of DECE in accordance with the DECE LLC Operating Agreement. Unless there is notice to the contrary, this specification will become an adopted “Ecosystem Specification” on 17 April 2016.

THIS DOCUMENT IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE. Digital Entertainment Content Ecosystem (DECE) LLC (“DECE”) and its members disclaim all liability, including liability for infringement of any proprietary rights, relating to use of information in this specification. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted herein.

Implementation of this specification requires a license from DECE. This document is subject to change under applicable license provisions.

Contact Information:

Licensing inquiries and requests should be addressed to us at: http://www.uvvu.com/uv-for-business

Coordinator API Specification Version 2.4

Contents

1 Introduction and Overview

This specification details the API protocols and message structures of the Coordinator. The Coordinator provides an in-network architecture component, which houses shared resources amongst the various Roles specified in [DSystem]. This specification also covers the Web Portal, an independent HTML-based user interface to Coordinator functionality.

1.1 Scope

The APIs specified here are written in terms of Roles, such as DSPs, LASPs, Retailers, Content Providers, Portals, and customer support. The Coordinator Customer Support Role is part of the broader definition of Coordinator, and therefore APIs are designed to model behavior rather than to specify implementation. Each instantiation of a Role, such as a particular Retailer or DSP, is called a Node.

1.2 Document Organization

This document is organized as follows:

Introduction and Overview—Provides background, scope and conventions

Communications Security – Provides Coordinator-specific security requirements beyond what is already specified in [DSecMech]

Resource-Oriented API – Introduces the Representational State Transfer (REST) model, and its application to the Coordinator interfaces

DECE Coordinator API Overview – Briefly introduces the Coordinator interfaces

Policies – Specifies the Policy data model and related APIs

Assets, Metadata, Asset Mapping and Bundles – Specifies the Assets and Asset Metadata data model and related APIs

Rights – Specifies the RightsToken data model and related APIs

Legacy Devices – Specifies the Legacy Device data model and associated APIs

Streams – Specifies the Stream and Stream Lease data model and associated APIs

User Delegation – Specifies the delegation model between Nodes and Users

Node to Account Delegation – Specifies the various types of delegations and their management

Accounts – Specifies the Account data model and associated APIs

Users – Specifies the User data model and associated APIs

Node Management – Specifies the Node data model and associated APIs

Discrete Media – Specifies the Discrete Media Token data model and associated APIs

Other – Specifies other various structures, in particular resource status and its management API

1.3 Document Conventions

The following terms are used to specify conformance elements of this specification. These are adopted from the ISO/IEC Directives, Part 2, Annex H [ISO-P2H].

> The terms SHALL and SHALL NOT indicate requirements strictly to be followed in order to conform to the document and from which no deviation is permitted.
>
> The terms SHOULD and SHOULD NOT indicate that among several possibilities one is recommended as particularly suitable, without mentioning or excluding others, or that a certain course of action is preferred but not necessarily required, or that (in the negative form) a certain possibility or course of action is deprecated but not prohibited.
>
> The terms MAY and NEED NOT indicate a course of action permissible within the limits of the document.

Terms defined to have a specific meaning within this specification will be capitalized, for example, “User,” and should be interpreted with their general meaning if not capitalized. Normative key words are written in all caps, for example, “SHALL.”

1.3.1 XML Conventions

This document uses tables to define XML structures. These tables may combine multiple elements and attributes in a single table. The tables do not align precisely with the XML schema; but they should not conflict with the schema. In any case where the XSD and annotations within this specification differ, the Coordinator Schema XSD [DCSchema] SHALL be considered authoritative.

Most elements and attributes defined in [DCSchema] have practical maximum length restrictions. Such restrictions are defined in Appendix G.

1.3.1.1 Naming Conventions

This section describes naming conventions for DECE XML attributes, element and other named entities. The conventions are as follows:

- Names use initial caps, as in Names.
- Elements begin with a capital letter, and use camel-case, as in InitialCapitalLetters.
- Attributes begin with a capital letter, as in Attribute.
- XML structures are formatted using a monospace font, for example: `RightsToken`.
- The names of both simple and complex types are followed with the suffix"`-type`."

1.3.1.2 Element Table Overview

The element-definition tables, found throughout the document, contain the following headings:

Element: the name of the element.

Attribute: the name of the attribute.

Definition: a descriptive definition, which may define conditions of use or other constraints.

Value: the format of the attribute or element. The value may be an XML type (for example `string`) or a reference to another element table (for example, "see Table 999") or section in the document. Annotations for limits or enumerations may be included.

Cardinality: specifies the cardinality of the element, for example, 0...n. The default cardinality value is 1.

The first row in the table names the element being defined. It is followed by the element's attributes, and then by child elements. All child elements are included. Simple child elements may be fully defined in the table.

DECE defined data types and values are shown in monospace font, as in `urn:dece:role:retailer:customersupport`.

1.3.1.3 Parameter Naming Convention

There are numerous parameters in the DECE architecture that are referred to across documents. These may be DECE variables, which are specified in [DSystem], while others may be defined in other

publications. All of these variables use the same naming convention, however. They are always rendered in uppercase:

[documentref]_VARIABLE

where:

[documentref] is a reference to the publication where the variable is defined.

If the variable does not include a [documentref], it is defined in [DSystem]

1.3.2 XML Namespaces

Conventional XML namespace prefixes are used throughout the listings in this specification to stand for their respective namespaces as follows, whether or not a namespace declaration is present in the example:

Prefix	XML Namespace	Description
dece:	http://www.decellc.org/schema/2015/03/coordinator	This is the DECE Coordinator Schema namespace, as defined in the schema [DCSchema].
md:	XML schema namespace as defined in [DMeta] for [XSD-META-CM]	This schema defines common metadata, which is the basis for DECE metadata.
xenc:	http://www.w3.org/2001/04/xmlenc#	This is the W3C XML Encryption namespace.
xs:	http://www.w3.org/2001/XMLSchema	This is the W3C XML schema namespace [XML].

Table 1: XML Namespaces

1.4 Normative References

The following table contains the complete list of normative DECE and external publications.

Reference	Description
[DCMeta]	Common Metadata Specification and Schema
[DCSchema]	Coordinator API XML Schema
[DDiscreteMedia]	Discrete Media Specification
[DGeo]	Geography Policies Specification
[DMeta]	Content Metadata Specification
[DMetaCR]	Common Metadata Content Ratings Specification

Reference	Description
[DPublisher]	Content Publishing Specification
[DSecMech]	Message Security Mechanisms Specification
[DSystem]	System Specification
[DNSSEC]	R. Arends, et al, *DNS Security Introduction and Requirements*, IETF, March 2005. Available at *http://www.ietf.org/rfc/rfc4033.txt* R. Arends, et al, *Resource Records for the DNS Security Extensions*, IETF, March 2005. Available at *http://www.ietf.org/rfc/rfc4034.txt* R. Arends, et al, *Protocol Modifications for the DNS Security Extensions*, IETF March 2005. Available at *http://www.ietf.org/rfc/rfc4035.txt*
[HTML4]	D Raggett , et al, HTML 4.01 Specification, W3C, December 1999. Avaiable at http://www.w3.org/TR/html401/
[ISO-P2H]	ISO/IEC Directives, Part 2, Annex H http://www.iso.org
[ISO3166-1]	Codes for the representation of names of countries and their subdivisions— Part 1: Country codes, 2007
[ISO3166-2]	Codes for the representation of names of countries and their subdivisions— Part 2: Country subdivision codes
[ISO8601]	ISO 8601:2000 Second Edition, Representation of dates and times, second edition, 2000-12-15
[RFC2045]	N. Freed, et al, *Multipurpose Internet Mail Extensions. (MIME) Part One: Format of Internet Message Bodies*, November 1996. Available at *http://www.ietf.org/rfc/rfc2045.txt*
[RFC2616]	Hypertext Transfer Protocol —HTTP/1.1
[RFC3986]	Uniform Resource Identifier (URI): Generic Syntax
[RFC4346]	The Transport Layer Security (TLS) Protocol Version 1.1
[RFC4646]	Philips, A, et al, RFC 4646, *Tags for Identifying Languages*, IETF, September 2006. Available at *http://www.ietf.org/rfc/rfc4646.txt*
[RFC4647]	Philips, A, et al, RFC 4647, *Matching of Language Tags*, IETF, September 2006. Available at *http://www.ietf.org/rfc/rfc4647.txt*
[RFC5246]	The Transport Layer Security (TLS) Protocol Version 1.2
[Unicode]	J. D. Allen, et al, The Unicode Standard Version 6.0 – Core Specification (ISO/IEC 10646:2010), The Unicode Consortium, October 2010. Avaiable at *http://www.unicode.org/versions/Unicode6.0.0/*
[XML]	"XML Schema Part 1: Structures", Henry S. Thompson, David Beech, Murray Maloney, Noah Mendelsohn, W3C Recommendation 28 October 2004, http://www.w3.org/TR/xmlschema-1/ "XML Schema Part 2: Datatypes", Paul Biron and Ashok Malhotra, W3C Recommendation 28 October 2004, http://www.w3.org/TR/xmlschema-2/
[XMLENC]	XML Encryption Syntax and Processing – W3C Recommendation *http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/*
[XPATH]	XML Path Language (XPath) 2.0 (Second Edition) – W3C Recommendation *http://www.w3.org/TR/xpath20/*
[XPATHFN]	XQuery 1.0 and XPath 2.0 Functions and Operators (Second Edition) – W3C Recommendation, 14 December 2010, *http://www.w3.org/TR/xpath-functions/*

1.5 Informative References

Reference	Description
[UCheckout]	H. Nielsen, et al, Detecting the Lost Update Problem Using Unreserved Checkout, W3C. May 1999. http://www.w3.org/1999/04/Editing/
[SAML]	OASIS, "Security Assertion Markup Language (SAML) v2.0" See http://docs.oasis-open.org/security/saml/v2.0/saml-2.0-os.zip
[CMCRSchema]	See http://www.movielabs.com/md/ratings/

1.6 General Notes

- All times are in Coordinated Universal Time (UTC) unless otherwise stated.
- An unspecified cardinality (“Card.”) is always 1.
- Character encoding support for XML instance documents SHALL be UTF-8 (see Character encoding for details)

1.7 Glossary of Terms

The following terms have specific meanings in the context of this specification. Additional terms employed in other specifications, agreements or guidelines are defined there. The definitions of many terms have been consolidated in [DSystem].

Affiliated Node: A Node is said to be an Affiliated Node if the Nodes share a common parent Organization. For example, a Retailer and LASP Node within the same Organization are Affiliated Nodes. See section 2.3.2.1.

API Client: An authorized client of one or more of the APIs defined in this specification. For example, a Node or Licensed Application.

Delegation Security Token: A Security Token, as defined in [DSecMech], used by a Node to demonstrate authorization has been granted to it in order to performed specific operations on Accounts, Users, Devices, or Lockers, based on established User and Account policies.

Geography Policy: Publication that details specific operational concerns, constraints, or guidance when providing services to a User. Typically, these include guardianship requirements, privacy requirements, etc.

Policy: A resource, defined by a policy class, which establishes a rule set, the Resources to which the rules apply, and the requesting entity. A policy may be a component of a policy list.

Resource: Any coherent and meaningful concept that may be addressed. A representation of a Resource is typically a document that captures the current or intended state of the Resource. This specification defines the following concrete Resources: Asset, Logical Asset, Node, Account, User, Policy, Rights Token, Rights Locker, Stream, and Discrete Media Rights Token.

UTC: Coordinated Universal Time, a time standard base on the Greenwich Mean Time (GMT) updated with leap seconds (see *http://www.bipm.org/en/scientific/tai/time_server.html*)

1.8 Customer Support Considerations

The customer support Role requires historical data and must occasionally manipulate the status of resources; for example, to restore a mistakenly deleted item. Accordingly, the data models include provisions for element management. For example, most resources contain a ResourceStatus element, which is defined as `dece:ElementStatus-type`. The setting of this element determines the current state of the element (for example, `active`, `deleted`, `suspended`, etc.). The element also records the prior status of the resource.

In general, for each Role specified, there is a corresponding customer support sub-role (for example, `urn:dece:role:coordinator:customersupport`). The degree of access to system-maintained resources that is allowed to customer support roles is generally greater than that allowed to the parent role. This is intended to facilitate good customer support. For more information about the relationship between Nodes in an organization, see section 2.3.

2 Communications Security

Transport security requirements and authentication mechanisms between Users, Nodes, and the Coordinator are specified in [DSecMech]. Implementations SHALL conform to the requirements specified there.

2.1 User Credentials

The Coordinator SHALL verify the User Credentials established by the User.

These credentials SHALL conform to the User Credential Token Profile specified in [DSecMech].

2.1.1 User Credential Recovery

The Coordinator SHALL provide e-mail-based credential recovery.

After the User has recovered his or her credentials, the Coordinator SHALL send an e-mail message to the User's primary `EmailAddress`, indicating that the User's password has been changed.

2.1.1.1 E-mail-based User Credential Recovery

To initiate an e-mail-based password recovery process, the User may use the password-recovery mechanisms provided by the Web Portal, or a Node may employ the UserValidationTokenCreate API defined in section 14.1.7. In either case, an e-mail message is sent, by the Coordinator, to the provisioned primary `EmailAddress`.

The confirmation e-mail SHALL adhere to the requirements set forth below in section 2.1.2.

The confirmation e-mail SHALL contain a confirmation token, and instructions for the User.

The confirmation token SHALL be no fewer than the number of alphanumeric characters determined by the defined Ecosystem parameter DCOORD_E-MAIL_CONFIRM_TOKEN_MINLENGTH.

This token SHALL be valid for the minimum length of time determined by the defined Ecosystem parameter DCOORD_E-MAIL_CONFIRM_TOKEN_MINLIFE, and SHALL NOT be valid for more than the maximum length of time determined by the defined Ecosystem parameter DCOORD_E-MAIL_CONFIRM_TOKEN_MAXLIFE. It can be used only once.

The Coordinator SHALL require the User to provide a valid confirmation token before establishing a new password.

The Coordinator SHALL provide the means to distinguish and select between multiple Users with the same email address.

After the token is submitted by the User, the Coordinator SHALL require the User to establish a password. Note that the User may reuse the same password.

The Coordinator SHALL then accept the User's credentials.

2.1.2 Securing E-mail Communications

E-mails sent to Users MAY include links to the Coordinator.

Senders SHOULD make a reasonable effort to avoid sending DNS names, e-mail addresses, and other strings in a format that may be converted to HTML anchor (<A/>) entities when displayed in email user agents. That is, links to the Coordinator should be the only 'clickable' items in email messages.

2.2 Invocation URL-based Security

Many of the URL patterns defined in the Coordinator APIs include identifiers for resources like Account or User. Whenever present, those identifiers SHALL be verified against the corresponding values available in the security context of the invocation. For instance, a call to the RightsTokenCreate() API is performed by invoking a URL in the form:

```
[BaseURL]/Account/{AccountID}/RightsToken
```

where:

`AccountID` is the identifier for the Account. (AccountIDs are unique to the Node.)

The Coordinator SHALL compare the identifiers employed in the Resource locations (that is, the URLs) to the identifiers supplied in the Delegation Security Token.

The Coordinator SHALL verify the AccountID in the invocation URL, against the corresponding value in the presented Delegation Security Token.

2.3 Node Authentication and Authorization

The Coordinator SHALL require all Nodes to authenticate in accordance with the security provisions specified in [DSecMech].

2.3.1 Node Authentication

Nodes SHALL be identified by their NodeID in the associated Node's x509 certificate as defined in [DSecMech]. The list of approved Nodes creates an inclusion list that the Coordinator SHALL use to authorize access to all Coordinator resources and services. Access to any Coordinator interface by a Node whose identity cannot be mapped SHALL be rejected. The Coordinator MAY respond with a TLS alert message, as specified in [RFC4346], [RFC5246] or [SSL3] as applicable.

2.3.2 Node Authorization

Node authorization is enabled by an access-control list that maps Nodes to Roles. A Node is said to possess a given Role if the DECE Role Authority function, provided by the Coordinator, has asserted that the Node has the given Role in the Coordinator.

API interfaces specify any necessary Delegation Security Token requirements that may be required for API invocation. Requests omitting Delegation Security Tokens where API requirements indicate they are required SHALL result in the appropriate 4xx HTTP response.

Each API call defined in this document has a corresponding list of Roles that are authorized to invoke said API call. A Node SHALL NOT invoke an API call while donning a Role that is not present in the corresponding authorized Roles list.

A Node SHALL NOT don more than one Role. The roles are enumerated in Table 2 and Table 3 on page 26.

The Coordinator SHALL verify the Delegation Security Token, as defined in [DSecMech], which:

- SHALL be a valid, `active` token issued by the Coordinator.
- SHALL contain at least an AccountID (and SHOULD contain a UserID), each of which SHALL be unique in the Coordinator-Organization namespace.
- SHALL map to the associated API endpoint, by matching the AccountID and UserID of the endpoint with the AccountID and the UserID in the Delegation Security Token (as described in section 2.2).
- SHALL be presented by a Node identified in the token, by matching the Node subject of the certificate with a member of the <Audience> element of the Delegation Security Token.

2.3.2.1 Node Equivalence in Policy Evaluations

The following relational diagram shows the Coordinator API authorization model. For the purposes of evaluating API authorization, the Coordinator SHALL evaluate policies established on Nodes, Roles and Organizations. Although one can consider an Organization as a set of Roles mapped to different Nodes (see section 6 in [DSystem]) it is better, in the context of the authorization model, to consider an organization as a set of Nodes, each donning a particular role. Such Nodes are considered Affiliated Nodes.

It is possible that an Organization will have more than one Node with identical Roles. In such circumstances, the Coordinator SHALL consider all Nodes in the same Organization, which are cast in the same Role, as the same Node. Of course, their NodeIDs will be different.

For example, consider a retailer, which has Nodes X, Y, and Z. Nodes X and Y are cast in the role `urn:dece:role:retailer`, and Node Z is cast in the role `urn:dece:role:dsp`. In this case, where access to resources (such as a Rights Token) is restricted based on the NodeID and Role, the Coordinator would allow access to the resource to both Nodes X and Y.

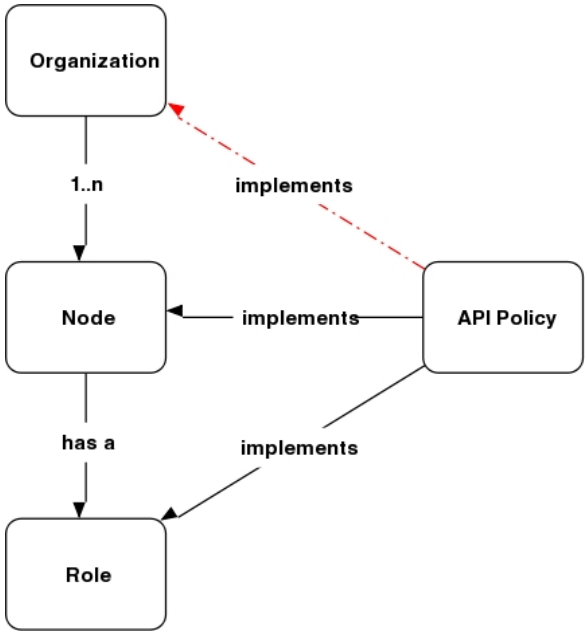


Figure 1: Resource Relationships

2.3.3 Role Enumeration

The following tables describe all Roles in the DECE ecosystem, including each Role's URI and description.

Role	Role Identifier	Description (Informative)
Coordinator	urn:dece:role:coordinator	The Coordinator is a central entity owned and operated by the DECE LLC that facilitates interoperability across Ecosystem services and stores/manages the Account. The Coordinator operates at a known Internet address. The Coordinator Role implicitly has access to all Coordinator APIs.
Coordinator Customer Support	urn:dece:role:coordinator:customersupport	The Tier 2 Customer Support function of the Coordinator Role.
Customer Support	urn:dece:role:dece:customersupport	A generalized Tier 1 customer support function, which is not affiliated with any other Role
Retailer	urn:dece:role:retailer	The Retailer Role provides the customer-facing storefront service and sells Ecosystem-specific content to consumers.
Retailer Customer Support	urn:dece:role:retailer:customersupport	The Tier 1 Customer Support function of the Retailer Role.
LASP	urn:dece:role:lasp	A Locker Access Streaming Provider (LASP) is defined as a streaming media service provider that participates in the Ecosystem and complies with DECE policies to stream Content to LASP Clients.
Linked LASP	urn:dece:role:lasp:linked	A Linked LASP is a service that may stream content to any LASP Client. However, Linked LASPs accounts are persistently bound and provisioned to a single DECE Account versus a User, as Linked LASPs services are not associated with a particular User but to an Account.
Linked LASP Customer Support	urn:dece:role:lasp:linked:customersupport	The Tier 1 Customer Support function of the Linked Lasp Role.
Dynamic LASP	urn:dece:role:lasp:dynamic	A Dynamic LASP is a LASP service that streams Content to a LASP Client to an authenticated User.
Dynamic LASP Customer Support	urn:dece:role:lasp:dynamic:customersupport	The Tier 1 Customer Support function of the Dynamic Lasp Role.
DSP Customer Support	urn:dece:role:dsp:customersupport	The Tier 1 or Tier 2 Customer Support function of the DSP Role supporting its affiliated Retailer Role and (optionally) the Retailers customers.
Content Provider	urn:dece:role:contentprovider	The Content Provider Role is the authoritative source for all DECE Content and is implemented and run by the various content owner or their partners.

Role	Role Identifier	Description (Informative)
Portal	urn:dece:role:portal	This role makes available an interactive web application (referred to as the Web Portal) for the DECE consumer brand and gives Users direct access to Account settings such as a view of their Rights, management of Users in their Account and the ability to add and remove Devices via the use of standard web browsers.
Portal Customer Support	urn:dece:role:portal:customersupport	The Tier 2 Customer Support function of the Portal roles.
DECE	urn:dece:role:dece	The DECE role is reserved for official use by the consortium. It will be employed when the Coordinator is asked by DECE to take some action on a resource in the system (for example, to disable an Account due to fraudulent activities detected by the system).
Access Portal	urn:dece:role:accessportal	The Access Portal Role provides User access to DECE functions such as User and Account management, Device management, and so on, similar to the access that may be provided by a Retailer or LASP, or Web Portal.
Access Portal Customer Support	urn:dece:role:accessportal:customersupport	The Tier 1 Customer Support function of the Access Portal role.

Table 2: Roles

User Access Level	Description
urn:dece:role:account	Represents the Account. Used to describe security requirements on API definitions.
urn:dece:role:user	Represents any user in the system. Used to describe security requirements on API definitions.
urn:dece:role:user:class:basic	A user with the most limited access level to the DECE account it belongs to (see [DSystem] section 7.2.2).
urn:dece:role:user:class:standard	A user with an intermediate access level to the DECE account it belongs to (see [DSystem] section 7.2.2).
urn:dece:role:user:class:full	A user with the highest access level to the DECE account it belongs to (see [DSystem] section 7.2.2).

Table 3: User Access Levels

2.3.4 Node-Based Access Control

As described in the above sections (2.3.1 and 2.3.2) the Coordinator presently requires API invocation authentication and authorization mechanisms in the form of Node TLS credentials, Role authorization, and a Delegation Security Token profile.

In addition to those mechanisms, the Coordinator SHALL enforce a Node-based access control for each API call. This access control is based on a, per API call, list of authorized Nodes. The access control list is defined as follows:

- An enumeration of Nodes which lists the only Nodes allowed to invoke this API call.
- If the list is empty (or not present), the API call cannot be invoked at all.
- If the list contains the '*' character, the API call is available to all Nodes (note that when '*' is included, any other Node name in the list becomes irrelevant).

By default, a Coordinator API call is not subject to Node-based access control. When enforced for a particular API call, it is indicated in that API call's detail description with the following text:
Node-based Access Control: Yes

2.4 User Access Levels

[DSystem] defines three DECE User access levels (section 7.2.2). The Coordinator uses these access levels during the authorization phase of API invocations. The Coordinator calculates the role of a user referenced in the Security Token presented to the API, as it is not present in the token itself. Each API defined in this specification indicates the Security Token Subject Scope, and, when present, will have one or more of the following values:

- `urn:dece:role:user` – the API can be used by any User Access Level. User and Account Policies are used in the authorization decision process.
- `urn:dece:role:self` – the API can be used only on resources that are bound to the User identified in the Security Token presented to the API.
- `urn:dece:role:user:basic` – the API can be used by the Basic-Access User Access Level. User and Account Policies are used in the authorization decision process.
- `urn:dece:role:user:standard` – the API can be used by the Standard-Access User Access Level. User and Account Policies are used in the authorization decision process.

- `urn:dece:role:user:full` – the API can be used by the Full-Access User Access Level. User and Account Policies are used in the authorization decision process.
- `urn:dece:role:account` – the API can by used by any User Access Level. No User Policies are used in any authorization decision process.
- `urn:dece:role:user:parent` – the API can by used by the User identified as the parent or legal guardian of the resource. User and Account Policies are used in the authorization decision process.

A User's access level in combination with the User Resource Status uniquely determine the APIs available to the User at any time. There are several factors that influence User status predominantly including mandatory and elective policy consents for self, additional policies set for the User by other Users within the Account, dependencies between Users (e.g., a Child's status on the Child's Connected Legal Guardian should that Connected Legal Guardian be in a non `active` state for any reason), and other lesser influences. APIs available to a User, as identified in the presented Delegation Security Token, SHALL be as defined in Appendix H, based on User status. API invocations not available to the User per Appendix H SHALL receive an HTTP 403 status code (*Forbidden*).

2.5 User Delegation Token Profiles

There are many scenarios where a Node, such as a Retailer or LASP, is interacting with the Coordinator on behalf of a User. To properly control access to User data while at the same time providing a simple yet secure user experience, authorization is explicitly delegated by the User to the Node using the Delegation Security Token profiles defined in [DSecMech].

The Coordinator SHALL only provide Delegation Security Tokens as described in [DSecMech] Section 5 to Nodes on behalf of Users whose status is one of `urn:dece:type:status:pending`, `urn:dece:type:status:active` or `urn:dece:type:status:blocked:tou`. Valid status values are defined in Table 73, on page 218.

[DSecMech] restricts certain (user-level) Delegation Security Tokens to be evaluated at the Account level. Such evaluations shall supersede any Delegation Security Token Subject Scope defined in this specification.

Every Delegation Security Token Profile defined in [DSecMech] is required to specify methods for acquisition and revocation of the delegation.

Retailer and LASP Node Roles SHALL support at least one Delegation Security Token Profile other than User Credential Token Profile. These Roles will be required to support the request/acquisition method of a Delegation Security Token Profile from the Coordinator, as well as its revocation method.

3 Resource-Oriented API (REST)

The DECE architecture is comprised of a set of resource-oriented HTTP services. All requests to a service target a specific resource with a fixed set of request methods. The set of methods that may be successfully invoked on a specific resource depends on the resource being requested and the identity of the requestor. Such requestors are termed API Clients in this section, any authorized client of an API.

3.1 Terminology

Resources: Data entities that are the subject of a request submitted to the server. Every HTTP message received by the service is a request to perform a specific action (as defined by the method header) on a specific resource (as identified by the URI path).

Resource Identifiers: All resources in the DECE ecosystem can be identified using a URI.

Resource Groups: A resource template defines a parameterized resource identifier that identifies a group of resources, usually of the same type. Resources within the same resource group generally have the same semantics (methods, authorization rules, query parameters, etc.).

3.2 Transport Binding

The DECE REST architecture is intended to employ functionality only specified in [RFC2616]. The Coordinator SHALL be unconditionally compliant with HTTP/1.1. Furthermore, the REST API interfaces SHALL conform to the transport security requirements specified in [DSecMech].

3.3 Resource Requests

For all requests that cannot be mapped to a resource, a 404 status code SHALL be returned in the response. If the resource does not allow a request method, a 405 status code will be returned. In compliance with HTTP 1.1 [RFC2616], the server will also include an “Allow” header.

Authorization rules are defined for each method of a resource. If a request is received that requires Delegation Security Token-based authorization, the server SHALL return a 401 status code. If the client is already authenticated and the request is not permitted for the principal identified by the authentication header, a 401 status code will also be returned.

3.3.1 Character encoding

All XML elements in requests (or responses) utilize UTF-8 character encoding except where XML restricts. Although these Unicode characters don’t usually appear in character sets, the following

Unicode characters are treated as non-whitespace and thus accepted as valid for any element: {U+0085, U+00A0, U+2007, U+202F}. Note that.

All elements accept the following as valid values:

- Non-whitespace
- Whitespace co-mingled with non-whitespace
- Any combination of {U+0085, U+00A0, U+2007, U+202F} with or without other non-whitespace

3.3.2 Connection Reuse

To avoid the overhead associated with establishing HTTP connections and TLS sessions for each transaction, the Coordinator supports persistent HTTP connections, as defined by [RFC2616] as well as TLS session reuse as defined by [RFC5246]. The duration of TLS sessions and HTTP connection reuse is variable based on resource utilization of Coordinator APIs. API Clients can use persistent connections for multiple transactions.

API Clients SHOULD establish persistent HTTP connections in accordance with [RFC2616].

API Clients SHOULD reuse TLS sessions in accordance with [RFC5246].

Although persistent connections and TLS session reuse are strongly recommended, API Clients should negotiate multiple concurrent connections when necessary (e.g. to fulfill multiple requests associated with Resource collections and different user sessions).

3.4 Resource Operations

Resource requests (individually documented below) follow a pattern whereby:

- Successful (2xx) requests which create a new resource return a response containing a reference to the Location of the new resource, and successful (2xx) requests which update or delete existing resources return a 200 status code (*OK*).
- Unsuccessful requests which failed due to client error (4xx) include an Errors object describing the error, and SHALL include language-neutral application errors defined in section 3.14.

All of the status codes used by the Coordinator are standard HTTP-defined status codes.

3.5 Conditional Requests

DECE resource authorities and resource clients SHALL support strong entity tags as defined in Section 3.11 of [RFC2616]. Resource Authorities must also support conditional request headers for use with entity tags (If-Match and If-None-Match). Such headers provide clients with a reliable way to avoid lost updates and the ability to perform strong cache validation. Coordinator services are not required to support the HTTP If-Range header.

Clients SHALL use unreserved-checkout mechanisms as described in [UCheckout] to avoid lost updates.

Following recommendations in [RFC2616], the Coordinator generates both an entity tag (ETag) and a Last-Modified value for all cacheable Resources. The Coordinator includes those validators in its responses.

When an ETag has been provided, Nodes SHALL use the ETag in any subsequent conditional requests (using If-Match or If-None-Match). If both ETag and Last-Modified are available, Nodes SHOULD combine those in any subsequent conditional requests.

The tables 4 and 5 describe the supported HTTP headers for conditional requests. Nodes SHALL only use those headers for the type of request defined in the table 4. The Coordinator ignores any other HTTP header (for caching or conditional request). Note that for conditional requests, the Coordinator reserves the option to respond with 200 accompanied by an ETag and Last-Modified value for any cacheable resource.

HTTP header	Supplied By	Possible Values	Requests	Possible HTTP Error Status Code	Example
If-None-Match	Node	* or ETag	GET/HEAD	304 Not Modified	`If-None-Match: “1352401382138”` or `If-None-Match: *`
If-Match	Node	* or ETag	PUT/DELETE	412 Precondition failed	`If-Match: “1352401382138”` or `If-Match: *`
If-Modified-Since	Node	HTTP-date	GET/HEAD	304 Not Modified	`If-Modified-Since: Wed, 07 Nov 2012 21:18:28 GMT`
If-Unmodified-Since	Node	HTTP-date	PUT/DELETE	412 Precondition failed	`If-Unmodified-Since: Wed, 07 Nov 2012 21:18:28 GMT`

Table 4: Supported HTTP headers for conditional requests

HTTP header	Supplied By	Possible Values	Supported Responses	Example
ETag	Coordinator	(strong validator)	GET/HEAD	`ETag: “1352401382138”`

Last-Modified	Coordinator	HTTP-date (weak validator)	GET/HEAD	`Last-Modified: Wed, 07 Nov 2012 21:18:28 GMT`

Table 5: Coordinator-supported HTTP headers for conditional requests

3.6 Request Throttling

The Coordinator MAY use request throttling techniques at the HTTP or TCP level to manage load on the Coordinator.

The Coordinator MAY use HTTP-level responses, TCP-level responses or in any other appropriate technical responses to protect the Coordinator from harmful behavior such as Denial of Service (DoS) attacks. An example of TCP-level response is limiting the number of concurrent open sockets.

When request throttling is enforced with HTTP, the Coordinator SHALL respond with an HTTP status code 503 (*Service Unavailable*) and include the HTTP header `Retry-After: {delay}`. The value `delay` may be expressed in either time or number of seconds.

The Coordinator SHALL issue `delay` values using algorithms that avoid unfairly starving properly behaving Nodes. Fairness is treating all Nodes equivalently. Starvation is excessive delays, virtually denying service. This requires balancing delays across all requestors.

Nodes and Devices SHALL properly handle HTTP status code 503 (*Service Unavailable*) and with `Retry-After: {delay}` to ensure proper behavior under request throttling conditions.

3.7 Temporary Failures

If the Coordinator requires, for operational reasons, to make resources temporarily unavailable, it may respond with a 307 status code (*Temporary Redirect*) indicating a temporary relocation of the resource. The Coordinator may also respond with a 503 status code (*Service Unavailable*) if the resource request cannot be fulfilled, and the resource (or the requested operation on a resource) cannot be performed elsewhere.

3.8 Cache Negotiation

The Coordinator implements HTTP caching using the following cache response directives:

cache-response-directive	Set By	Comment	Example

Cache-Control:	*max-age*	Coordinator	Defines Resource lifetime at cache server or Node	`Cache-Control: max-age=86400`
	must-revalidate	Coordinator	Forces cache server or Node to refresh Resource when max-age is reached	`Cache-Control: max-age=86400, must-revalidate`
	public	Coordinator	Permits caching even if HTTP authentication or TLS is used.	`Cache-Control: public`
	no-cache, no-store	Coordinator	Skip cached representation and do not store any part of the response.	`Cache-Control: no-cache, no-store`

Table 6: Supported cache-response-directives

The *Cache-control: no-cache, no-store* cache directive is only used in response to the following Coordinator API calls: LicAppJoinTriggerGet, LicAppLeaveTriggerGet, DeviceAuthTokenGet, ResourcePropertyQuery and UserGet (when invoked with the `DataSharing` form of the invocation URL). Note that it is also used in some API calls related to Security Tokens (see [DSecMech]). The Coordinator may apply any of the other cache response directives defined in Table 6 in response to any Coordinator API call.

Nodes SHOULD cache Coordinator Resources in local caches.

When retrieving resources from the Coordinator that are locally cached, Nodes and Devices SHALL utilize HTTP cache validation per [RFC2616].

Collection Resources in the Coordinator (such as the RightsTokenList, StreamList or UserList) have unique cache control processing requirements at the Coordinator. In particular, resource changes, policy changes, client permission changes, etc. may invalidate any client caches, and the Coordinator must consider such changes when evaluating conditional requests of the resource.

Any Resource that requires a Delegation Security Token will result in the HTTP `Vary` header to be included in the response. This instructs a cache to include the listed headers when determining the appropriateness of a cached representation (see [RFC2616] section 14.44). In this case, a response would include:

```
Vary: Authorization
```

Other header values may also be included in the response `Vary` list if needed.

3.9 Request Methods

The following methods are supported by DECE resources. Most resources support HEAD and GET requests but not all resources support PUT, POST or DELETE. The Coordinator does not support the OPTIONS method.

3.9.1 HEAD

To support cache validation in the presence of HTTP proxy servers, all DECE resources SHOULD support HEAD requests.

3.9.2 GET

A request with the GET method returns an XML representation of that resource. If the URL does not exist, an HTTP 404 status code (*Not Found*) is returned. If the representation has not changed and the request contained supported conditional headers, the Coordinator SHALL respond with an HTTP 304 status code (*Not Modified*). Some APIs are provided for non real-time processing, or require post processing after a request is received, and will return a 202 status code (*Accepted*).

3.9.3 PUT and POST

The HTTP PUT method may be used to create a resource when the full resource address is known in advance of the request's submission, or to update an existing resource by completely replacing it. Otherwise, the HTTP POST will be used when creating a new resource. The HTTP PUT request SHALL be used in cases where a client has control over the resulting resource URI. The POST method SHALL NOT be used to update a resource. Unless specified otherwise, all resource creations at the Coordinator are requested via the POST method.

If a request results in the creation of a resource, the HTTP response status code returned SHALL be 201 (*Created*) and a Location header containing the URL of the created resource. Otherwise, successful requests SHALL result in an HTTP 200 status code (*OK*) or HTTP 202 (*Accepted*). Update requests may require post-processing by the Coordinator, in which case, an HTTP 202 status code (*Accepted*) SHALL be returned.

The structure and encoding of the request depends on the resource. If the content-type is not supported for that resource, the Coordinator SHALL return an HTTP 415 status code (*Unsupported Media Type*). If the structure is invalid, an HTTP 400 status code (*Bad Request*) SHALL be returned. The server SHALL return an explanation of the reason the request is being rejected. Such responses are not intended for end users. Clients that receive 400 status codes SHOULD log such requests and consider such errors critical. When updating resources, the invoking Node SHALL provide a fully populated resource (subject to restrictions on certain attributes and elements managed by the Coordinator).

3.9.4 DELETE

The Coordinator SHALL support the invocation of the HTTP DELETE method on resources that may be deleted by clients, based on authorizations governed by the Node's Role, the presented Security Token, and the Node's certificate. An HTTP DELETE request might not necessarily remove the resource from the

database immediately, in which case the response would contain an HTTP 202 status code (*Accepted*). For example, a delete action may require some other action or confirmation before the resource is removed, In compliance with [RFC2616], the use of the 202 status code should enable API clients to track the status of a request.

3.10 Request Encodings

Coordinator services SHALL support the request encodings supported in XML response messages. The requested response content-type need not be the same as the content-type of the request. For various resources, the Coordinator MAY broaden the set of accepted requests to suit additional clients. This will not necessarily change the set of supported response types. All POST and PUT requests SHALL include a Content-Type header with a value of application/xml, and SHALL otherwise conform to the encodings specified in [RFC2616].

3.11 Coordinator REST URL

To optimize request routing, the Coordinator baseURL shall be separately defined for query operations (typically using the HTTP GET method) and provisioning operations (typically using POST or PUT methods).

For this version of the specification, the baseURL for all APIs is:

```
[baseHost] = DGEO_API_DNSNAME

[versionPath] = /rest/2015/02

[iHost] = q.[baseHost]

[pHost] = p.[baseHost]

 [bHost] = b.[baseHost]

[cHost] = c.[baseHost]

[baseURL] = https://[pHost|iHost|bHost][versionPath]
```

For Nodes, query requests (using the HTTP GET or HEAD method) SHALL use the [iHost] form of the URL unless specifically noted in the API definition. All other requests SHALL use the [pHost] form of the URL POST, PUT, DELETE).

The [bHost] is equivalent in all respects to the [pHost] but is intended for non-live activities performed by Nodes on behalf of Users. For example, APIs that support the `LastModified` filter or perform RightsToken maintenance.

The [cHost] is dedicated to consent related queries (see [DGEO]).

The [sHost] is dedicated to security related queries (see [DSecMech]).

The Coordinator will manage the distribution of service invocations using the HTTP 307 status code (*Temporary Redirect*) rather than 302 (*Found*). This enables temporary service relocation without disruption. The Coordinator SHALL redirect the request to hosts within the baseHost definition. API clients SHALL verify that that all redirections remain within the DNS zone or zones defined in the DGEO_API_DNSNAME. Clients SHALL obtain a set of operational baseURLs that may include additional or alternative baseURLs as specified in section 3.12.

If resource invocations of the incorrect HTTP method are received by the Coordinator, a 405 status code (*Method Not Supported***)** will be returned. Finally, if the resource invocation cannot be satisfied because of a conflict with the current state of the requested resource, the Coordinator will respond with a 409 status code (*Conflict*). The requester might be able to resolve the conflict and resubmit the request.

3.11.1 Coordinator REST URL Parameter Encoding

Most Coordinator Resources incorporate well-known parameters in path segments or query parameters values of the Resource location (for example the `{AccountID}` in `[BaseURL]/Account/{AccountID}/RightsToken/List`). Some of these parameters may include characters from the reserved character set (see definition below). Clients SHALL percent-encode such arguments before de-referencing the resource to preserve its integrity.

The reserved character set, in the context of the Coordinator, is composed of the following characters:
"%" / "/" / "?" / "#" / "[" / "]" / "@" / "!" / "$" / "&" / "'" / "(" / ")" / "*" / "+" / "," / ";" / "="

The percent-encoded values of this character set is defined below:

%	/	?	#	[	]	@	!	$	&	'	(	)	*	+	,	;	=
%25	%2F	%3F	%23	%5B	%5D	%40	%21	%24	%26	%27	%28	%29	%2A	%2B	%2C	%3B	%3D

Below are 3 examples highlighting the percent-encoding of parameters (underlined and bold):

`https://q.uvvu.com/rest/2015/02/Account/urn:dece:accountid:org:dece:D40A4402AD/RightsToken/List`

`https://p.uvvu.com/rest/1/06/Asset/Metadata/Basic/urn:dece:cid:eidr-`

```
s:4E04-87A5-2C1F-CA5B-M

https://q.uvvu.com/rest/1/06/Account/urn:dece:accountid:org:dece:D40A4
402AD/User/List?
response=reference&filterclass=urn:dece:type:viewfilter:userbuyer
```

3.12 Coordinator URL Configuration Requests

The Coordinator SHALL publish any additional API baseHost endpoints by establishing, within the DECE DNS zone, one or more SRV resource records as follows:

```
_api._query._tcp.[baseHost]

_api._provision._tcp.[baseHost]
```

The additional resource record parameters are as defined in [RFC2782], for example:

```
_Service._Proto.Name              TTL   Class SRV Pr W  Port Target

_api._query._tcp.decellc.com.     86400 IN    SRV 10 60 5060 i.east.coordinator.decellc.com.

_api._query._tcp.decellc.com.     86400 IN    SRV 20 60 5060 i.west.coordinator.decellc.com.

_api._provision._tcp.decellc.com. 86400 IN    SRV 10 60 5060 p.east.coordinator.decellc.com.

_api._provision._tcp.decellc.com. 86400 IN    SRV 20 60 5060 p.west.coordinator.decellc.com.
```

The response resource record SHALL be from the same DNS zone second-level name. The published DNS zone file SHOULD be signed as defined in [DNSSEC]. Resolving clients SHOULD verify the signature on the DNS zone.

3.13 DECE Response Format

All responses SHALL include:

For all responses:

A custom HTTP Header x-Transaction-Info, which will include the following white space delimited values:

- t=[time expressed as seconds from epoch the response was processed]
- a DECE-unique transaction id string no larger than 48 bytes
- the nodeID of the API client

- the IP address of the API client

This header, in particular, the transactionID, may be useful when involved in customer support activities and during Coordinator client developement.

For example (newline for formatting purposes only):

```
x-Transaction-Info: t=1319570830469360 hpso8ApbMosAAGMt6kYAAAAW
urn:dece:org:org:dece:test:acmestore:retailer 10.1.2.3
```

For 200 status codes:

- A valid Coordinator Resource
- A Location header response (in the case of some new resource creations)
- No additional body data (generally, as a result of an update to an existing resource)

For 300 status codes:

- The Location of the resource

HTTP error status codes (4xx or 5xx) SHOULD include an Error object, with URI and a textual description of the error. A detailed description of each response is provided in section 3.14.

3.13.1 Compression

A request may indicate support for gzip compression, in the response, by setting the `Accept-Encoding` HTTP request header as defined in [RFC2616].

The Coordinator supports the content encodings "gzip" and "compress" defined in [RFC2616] section 3.5 for resources of the types `application/xml`.

Other content types may be supported, but HTTP headers will instruct the client as to whether the Coordinator response is encoded, if an encoded response was requested by the client.

3.14 HTTP Status Codes

All responses from the Coordinator will contain HTTP1.1-compliant status codes. This section details intended semantics for these status codes and recommended client behavior.

3.14.1 Informational (1xx)

The current version of the Coordinator does not support informational status requests for any of its resources.

3.14.2 Successful (2xx)

200 OK
This response message means that the request was successfully received and processed. For requests that result in a change to the identified resource, the client can safely assume that the change has been committed.

201 Created
For requests that result in the creation of a new resource, clients should expect this status code (instead of 200) to indicate successful resource creation. The response message SHALL also contain a Location header field indicating the URL for the created resource. If the request requires further processing or interaction to fully create the resource, a 202 response will be returned.

202 Accepted
This status code will be used to indicate that the request has been received but is not yet complete, for example, updating image references in Basic Metadata. All resource groups that use this status code for a specific method will indicate this in their description. In each case, a separate URL may be specified that can be used to determine the status of the request.

203 Non-Authoritative Information
The Coordinator will not return this header, but intermediary proxies may do so.

204 No Content
Clients should treat this status code the same as a 200 response, but without a message body. There may be updated headers.

205 Reset Content
The Coordinator does not have a need for this status code.

206 Partial Content
The Coordinator does not use Range header fields, and thus has no need for this status code.

3.14.3 Redirection (3xx)

Redirection status codes indicate that the client should visit another URL to obtain a valid response for the request. W3C guidelines recommend designing URLs that do not need changing and thus do not need redirection.

300 Multiple Choices
The requested resource corresponds to any one of a set of representations, each with its own specific location, and agent- driven negotiation information (section 12) is being provided so that the user (or user agent) can select a preferred representation and redirect its request to that location.

The Coordinator only uses this status code in the context of the ResourcePropertyQuery API.

301 Moved Permanently
This status code shall be returned if the Coordinator moves a resource. Clients are STRONGLY RECOMMENDED to remove any persistent reference to the resource, and replace it with the new resource location provided in the Location header.

302 Found
The Coordinator will not use this status code for resource location changes. Instead, status codes 303 and 307 will be used to respond to redirections. The Coordinator does use the status code for certain special resource operations, where its use and meaning will be clearly documented.

303 See Other
The Coordinator will use this status code to indicate that the response will be found at another URI (using an HTTP GET method).

307 Temporary Redirect
If a resource has been temporarily moved, this response shall be used to indicate its temporary location. Clients SHALL attempt to access the resource at its original location in subsequent requests.

304 Not Modified
It is STRONGLY RECOMMENDED that clients perform conditional requests on resources. Clients supporting conditional requests SHALL handle this status code to support response caching.

305 Use Proxy
If edge caching is used by the Coordinator, then unauthorized requests to the origin servers might result in this status code. Clients SHALL handle 305 responses, as they may indicate changes to Coordinator topography, service relocation, or geographic indirections.

3.14.4 Client Error (4xx)

400 Bad Request

This status code is returned whenever the client sends a request using a valid URI path, which cannot be processed due to a malformed query string, header values, or message content. The Coordinator SHALL include a description of the issue in the response and the client should log the error. This description is not intended for end users, and may be used to submit a support issue.

401 Unauthorized

A 401 status code means a client is not authorized to access the requested resource. Clients making a request where the Security Token does not meet specified criteria, or where the user represented by the Security Token is not authorized to perform the requested operation, can expect to receive this response. The Coordinator SHALL respond with an HTTP WWW-Authenticate header as specified in [HTTP11] section 10. Security Token profiles in [DSecMech] specify the appropriate challenge responses.

402 Payment Required

The Coordinator has no need for this status code.

403 Forbidden

The Coordinator will respond with this code where the identified resource is never available to the client, for example, when the resource requested does not match the provided Delegation Security Token.

404 Not Found

This status code indicates that the Coordinator does not understand the resource targeted by the request.

405 Method Not Supported

This status code is returned (along with an Allows header) when clients make a request with a method that is not allowed. It indicates a defect in either the client or the server implementation.

406 Not Acceptable

The Coordinator will not use with this status code. Such responses indicate a misconfigured client.

407 Proxy Authentication Required

The client must first authenticate with the proxy before gaining access to the resource.

408 Request Timeout

The Coordinator may return this code in response to a request that took too long.

409 Conflict

The request could not be fulfilled because of a conflict with the current state of the targeted resource. The 409 status code indicates that the requester may be able to resolve the conflict and resubmit the request.

410 Gone

The Coordinator may return this status code for resources that can be deleted. A status code of 410 can be sent to indicate that the resource is no longer available.

411 Length Required | 416 Requested Range Not Satisfiable

The Coordinator does not use Range header fields, and thus has no need for these status codes.

412 Precondition Failed

This status code should only be sent when a client sends a conditional PUT, POST or DELETE request. Clients should notify the user of the conflict and provide options to resolve it.

413 Request Entity Too Large | 414 Request-URI Too Long

The Coordinator has no need for either of these codes.

415 Unsupported Media Type

If the content-type header of the request is not understood, the Coordinator will return this code. This indicates a defect in the client.

417 Expectation Failed

The Coordinator has no need for this status code.

3.14.5 Server Errors (5xx)

When the Coordinator is unable to process a client request because of server-side conditions, various codes are used to communicate with the client.

500 Internal Server Error

If the server is unable to respond to a request for internal reasons, this status code will be returned.

501 Not Implemented

If the server does not recognize the requested method, it may return this status code. This response is not returned for any of the supported methods.

503 Service Unavailable

This status code will be returned during planned server unavailability. The length of the downtime, if known, will be returned in a Retry-After header. A 503 status code may also be returned if a client exceeds request limits.

502 Bad Gateway | 504 Gateway Timeout

The Coordinator will not reply to responses with this status code directly. Clients may receive this status code from intermediary proxies.

505 HTTP Version Not Supported

Clients that make requests using versions of HTTP other than 1.1 may receive this status code.

3.15 Response Filtering and Ordering

The Coordinator supports range requests using the `ViewFilterAttr-type`. Range requests are provided as query parameters to the following resource collections.

```
[BaseURL]/Account/{AccountID}/RightsToken/List

[BaseURL]/Asset/Metadata/Basic/List

[BaseURL]/Asset/Metadata/Digital/List

[BaseURL]/Asset/Map/List
```

The ViewFilter is used with a parameter identifying the property that will be used to filter the collection.

ViewFilter URI	Description
`urn:dece:type:viewfilter:userbuyer`	Filters the Rights Token collection such that the result set includes only those resources that match the User in the Security Token presented and the PurchaseUser in the Rights Token. This only applies to the RightsToken collections identified above.
`urn:dece:type:viewfilter:lastmodifieddate`	Sorts the returned collection by decreasing update or created date, then by resource ID. This ensures that the most recent changes are always in the first response(s). This filter is implicitly applied to requests for Rights Tokens and metadata (basic, digital and logical assets) lists. Including this filter is not an error for those collections. This filter SHALL NOT be applied to any other collections.

The `OnOrAfter` parameter is a date or a date and time expressed in [ISO 8601] format using the GMT time zone). It is optional and can only be combined with the `urn:dece:type:viewfilter:lastmodifieddate` filter class. When this parameter is present in the request, the Coordinator's response SHALL only contain resources that were updated on or after the provided date. If this parameter is present in the request for RightsLockerDataGet requests (by

reference flavor ONLY), Coordinator SHALL return both active and deleted Rights Tokens in the response.

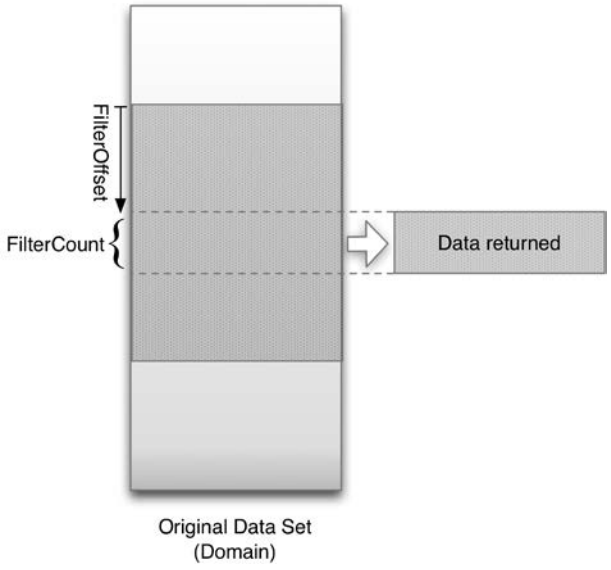
The `FilterCount` parameter is a positive integer used to constrain the number of items in the response collection. No more than `FilterCount` elements will be returned.

The `FilterOffset` parameter may be used to indicate the offset from the beginning of the present request. `FilterOffset` is used in conjunction with `FilterCount` to iteratively query small groups of elements. For instance, to request groups of 10, the first query would have `FilterOffset=0` and `FilterCount=10` (note that `FilterOffset` may be omitted for the first request). The next request would have `FilterOffset=10` and `FilterCount=10`. Next, `FilterOffset=20` and `FilterCount=10`. And, so forth.

The `FilterMoreAvailable` property is a Boolean value that indicates whether there are results in the collection that have not been returned. This value is TRUE when the total number of resources in the collection is greater than `FilterCount` (if present) plus `FilterOffset` (if present).

When the Coordinator services a request for a collection, it SHALL respond with the portion of the entire collection as indicated by the `ViewFilterAttr-type` attributes included in the query string. In such cases, the `ViewFilterAttr-type` attributes will be set on the root element in the response to reflect the data actually returned (e.g., the request exceeds the number of remaining resource). The `FilterClass` used to order the response SHALL be `urn:dece:type:viewfilter:lastmodifieddate` for RightsTokens and Asset lists.

The following illustrates the relationship of these parameters.



Example 1: to create a range request for a Rights Locker, returning 10 items beginning at the 21st item, the request would be:

```
[BaseURL]/Account/{AccountID}/RightsToken/List?FilterOffset=20&FilterCount=10
```

Example 2: following the above example, to create a range request returning the next 10 items, the request would be:

```
[BaseURL]/Account/{AccountID}/RightsToken/List?FilterOffset=30&FilterCount=10
```

Example 3: to create a range request for Basic Assets, returning all asset (references) that were updated or created after 2013-06-01, the request would be:

```
[BaseURL]/Asset/Metadata/Basic/List?FilterClass=urn:dece:type:viewfilter:lastmodifieddate&OnOrAfter=2015-06-01
```

3.15.1 Additional Attributes for Resource Collections

Element	Attribute	Definition	Value	Card.
RightsTokenList, BasicAssetList, DigitalAssetList, LogicalAssetList		Collections of Resources	Each includes the `dece:ViewFilterAttr-type`	
	FilterClass	Filtering performed to generate the response	`xs:anyURI`	0..1
	FilterOffset	FilterOffset indicates the offset from the beginning of the present request. An offset of '0' indicates the beginning of the domain. If not present, the implicit value of FilterOffset is 0.	`xs:nonNegativeInteger`	0..1
	FilterCount	The maximum number of resources in the collection returned	`xs:positiveInteger`	0..1
	FilterMoreAvailable	Indicates whether there are additional results remaining.	`xs:boolean`	0..1

Table 7: Additional Attributes for Resource Collections

3.16 Entity Identifiers

Many Resources are assigned an identifier that is unique within the ecosystem. Those identifiers are defined using the following definition:

Element	Attribute	Definition	Value	Card.
EntityID		Identifiers of the form urn:dece:* as defined in Section 5 of [DSystem]	`dece:EntityID-type restricts xs:anyURI <xs:pattern value=”urn:dece:.*”/>`	

Table 8: EntityID-type definition

4 DECE Coordinator API Overview

This specification defines the interfaces used to interact with the Coordinator. The overall architecture, the description of primary Roles, and informative descriptions of use cases can be found in [DSystem].

The Coordinator interfaces are REST endpoints, which are used to manage various DECE Resources and Resource collections. Most Roles in the DECE ecosystem will implement some subset of the APIs specified in this document.

The sections of this specification are organized by Resource type. API's defined in each section indicate which Roles are authorized to invoke the API at the Coordinator, indicate Delegation Security Token requirements, the URL endpoint of the API, the request method or methods supported at that resource, the XML structure which applies for that endpoint, and processing instructions for each request and response. The "API Invocation by Role" table in Appendix A, provides an overview of the APIs that apply to each Role.

5 Policies

The Coordinator's Policies describe access control and consent rules that govern the behavior and responses of the Coordinator when it interacts with Nodes. These rules are applied to Users, Accounts and Rights. Policies are concise and unambiguous definitions of allowed behavior. A Policy may be one of three types: consent policies, User-age policies, or parental-control policies.

5.1 Policy Resource Structure

Policies are object-oriented, in the sense that Policies are defined as Policy objects that have classes (the Policy class) and are instantiated as a Policy. The Policy Object is encoded in `Policy-type`, which is defined in Table 10, below. The Policy resource contains the various components of a Policy.

5.1.1 Policy Resource

A Policy Resource is a URN that defines the scope of the Policy, that is, the Resource to which the policy applies. For example, for a parental-control policy, the Resource is the established rating. Each policy class defines the applicable Policy Resource or Resources that apply. For more information about the Resources that each Policy class can be applied to, see section 5.5.

5.2 Using Policies

The Policy element is a structure maintained by the Coordinator. It governs Coordinator protocol responses for the Resource it applies to. Other Roles may obtain certain Policies using the provided APIs in order to ensure a consistent user experience.

Geography Policies may dictate default policies or mandatory policies (for example, mandatory Parental Controls for children). Such policies will be created by the Coordinator when the applicable resource is created (for example after UserCreate() of a child). Default policies may subsequently be modified, mandatory policies SHALL NOT be removed, and any attempt to modify or remove them will result in an error response. Mandatory policies are indicated with the `Immutable` attribute.

The Web Portal Role is exempt from all Consent Policy restrictions.

Consent Policies set by a Node may be deleted by that same Node, regardless of the presence of ManageUserConsent.

5.3 Precedence of Policies

When more than one Policy applies to a resource request, they are evaluated in the following order:

- Node-level policies (Requestor is a Node)
- Account-level policies (Resource is the Account)
- User-level policies (including parental-control policies)

Inheritance and mutual exclusiveness of the Policies are addressed in the descriptions of each Policy class. For example, an EnableManageUserConsent Account-level policy would be evaluated before the User-level ManageUserConsent policy would be evaluated.

When Policies are evaluated in cases where the Delegation Security Token is evaluated with an Account-level security context (for example, when the requestor is any of the customer support Roles), User-level Policies SHALL NOT be considered unless otherwise noted in the API. For example, Parental Control Policies are not evaluated when the invoking node is one of the customer support role.

5.4 Policy Data Structures

This section describes the Policy resource model as encoded in the `Policy-type` complex type.

5.4.1 PolicyList-type Definition

The policy list collection captures all policies, including opt-in attestations. It is conveyed in the PolicyList element, which holds a list of individual Policy elements (as defined in section 5.4.1).

Element	Attribute	Definition	Value	Card.
PolicyList			`dece:PolicyList-type`	
	PolicyListID	A unique identifier for the policy list. Used in resource responses after the creation of a set of policies (that is, a POST with a PolicyList in message body)	`dece:EntityID-type`	0..1
Policy		Policy elements	`dece:Policy-type`	1..n

Table 9: PolicyList-type Definition

5.4.2 Policy Type Definition

The following table describes the Policy-type complex type

Element	Attribute	Definition	Value	Card.
	PolicyID	This unique identifier of the Policy is used when referring to an established policy in protocol messages. It is a Coordinator-defined value, and is therefore omitted from the PolicyCreate messages. It SHALL NOT be altered by PolicyUpdate() messages.	xs:anyURI	0..1
	Immutable	A boolean indication of whether the Policy can be altered, typically, as a result of a Geography Policy. Its default value is false.	xs:boolean	0..1
PolicyClass		Policy Classes are defined in section 5.5 If the request body of an API includes a PolicyList, it SHALL always include PolicyClass. It is provided as optional exclusively for the support of Security Token bindings.	dece:EntityID-type	0..1
Resource		The Resources that each Policy Class can be applied to are listed in section 5.5.	xs:anyURI	0..n
RequestingEntity		The identifier of the User ,Node or Organization making the request (for example, a user who is trying to view the title of a digital asset). If absent or NULL, the policy applies to all requesting entities. If several requesters are identified, the policy applies to each of them. Note: RequestingEntity in the case of a Node means the Node to which the policy applies, not necessarily the Node calling the API.	dece:EntityID-type	0..n
PolicyAuthority		The identifier of the policy decision point, which is currently the Coordinator.	dece:EntityID-type defaults to urn:dece:role:coordinator	0..1
ResourceStatus		Information about the status of the policy, see section 17.2.	dece:ResourceStatus-type	0..1

Table 10: Policy Type Definition

5.5 Policy Classes

The policy classes define each policy. They determine its evaluation criteria, which are characterized by a set of rules and a rule-composition algorithm.

Policies Classes are expressed as URNs [RFC3986] of the form:

```
urn:dece:type:policy: + ClassString
```

where:

`ClassString` is a unique identifier for a Policy class.

The availability of policy classes and their evaluation criteria may be modified by Geography Policies (see [DGeo]). Implementations should consult any applicable Geography Policy to ensure adherence to local jurisdictional requirements.

Some consent policies below have corresponding resources detailing the nature of the consent (for example, the terms of use). Since these may vary according to jurisdiction, [DGeo] appendices will specify the precise resource location for each policy class, which will conform to the resource location pattern defined in section 5.5.3.

5.5.1 Account Consent Policy Classes

Consent policy classes describe the details of the consents granted by or to Accounts and Users. Account-level consent policies, when in place, apply to named resources within an Account. When the last remaining Full Access User's Delegation Security Token is revoked or expired for a Node, the Coordinator deletes any corresponding Account-level policies.

The following policy consents for Roles `Portal[:customersupport]`, `Coordinator[:customersupport]` and `dece:customersupport` shall be implicitly evaluated as "true":

- LockerViewAllConsent
- EnableUserDataUsageConsent
- EnableManageUserConsent
- ManageAccountConsent

Implicit evaluation means that there will be no instantiation of these policies; rather, when one of these Roles' invocation of a Coordinator API is being processed, any operation dependent on the presence of one of these policy consents shall be considered to have the policy set even though such an instantiation may not exist.

5.5.1.1 LockerViewAllConsent

Class Identifier: `urn:dece:type:policy:LockerViewAllConsent`

Resource: One or more Rights Lockers associated with the Account (identified by RightsLockerID).

RequestingEntity: One or more entities that requested the policy's application (identified by NodeID or OrgID).

PolicyCreator: The User who provided consent (identified by UserID).

Description: This policy indicates a full access User has consented to the entity identified in the RequestingEntity obtaining all items in the Rights Locker (while still evaluating other policies which may narrow the scope of the access to the locker). The Resource for policies of this class SHALL be one or more RightsLockerIDs associated with the Account. The PolicyCreator is the UserID of the User who instantiated the policy. When establishing a link (represented by a Delegation Security Token) with any LASP role, this Policy SHALL be automatically created by the Coordinator, enabling LASPs to provide basic streaming services. Without it, the LASP Node would not be able to verify the existence of any Rights Tokens in a Rights Locker.

5.5.1.2 EnableUserDataUsageConsent

Class Identifier: `urn:dece:type:policy:EnableUserDataUsageConsent`

Resource: One or more Users associated with the household Account (identified by UserID).

RequestingEntity: One or more entities that requested the policy's application (identified by NodeID or OrgID).

PolicyCreator: The user who provided consent (identified by UserID).

Description: This policy indicates that a full-access user has consented to enabling users within the Account to establish `urn:dece:type:policy:UserDataUsageConsent` policies on their own User Resource. For more information about the UserDataUsageConsent policy, see section 5.5.2.2.

5.5.1.3 EnableManageUserConsent

Class Identifier: `urn:dece:type:policy:EnableManageUserConsent`

Resource: One or more Users associated with the Account (identified by UserID).

RequestingEntity: One or more entities that requested the policy's application (identified by NodeID or OrgID).

PolicyCreator: The user who provided consent (identified by UserID).

Description: This policy indicates that a full-access user has consented to enabling users within the Account to establish `urn:dece:type:policy:ManageUserConsent` policies on their own User Resource. For more information about the ManageUserConsent policy, see section 5.5.2.1.

It also allows the entity identified in the RequestingEntity to perform write operations on the identified User resource. This policy is required to enable creation and deletion of Users by any Role other than the Web Portal.

5.5.1.4 ManageAccountConsent

Class Identifier: `urn:dece:type:policy:ManageAccountConsent`

Resource: The Account (identified by AccountID).

RequestingEntity: One or more entities that requested the policy's application (identified by NodeID or OrgID).

PolicyCreator: The user who provided consent (identified by UserID).

Description: This policy indicates that a full access user has consented to allow the entity identified in the RequestingEntity element to manage Account information, including the creation of new Users in the Account.

5.5.2 User Consent Policy Classes

User-level consent policies apply to an identified User resource. Typically, the PolicyCreator value should be the UserID of the User to which the policy applies. Some implementations, however, may allow a User in the Account to create consent policies on another User's behalf.

The following policy consents for Roles `Portal[:customersupport]`, `Coordinator[:customersupport]` and `dece:customersupport` shall be implicitly evaluated as "true":

- ManageUserConsent
- UserDataUsageConsent
- UserLinkConsent

Implicit evaluation means that there will be no instantiation of these policies; rather, when one of these Roles' invocation of a Coordinator API is being processed, any operation dependent on the presence of

one of these policy consents shall be considered to have the policy set even though such an instantiation may not exist.

5.5.2.1 ManageUserConsent

Class Identifier: `urn:dece:type:policy:ManageUserConsent`

Resource: One or more Users (identified by UserID).

RequestingEntity: One or more entities that requested the policy's application (identified by NodeID or OrgID).

PolicyCreator: The user who provided consent (identified by UserID).

Description: This policy indicates that a user has consented to allow the entity identified in the RequestingEntity element to update and delete the identified User resource. It requires the prior application of the Account-level EnableManageUserConsent policy. The deletion of the last remaining ManageUserConsent policy in an Account MAY result in the deletion of the ManageAccountConsent policy for the Node (see [DGeo] section 2.6.5).

5.5.2.2 UserDataUsageConsent

Note: This policy class is no longer in use.

Class Identifier: `urn:dece:type:policy:UserDataUsageConsent`

Resource: One or more Users (identified by UserID) and zero or more Rights Lockers (identified by RightsLockerID).

RequestingEntity: One or more entities that requested the policy's application (identified by NodeID or OrgID).

PolicyCreator: The user who provided consent (identified by UserID).

Description: This policy indicates that a user has consented to allow the identified entity to use the named resources' data for marketing and other purposes. The UserDataUsageConsent policy does not influence the Coordinator's response to a Node. It requires the prior application of the Account-level EnableUserDataUsageConsent policy.

5.5.2.3 TermsOfUse

Class Identifier: `urn:dece:type:policy:TermsOfUse`

Resource: The legal agreement and version identifier.

RequestingEntity: The user on whose behalf consent was provided (identified by UserID). This is frequently, but not always the same as the User identified in the PolicyCreator element.

PolicyCreator: The user who accepted the agreement (identified by UserID).

Description: This policy indicates that a user has agreed to the DECE terms of use. The Resource identifies the precise legal agreement and version that was acknowledged by the user. This identifier is managed by DECE. The presence of this policy is mandatory, and certain operations related to Content consumption (Rights acquisition and streaming) will be forbidden until this policy has been established.

The text of the Terms of Use and Privacy Policy may be updated with or without requiring Users to accept the new version. Acceptance by a User of an updated Terms of Use/Privacy Policy SHALL be recorded as a new TermsOfUse policy resource. The value of the `Resource` element is the URL referring to the TermsOfUse accepted by the User.

The ability of Nodes other than the Web Portal to set this Policy is determined by applicable policies prescribed in [DGeo].

5.5.2.4 UserLinkConsent

Class Identifier: `urn:dece:type:policy:UserLinkConsent`

Resource: A User (identified by UserID).

RequestingEntity: One or more entities that requested the policy's application (identified by NodeID or OrgID).

PolicyCreator: The User who provided consent (identified by UserID).

Description: This policy indicates that a user has consented to allow the identified entity to establish a persistent link between a Node and the Coordinator-managed User resource.

The Web Portal Role operated by the Coordinator is granted this policy implicitly and it cannot be removed.

Link consent SHOULD be granted at Organization level, by providing an OrgID in the `RequestingEntity` element. Granting this policy to an Organization will grant access to any Node that is mapped to that Organization.

Any Node MAY create or delete UserLinkConsent for itself and for other Nodes in the same Organization. Any Node, with appropriate Account Management consent, MAY create or delete UserLinkConsent for any other Node.

UserLinkConsent is independent of other Consent Policies (e.g., ManageUserConsent).

The presence of the UserLinkConsent policy affects the duration of Delegation Security Tokens (see [DSecMech] 4.3.2.1). Changes to the Delegation Security Tokens (creation, revocation, expiration, etc.) have no effect on the UserLinkConsent policy. In case of affiliated nodes that share the same Delegation Security Token, when the UserLinkConsent policy is deleted for a Node, the Coordinator SHALL NOT revoke the corresponding Delegation Security Token until the last UserLinkConsent policy for any affiliated node is deleted.

5.5.2.5 Connected Legal Guardian Attestation Policy

To record the attestation of a Connected Legal Guardian, the Connected Legal Guardian Attestation Policy defined below MAY be required in accordance with the applicable Geography Policy document. The CLG attestation policy SHALL be created on any User which has a LegalGuradian element set.

Applicability of this policy class is goverened by jurisdictional requirements. Geography Policy documents will indicate when this policy is required, and the conditions of its use. Typically, it will apply to Users under the DGEO_AGEOFMAJORITY defined in a Geography Policy document.

Class Identifier: `urn:dece:type:policy:CLGAttestation`

Resource: The UserID of the Child or Youth User for whom the CLG Attestation policy applies

RequestingEntity: null

PolicyCreator: The Connected Legal Guardian User who attests to being the Connected Legal Guardian (identified by UserID).

Description: Indication that the User identified in the PolicyCreator element attests to being the Connected Legal Guardian. Geography Policy documents will specify when this policy must be created for a User.

5.5.2.6 Special Geographic Privacy Assent Policy Class definition

The Special Geographic Privacy Assent policy class is a general policy class which may be employed by Geography Policy documents to indicate extreme privacy requirements must be enforced, and records the acknowledgement of notification to the `PolicyCreator`. The applicable processing rules for the application of this policy are defined in Geography Policy documents, and the proper geography is determined by the User or Account-level Country and/or regional properties for the User or Account. For

example, in the United States, this policy may be used to indicate that necessary COPPA notification obligations have been fulfilled and acknowledged by the Connected Legal Guardian.

Class Identifier: `urn:dece:type:policy:GeoPrivacyAssent`

Resource: The User to whom the special restrictions apply and assent was required (identified by UserID).

RequestingEntity: null

PolicyCreator: The User who provided the assent (identified by UserID).

Description: Indication that the assent obligations have been completed by the authorized User. Some Users shall be required to have this policy in place in order for a User to be considered `active` and available for use. The applicable Geography Policy document will specify which Users may be impacted, and the processes for obtaining assent.

5.5.2.7 DataSharingConsent

Class Identifier: `urn:dece:type:policy:DataSharingConsent`

Resource: A User (identified by UserID).

RequestingEntity: One or more entities that requested the policy's application (identified by NodeID or OrgID).

PolicyCreator: The user who provided consent (identified by UserID).

Description: This policy indicates that a user has consented to share a limited amount of data (to enable a licensee to create an Account using data from the Coordinator). This consent can only be manipulated (CREATE, GET, DELETE, UPDATE) by the Coordinator during a Federation Security Token request, as allowed for by [DGeo] or by the `urn:dece:role:dece:customersupport` Role (GET). DataSharingConsent is recorded at the Coordinator for tracking purposes but is not displayed at the Web Portal or in any other UI.

5.5.2.8 AdditionalEmailConsent

Class Identifier: `urn:dece:type:policy:AdditionalEmailConsent`

Resource: A User (identified by UserID).

RequestingEntity: SHALL be DECE Org only (identified by OrgID urn:dece:org:org:dece:o:dece).

PolicyCreator: The user who provided consent (identified by UserID).

Description: This policy indicates that a User has consented to participate in email-based education and survey campaigns by the entity identified in the RequestingEntity. The Resource for policies of this class SHALL be the User identified in the presented Delegation Security Token.

This policy may be created at any time and it can be removed any time. There are no dependencies on any other consent policies. This policy is not evaluated by the Coordinator nor does it have any effect on any of the current processes in the Coordinator, including user notifications.

Retailers, LASPs, AccessPortal Roles and their customersupport specialization MAY collect this policy on behalf of DECE.

[DGeo] will indicate if this policy is disallowed in specific geographies. The Coordinator will only enforce cases where the region prohibits this policy.

No notification email should be generated by Coordinator when this policy is created, deleted, or updated.

5.5.3 Obtaining Consent

5.5.3.1 Obtaining Consent at the Coordinator

Consent should occur with direct interaction between a User and the Coordinator. To obtain consent at the Coordinator, the Node SHOULD establish an authenticated request through the Users browser or other HTTP user-agent. The methods and mechanisms for creating this request SHALL be defined by a suitable Security Token Profile defined in [DSecMech].

Requesting Nodes SHOULD implement the same Security Token Profile employed for establishing delegation with the Coordinator and that Node.

Both User-level and Account-level Consent policies may be requested at once. The Coordinator will determine which policies are allowed to be established and agreed to by the User, based on the identified Users Role, age, or other restriction which may be defined for policies.

When Nodes and Users cannot be combined in a manner requested in the request, the Coordinator will attempt to reduce the combination in such a way to maximally honor the request. However, if the combination includes multiple UserIDs in the Consent, the Coordinator may not be able to perform any reasonable reduction, and will not attempt to collect the consent from the User, and instead return a suitable Security token Profile error response.

Nodes might request Consent Policies in either the aggregate (group) form, as defined in the User Interface Requirements appendix of the License Agreement or in a Geography Policy, however, the Coordinator will allow a User to disaggregate the group, allowing individual selection of Policies. The Coordinator always respond with a PolicyList including references to the individual policies the User chose, even in the case where the User chose to accept the aggregated request.

5.5.3.2 Obtaining Consent at a Node

In some jurisdictions, Nodes may collect consent directly from the User, and provision the applicable policies. Geography Policies shall indicate whether this mode of consent collection is available for a given jurisdiction. The profile shall indicate, in addition, which (if any) consent policies can be combined in any fashion, or if each must be agreed to by the User individually.

To obtain consent, and to ensure consistent terms are provided to the User, the Coordinator shall provide a set of well-known resource locations (URLs) that shall be used to deliver the applicable terms and detailed language. These locations shall provide language-specific plain text and un-styled HTML suitable for use in various implementations.

The well known URL for the Terms Of Use Policy SHALL redirect to the permanent location of the most recent policy language associated with the consent. Other well-known URLs such as Privacy Policy and Cookie Policy SHALL provide the most recent policy language but do not redirect to a separate dated version.

The well-known location is defined as follows:

```
[DGEO_TEXTBASE]/Consent/Text/{geo}[-{lang}]/{PolicyClass}/{format}/Current[/Short]
```

and the permanent location is as follows:

```
[DGEO_TEXTBASE]/Consent/Text/{geo}[-
{lang}]/{PolicyClass}+":"+{versiondate}/{format}[/Short]
```

where:

- {geo} is the Geography Identifier as defined in the Appendixes of [DGeo]
- {lang} is an optional parameter that signals a preferred presentation language. Allowed values only include those languages specified in the appendixes of [DGeo]. If not included, the language is assumed to be en. Requests for invalid languages will result in an HTTP 404 Not Found response.
- {PolicyClass} is the class identifier for the consent policy defined in section 5.5.1 and 5.5.2
- {versiondate} is the version of the {PolicyClass}. This versioned resource provides a reference to the specific policy language accepted by the User. [DGeo] defines the specific version dates, as required.
- {format} is either:
  - text - a plain text, UTF-8 [UNICODE] representation of the Policy Class' resource
  - html - an HTML4 representation of the Policy Class' resource
- Short may be optionally postpended to the initial URLto signal that the "short form" of the policy text is desired. If /Short is included in the well-known location form (i.e. the form using "../Current"), the server SHALL respond with a redirect to the then active policy resource /Short form URL.

Nodes SHALL NOT use URLs with "/Short" in Policies that require a consent URL to be included in the Resource element of a Policy.

When requesting the first form ("…/Current"), the response from this resource shall be a redirect to the then active policy resource (e.g. the second form above). The Node SHALL use this second URL to identify the consent policy version, as specified in sections 5.5.1 and 5.5.2.

An example for a Terms Of Use policy creation for a specific country:

```
<?xml version="1.0" encoding="UTF-8"?>
<dece:PolicyList xmlns:dece="http://www.decellc.org/schema/2015/03/coordinator">
    <dece:Policy>
        <dece:PolicyClass>urn:dece:type:policy:TermsOfUse</dece:PolicyClass>
```

```
<dece:Resource>https://c.uvvu.com/Consent/Text/us/urn:dece:type:policy:TermsOfUse:
20140811/html
         </dece:Resource>

<dece:RequestingEntity>urn:dece:userid:org:dece:ACED2DDA477DC85BE0401F0A0F994274</dece
:RequestingEntity>
        <dece:PolicyAuthority>urn:dece:role:coordinator</dece:PolicyAuthority>
        <dece:ResourceStatus>
            <dece:Current
CreatedBy="urn:dece:userid:org:dece:ACED2DDA477DC85BE0401F0A0F994274">
                <dece:Value>urn:dece:type:status:active</dece:Value>
            </dece:Current>
        </dece:ResourceStatus>
    </dece:Policy>
</dece:PolicyList>
```

5.5.4 Allowed Consent by User Access Level

The following table defines which User Level may set Policies within a Policy Class.

Policy Class	Basic-Access	Standard-Access	Full-Access
LockerViewAllConsent	N/A	N/A	Yes
EnableUserDataUsageConsent	N/A	N/A	Yes
EnableManageUserConsent	N/A	N/A	Yes
ManageAccountConsent	N/A	N/A	Yes
ManageUserConsent	Self Only	Self Only	Self Only
UserDataUsageConsent	Self Only	Self Only	Self Only
TermsOfUse	Self Only	Self Only	Yes
UserLinkConsent	Self Only	Self Only	Self Only
DataSharingConsent	Self Only	Self Only	Self Only

Table 11: Consent Permission by User Access Level

For each User Level, a Yes indicates that the policy may be set by that user; alternatively, an N/A indicates that the policy may not be set (these policies apply to the entire Account). The notation Self Only indicates that the policy may be set by that user, but applied only to that user's own User resource.

5.5.5 Parental Control Policy Classes

Note: This section is subject to change. Nodes SHALL NOT create Parental Control Policies.

Parental Control policies SHALL identify the user for which the policy applies in RequestingEntity, and identify the Rating Value as the Resource. The Coordinator performs no enforcement of parental control policies. Their inclusion here is to make possible a shared parental control repository that may be shared across Retailers and LASPs. By default, this specification defines no default Parental Control Policies. The absence of any Parental Control Policies is equivalent to `urn:dece:type:policy:ParentalControl:NoPolicyEnforcement`.

Geography Policies MAY specify default Parental Control Policies, mandatory Parental Control Policies, or both. In such cases, the Coordinator SHALL create such policies when an applicable User is created. Ratings-based policies created in such cases SHALL be of the Rating System prescribed by the applicable Geography Policy. In addition, Geography Policies may specify default or mandatory policy settings for `urn:dece:type:policy:ParentalControl:BlockUnratedContent`, `urn:dece:type:policy:ParentalControl:AllowAdult`, and `urn:dece:type:rating:us:RIAA:ProhibitExplicitLyrics`.

5.5.5.1 BlockUnratedContent

Class Identifier: `urn:dece:type:policy:ParentalControl:BlockUnratedContent`

Resource: NULL

RequestingEntity: The User that the parental control applies to (identified by UserID).

PolicyCreator: The User that created the parental control policy (identified by UserID).

Description: This policy indicates that the identified User should not have access to content in the Rights Locker which does not carry a rating corresponding to a ratings system for which the User has a Parental Control setting, and should apply to viewing, purchasing and, in some cases, the playback of content in the Rights Locker. The default policy for new users is to allow unrated content (that is, this policy is not created by default when a new User is created). Whether this Policy is set to TRUE when a new User is created is defined in the applicable Geography Policy.

This policy class is superseded by the application of the: `urn:dece:type:policy:ParentalControl:NoPolicyEnforcement` policy.

5.5.5.2 AllowAdult

Class Identifier: `urn:dece:type:policy:ParentalControl:AllowAdult`

Resource: NULL

RequestingEntity: The User that the parental control applies to (identified by UserID).

PolicyCreator: The User that created the parental control policy (identified by UserID).

Description: This policy indicates that the identified User is allowed access to digital content whose BasicAsset metadata has the AdultContent attribute set to TRUE. Whether this Policy is set to TRUE when a new User is created is defined in the applicable Geography Policy.

5.5.5.3 RatingPolicy

Class Identifier: `urn:dece:type:policy:ParentalControl:RatingPolicy`

Resource: The rating system value identifier (defined below).

RequestingEntity: The User that the parental control applies to (identified by UserID).

PolicyCreator: The User that created the parental control policy (identified by UserID).

Description: This policy indicates that a rating-based parental-control policy has been applied to a User. This policy applies to the viewing and playing of content. Rating identifiers take the general form:

```
urn:dece:type:rating:{region}:{system}:{ratings}
```

Rating reasons are similarly identified as:

```
urn:dece:type:rating:{region}:{system}:{ratings}:{reason}
```

The defined values for these parameters correspond to the column headings of Section 8 in [MLRatings], with the exception that the applicable ISO country codes in [ISO3166-1] SHALL be used.

Rating Policies may combine rating and reason identifiers to construct complex parental control policies.

When determining which rating systems to employ for the creation of Parental Controls, Nodes SHOULD use systems matching the User's Country value. Note that Nodes may choose from any available rating systems.

These policies are non-inclusive when evaluating for authorization to a RightsToken based on the Parental Control. That is, a policy with a Resource of `urn:dece:type:rating:us:mpaa:pg13` would only allow access to any MPAA rated content which is rated PG-13. To allow access to several ratings at once, the policy must include each rating for the identified system (for example, `urn:dece:type:rating:us:mpaa:pg13`, `urn:dece:type:rating:us:mpaa:pg`, as well as `urn:dece:type:rating:us:mpaa:g`, to enable access to PG13 and below in the United States). This eliminates ambiguities in interpretation when policies are evaluated. Parental Control user interfaces may provide simplified controls for a better user experience. This policy class is superseded by the application of the: `urn:dece:type:policy:ParentalControl:NoPolicyEnforcement` policy.

5.5.5.4 NoPolicyEnforcement

Class Identifier: `urn:dece:type:policy:ParentalControl:NoPolicyEnforcement`

Resource: NULL.

RequestingEntity: The User that the parental control applies to (identified by UserID).

PolicyCreator: The User that created the parental control policy (identified by UserID).

Description: This policy prohibits enforcement of any parental control policies for the identified User or Users. This policy class applies to the purchase, listing, and playing of digital content.

5.5.6 Policy Abstract Classes

All policy classes are defined in a hierarchical fashion, for example, the ParentalControl policy classes. To facilitate a simpler interface to policy queries (that is, the PolicyGet API), the following abstract policy class identifiers may be used:

- `urn:dece:type:policy:ParentalControl` -- Identifies all Parental Control policy classes as defined in section 5.5.5
- `urn:dece:type:policy:Consent` -- Identifies all consent policy classes as defined in sections 5.5.1 and 5.5.2.

5.5.7 Evaluation of Parental Controls

Note: This feature is no longer supported. It is retained here for historical purposes and potential re-introduction in the future.

It is recommended that nodes implementing Parental Control enforcement do so using the algorithm defined in [CMCRSchema] section 5.

5.5.7.1 RIAA Policies

Although there are no widespread content rating systems in the music industry, the Recording Industry Association of America (RIAA) defines an Explicit Content label for music videos. Unlike the movie industry, the Unrated Content label equates to universal availability. Because the RIAA rating system is the sole representation of explicit content, its syntax differs from normal ratings-based policies.

Class Identifier: `urn:dece:type:policy:ParentalControl:RatingPolicy`

Resource: `urn:dece:type:rating:us:RIAA:ProhibitExplicitLyrics`

RequestingEntity: The User that the parental control applies to (identified by UserID).

PolicyCreator: The User that created the parental control policy (identified by UserID).

Description: This policy indicates that an explicit content parental-control policy has been applied to a User for music or music videos. This policy applies to the viewing and playing of content.

5.6 Policy APIs

5.6.1 PolicyGet()

5.6.1.1 API Description

The PolicyGet API can be invoked to obtain the details of any policy.

5.6.1.2 API Details

Path:

For User-level policies:

```
[BaseURL]/Account/{AccountID}/User/{UserID}/Policy/{PolicyID}|{PolicyListID}

[BaseURL]/Account/{AccountID}/User/{UserID}/Policy/{PolicyClass}

[BaseURL]/Account/{AccountID}/User/{UserID}/Policy/List
```

For Account-level policies:

```
[BaseURL]/Account/{AccountID}/Policy/{PolicyID}|{PolicyListID}

[BaseURL]/Account/{AccountID}/Policy/{PolicyClass}

[BaseURL]/Account/{AccountID}/Policy/List
```

Method: GET

Authorized Roles:

```
urn:dece:role:portal[:customersupport]
urn:dece:role:dece[:customersupport]
urn:dece:role:retailer[:customersupport]
urn:dece:role:accessportal[:customersupport]
urn:dece:role:lasp:linked[:customersupport]
urn:dece:role:lasp:dynamic[:customersupport]
```

When `ManageAccountConsent` is not set for a particular Node for a particular Account/User, all forms of Policy Get requests made by that Node shall return policies limited to those set only for the requesting Node or set for its parent Org corresponding to the same Account/User. . However, if the `ManageAccountConsent` policy is set on the account for the requesting Node, all policies meeting the criteria shall be returned.

*The Node's access to the policy class is subject to the user's access level, as defined in the following table.

Policy Class	Basic Access	Standard Access	Full Access
LockerViewAllConsent	Yes	Yes	Yes
EnableUserDataUsageConsent	N/A	N/A	Yes
EnableManageUserConsent	N/A	N/A	Yes
ManageAccountConsent	N/A	N/A	Yes
ManageUserConsent	Self Only	Self Only	Yes†‡
UserDataUsageConsent	Self Only	Self Only	Yes†‡
TermsOfUse	Self Only	Self Only	Yes†‡
UserLinkConsent	Self Only	Self Only	Yes†‡
Parental Control	Yes	Yes	Yes‡
NoPolicyEnforcement	Yes†	Yes†	Yes†‡
AllowAdult	Yes†	Yes†	Yes†‡

† The Node's access to the policy class is allowed only if the `urn:dece:type:policy:ManageUserContent` policy is set to TRUE.

‡ The policy class may be further restricted based on Geography Policies found in [DGeo] limiting access to this policy class to the User's Connected Legal Guardian.

Table 12: User Access Level per Role

Request Parameters:

AccountID is the unique identifier for an Account

UserID is the unique identifier for a User

`PolicyID` is the unique identifier for a single Policy

`PolicyListID` is the unique identifier for a Policy collection (which was originally created as a list)

PolicyClass may be one of:

- A specific DECE policy class, for example: `urn:dece:type:policy:ManageUserConsent`
- A Policy Group URN defined in an applicable Geography Profile

- A policy abstract class, for example: `urn:dece:type:policy:ParentalControl`,

Security Token Subject Scope:

`urn:dece:role:user:self`

`urn:dece:role:user:parent`

Applicable Policy Classes: All

Request Body: None.

Response Body:

PolicyList or PolicyListFull.

Element	Attribute	Definition	Value	Card.
PolicyList		See Table 9	`dece:PolicyList-type`	

5.6.1.3 Behavior

The Coordinator responds with an enumeration of Policies with the identified PolicyClass, associated with Account (as applicable), and associated with the identified User (as applicable). Parental controls are only accessible if the ManageUserConsent policy is set to TRUE for the identified User.

The ManageUserConsent and ManageAccountConsent policies SHALL always evaluate to TRUE for the Web Portal and DECE and Coordinator roles (and their associated customer support roles).

5.6.2 PolicyCreate(), PolicyUpdate(), PolicyDelete()

5.6.2.1 API Description

Policies cannot be altered by creating or updating the resource to which the policy has been applied (for example, user-level policies cannot be updated using the UserUpdate API). Policies can be manipulated only by invoking these APIs.

5.6.2.2 API Details

Path:

The following forms can be used for POST:

```
[BaseURL]/Account/{AccountID}/Policy/{PolicyClass}

[BaseURL]/Account/{AccountID}/Policy/List

[BaseURL]/Account/{AccountID}/User/{UserID}/Policy/{PolicyClass}

[BaseURL]/Account/{AccountID}/User/{UserID}/Policy/List
```

The following forms can be used for PUT and DELETE:

```
[BaseURL]/Account/{AccountID}/Policy/{PolicyID}

[BaseURL]/Account/{AccountID}/User/{UserID}/Policy/{PolicyID}
```

Methods: POST | PUT | DELETE

Authorized Roles:

All policy classes may be manipulated using these APIs. The Consent Policy Classes may also be updated through the Consent mechanism, described in section 5.5.3.

Role	Parental Control
urn:dece:role:portal	●[1]
urn:dece:role:portal:customersupport	●
urn:dece:role:dece:customersupport	●
urn:dece:role:retailer	●[1]
urn:dece:role:retailer:customersupport	●[1]
urn:dece:role:accessportal	●[1]
urn:dece:role:accessportal:customersupport	●[1]
urn:dece:role:lasp:linked	●[1]
urn:dece:role:lasp:linked:customersupport	●[1]
urn:dece:role:lasp:dynamic	●[1]
urn:dece:role:lasp:dynamic:customersupport	●[1]

[1] Nodes may manipulate the listed policy on behalf of full-access Users only. This requires the application of the Account-level EnableManageUserConsent policy as well as the ManageUserConsent policy.

Request Parameters:

`AccountID` is the unique identifier for an Account
`UserID` is the unique identifier for a User
`PolicyID` is the unique identifier for a single Policy

`PolicyClass` is a DECE Policy Class, Policy Group, or Policy abstract class URN, for example, `urn:dece:type:policy:ParentalControl`

Security Token Subject Scope:

```
urn:dece:role:user:self
urn:dece:role:user:parent
```

Applicable Policy Classes:

ParentalControl Policy Classes (defined in section 5.5.5)

Request Body:

PolicyList is passed in GET and PUT request messages.

Element	Attribute	Definition	Value	Card.
PolicyList		See Table 9	`dece:PolicyList-type`	

A DELETE request message has no body.

Response Body: None.

5.6.2.3 Behavior

For PolicyCreate, Nodes SHALL NOT include a PolicyID attribute in a request. Nodes can create or update more than one consent policy at a time for a single User by submitting a PolicyList, that includes a sequence of Policy elements (one per consent policy class). When submitting multiple policies at once, if any individual policy cannot be created or updated, the entire list fails, and an error is returned (Error ID to be added).

The DataSharingConsent policy SHALL ONLY be created by itself (not included in a list with other policies).

For PolicyUpdate, Nodes SHALL include the PolicyID as provided by the Coordinator when updating existing Policies. If, as Part of the Update, additional Policies are being added, such new Policies SHALL NOT include the PolicyID attribute.

Multiple requesting entities may be included for any Policy with the restriction that each requesting entity value be from the same Organization as the requesting Node.

Resource Status, if included in the request body, shall be ignored. A Policy may only be deleted by use of the PolicyDelete method. The Coordinator SHALL generate the appropriate PolicyIDs as required.

The Coordinator responds with an enumeration of Policies with the identified PolicyClass, associated with Account (as applicable), and associated with the identified User (as applicable).

- For PolicyCreate, if the Policy does not exist, it is created. If a Policy already exists in the identified PolicyClass, an error is returned.
- For PolicyUpdate, if the Policy exists, the identified resource or resources are updated. If a Policy does not exist in the identified PolicyClass, an error is returned. If the Policy element in the update request contains no resources, an error is returned.
- For PolicyDelete, if the Policy exists, its Resource Status is set to *deleted*.

Parental controls are only accessible if the ManageUserConsent Account-level policy is set to TRUE, allowing access to the requested User resource.

The ManageUserConsent policy SHALL always evaluate to TRUE for the Web Portal and DECE Role (and their associated customer support roles), unless prohibited by a localized Terms Of Use (TOU), as required by a Geography Policy. For more information about Geography Policy requirements, see Appendix F.

Policy classes that depend upon the presence of other policies (for example, the EnableManageUserConsent class) may be created, updated or deleted irrespective of the presence of the dependant class, however, such policies will not have any effect until the parent policy class has been established with the necessary scope. For example, if the EnableManageUserConsent policy class is deleted, the subordinate ManageUserConsent policy class may remain in place. The policy evaluation during API invocation of, for instance, UserUpdate, will result in a 403 Forbidden response, as the absence of the EnableManageUserConsent policy class prevents access to the API.

Additional constraints are documented in the description of each Policy Class.

5.7 Consent Policy Dependencies and API availability

Figure 2 below documents the dependencies between consent policies. It also describes the set of APIs that becomes available after a policy is set in the related Account.

This figure indicates that some Policies may be created automatically by the Coordinator, which is determined by the `Country` property on the User, and the applicable Geography policy in [DGeo]. Automated policy creation, if any, SHALL occur when a Delegation Security Token is issued to the Node for any User in the Account. Please check [DGeo].

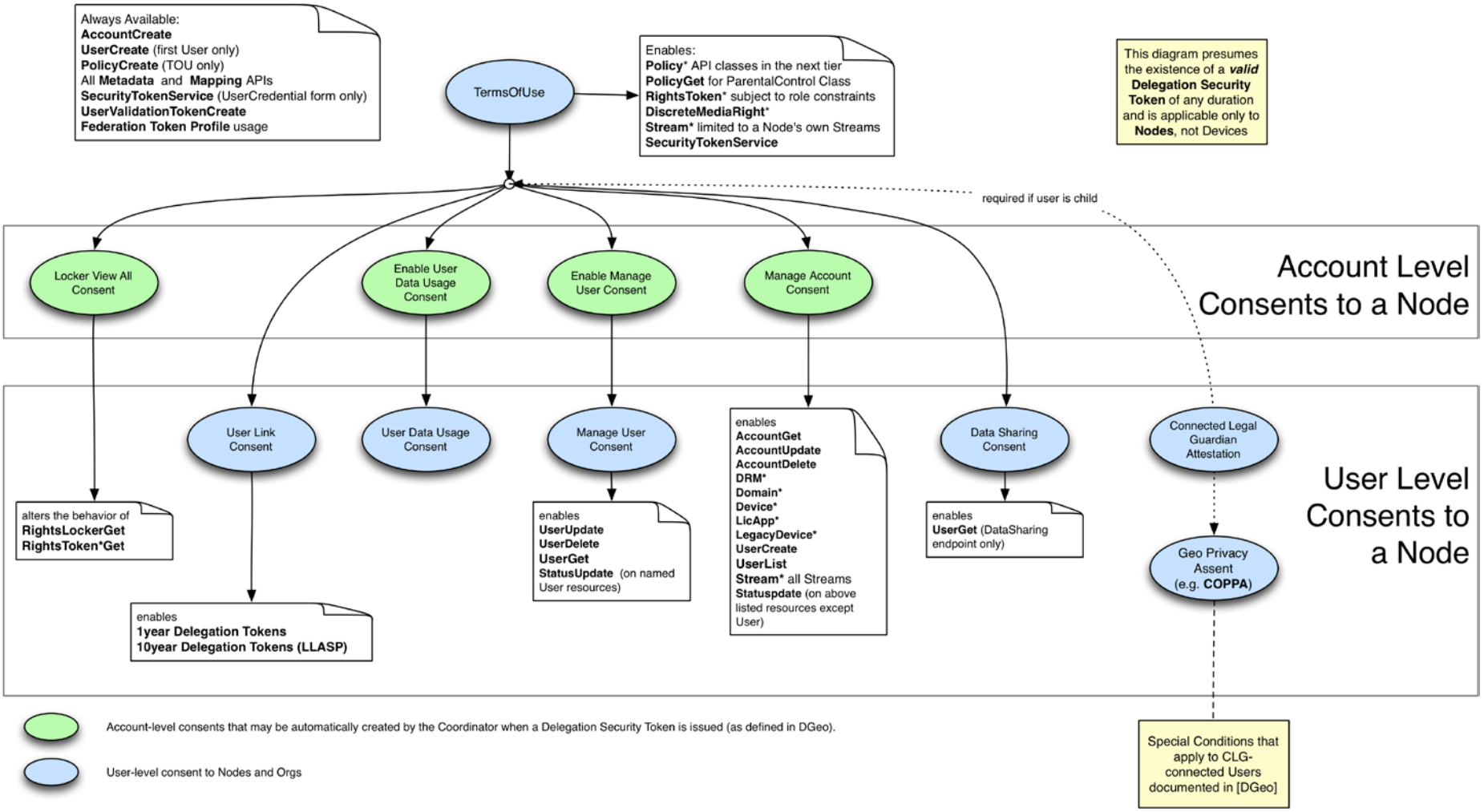


Figure 2: Policy Dependence and Enabled APIs

5.8 Grace Periods for User Actions

DECE defines 2 main grace periods to help manage the lifecycle of user's status. Each grace period is associated with an ecosystem parameter defining its duration. The expiration of a grace period always results in a status change for the User. The 2 grace periods are as follows:

- Terms Of Use Acceptance: this grace period defines the amount of time a newly created User has to accept the DECE Terms Of Use. Its duration is represented by the DGEO_TOU_ACCEPTANCE_GRACE_PERIOD ecosystem parameter as defined in [DGeo].
- Terms Of Use Update: this grace period defines the amount of time an existing User has to accept a revision of the DECE Terms of Use. Its duration is represented by the DGEO_TOU_UPDATE_GRACE_PERIOD ecosystem parameter as defined in [DGeo].

5.8.1 User Status and Grace Periods

The following figures describe various scenarios based on different values for the aforementioned grace periods as well as initial User status. Each diagram shows the evolution of the User status that can be triggered by either actions taken by the User or the expiration of a grace period.

For these figures, the terms Adult, Youth and Child are used as defined in [DGeo].

5.8.1.1 New Adult and Youth Users

In Figure 3, the TOU grace period is greater than 0, but is not exceeded.

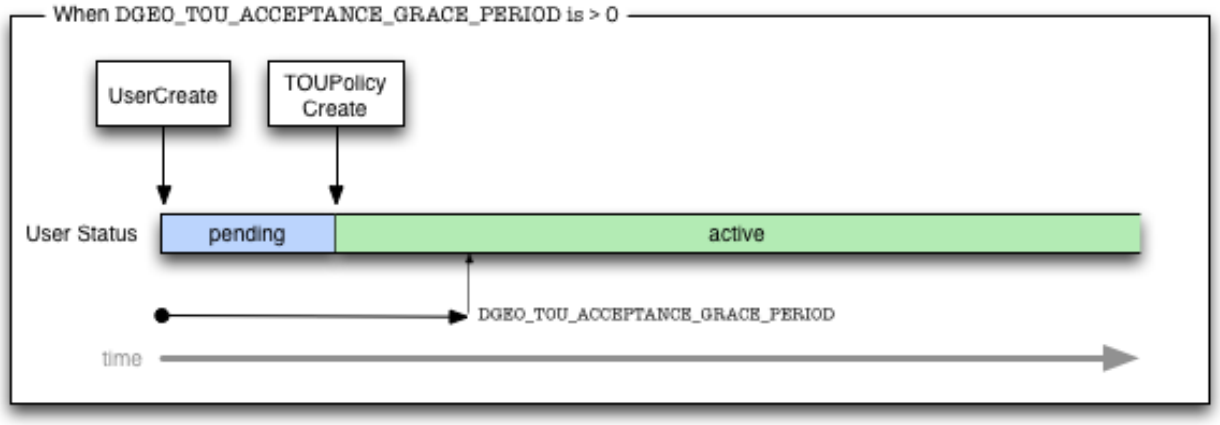


Figure 3: DGEO_TOU_ACCEPTANCE_GRACE_PERIOD > 0 – User accepts within the grace period

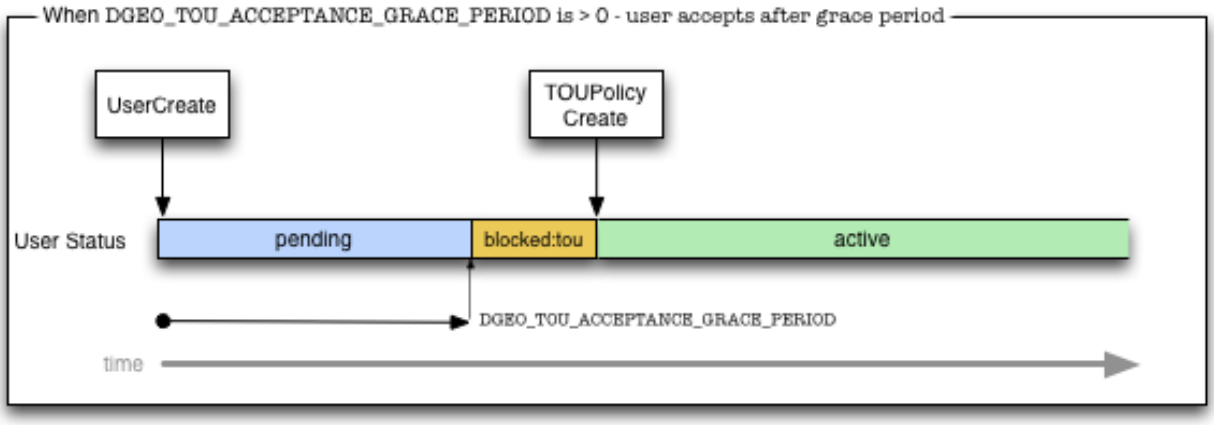


Figure 4: DGEO_TOU_ACCEPTANCE_GRACE_PERIOD > 0 – User accepts after the grace period

In Figure 5, the `DGEO_TOU_ACCEPTANCE_GRACE_PERIOD` is 0, and therefore, the User is created in a `blocked:tou` status.

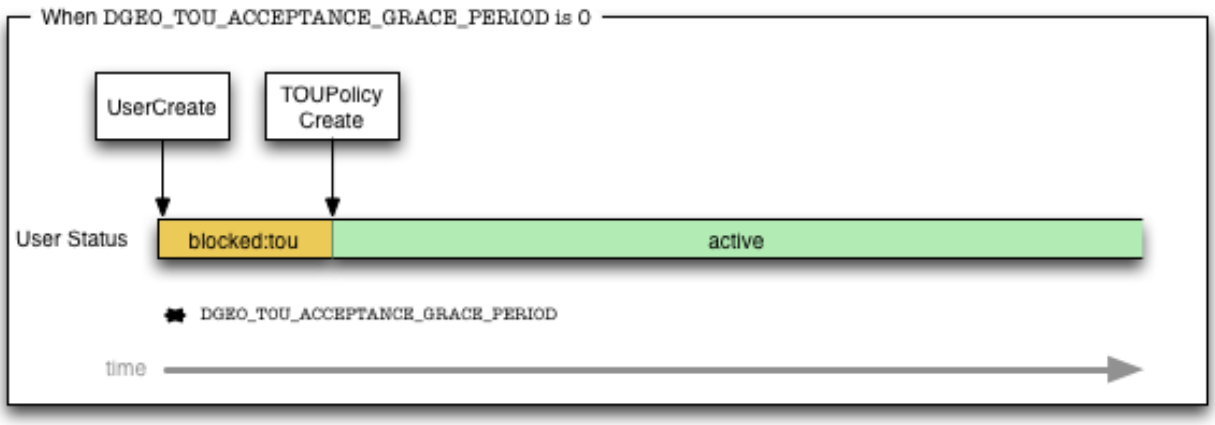


Figure 5: DGEO_TOU_ACCEPTANCE_GRACE_PERIOD is 0

5.8.1.2 TOU Change for Adult and Youth Users

In Figure 6, when the `DGEO_TOU_UPDATE_GRACE_PERIOD` is greater than 0, and the User accepts the new TOU within the grace period, no status change will occur.

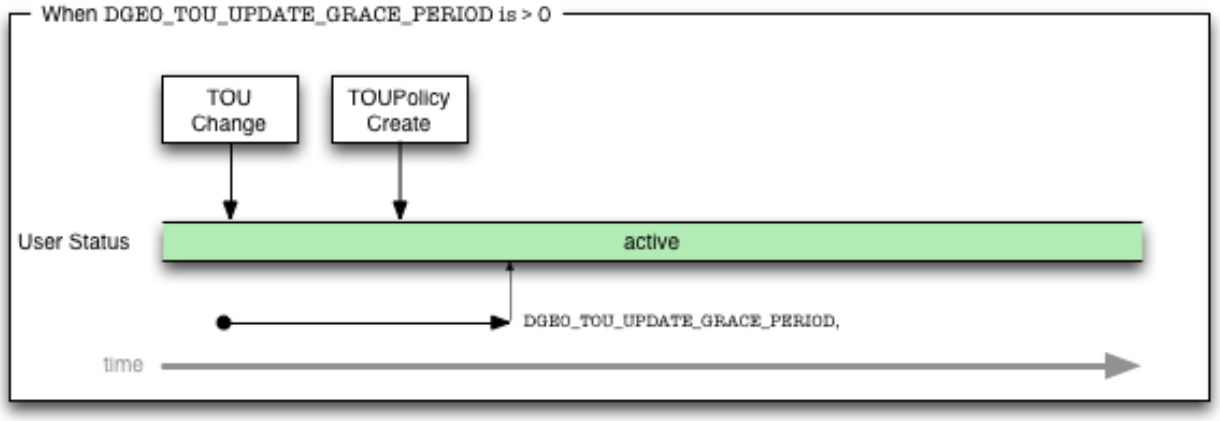


Figure 6: DGEO_TOU_UPDATE_GRACE_PERIOD is > 0

However, in the case where the `DGEO_TOU_UPDATE_GRACE_PERIOD` is 0, all Users will enter the `blocked:tou` status until the new TOU is accepted.

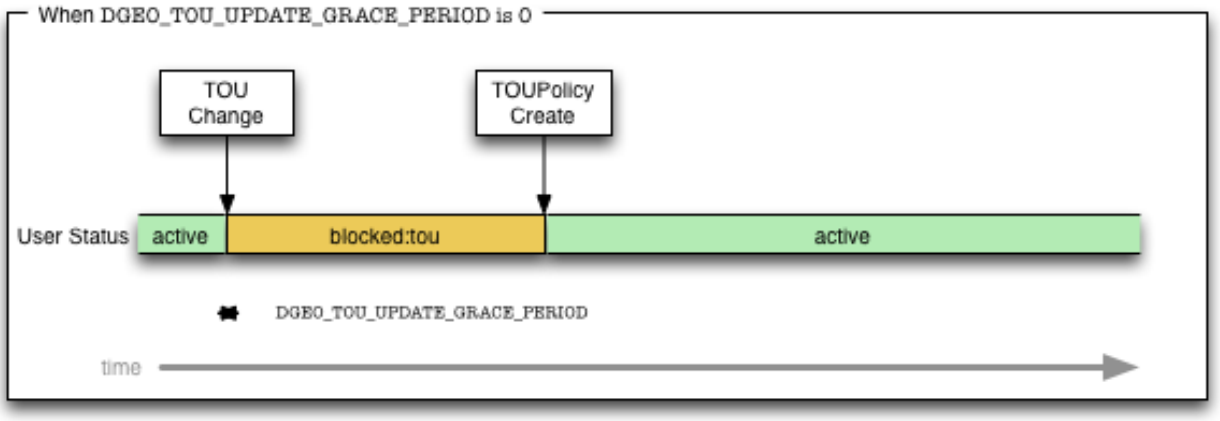


Figure 7: DGEO_TOU_UPDATE_GRACE_PERIOD is 0.

5.8.1.3 New Child User with Connected Legal Guardian

Some geographies may require additional policies, prohibit Child Users from accepting TOU and require a Connected Legal Guardian (CLG). In this case, modeled after the US Geography Profile in [DGeo], the CLG Attestation must occur prior to TOU acceptance (on behalf of the Child). In addition, the GEOPrivacyAssent policy is required in order to fully activate the Child. In Figure 8, with an initial TOU

grace period (exceeded) of greater than 0, the Child moves through several inactive statuses prior to becoming `active`.

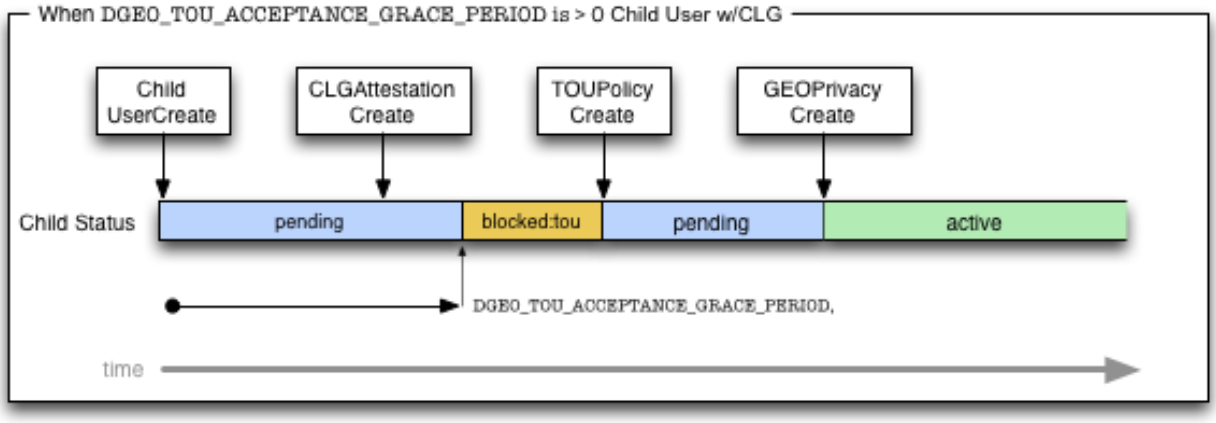


Figure 8: When DGEO_TOU_ACCEPTANCE_GRACE_PERIOD is > 0 - Child User with CLG

In the case of a TOU grace period of 0, Figure 9 shows the initial state of `blocked:tou`, as with an Adult, and still a `pending` status as before, until the GeoPrivacy Assent has been given.

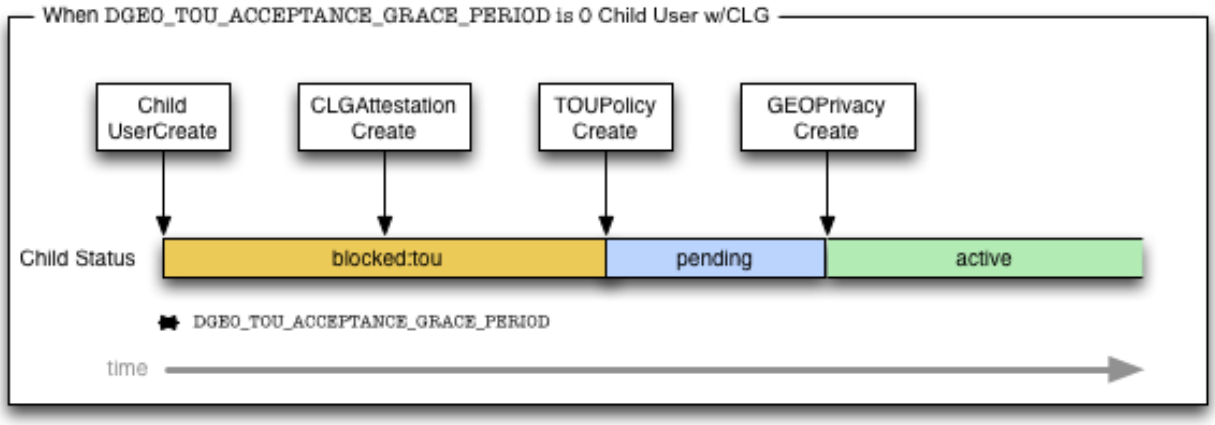


Figure 9: When DGEO_TOU_ACCEPTANCE_GRACE_PERIOD is 0 - Child User with CLG

5.8.1.4 TOU Change for Child Users and their CLG

When TOU change occurs, in the presence of a Child and their CLG, both Users will be required to accept the new TOU, with the CLG accepting first. In Figure 10, when there is a grace period, provided the CLG accepts the TOU for themselves and the Child, they will both remain in the `active` status.

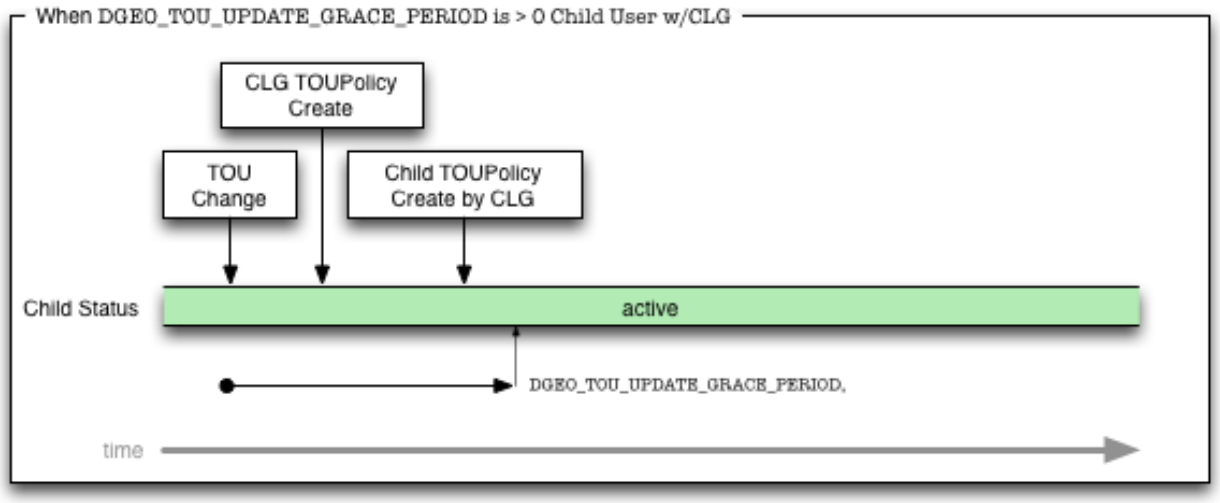


Figure 10: TOU Change with Grace Period > 0 Child and CLG

Without a grace period, the CLG (as an Adult from above in Figure 7), the Child, however moves into a `blocked:clg` status, because the CLG is no longer `active`. Once the CLG has accepted the new TOU, the Child moves to `blocked:tou`, because the CLG is now `active`. Once the CLG accepts the TOU for the Child, the child returns to the `active` status.

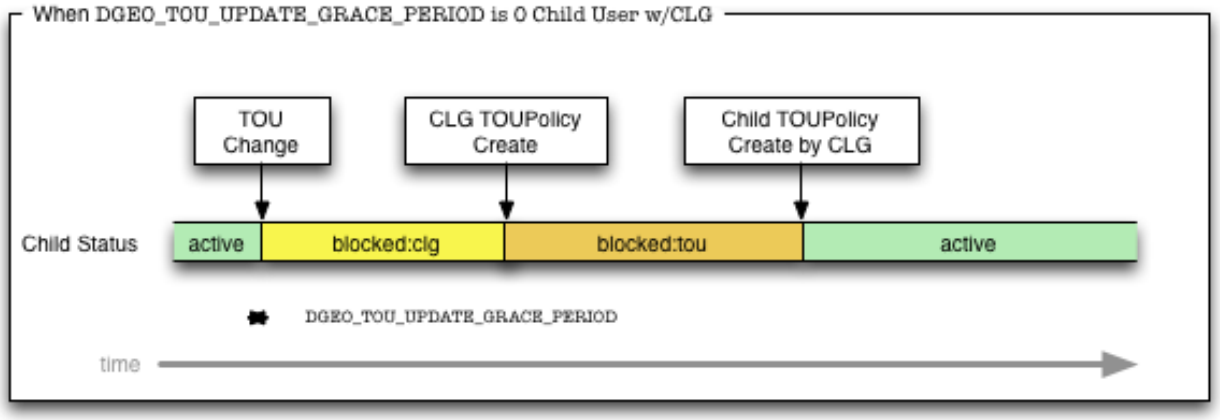


Figure 11 TOU Change with Grace Period of 0 Child and CLG

5.9 Policy Status Transistions

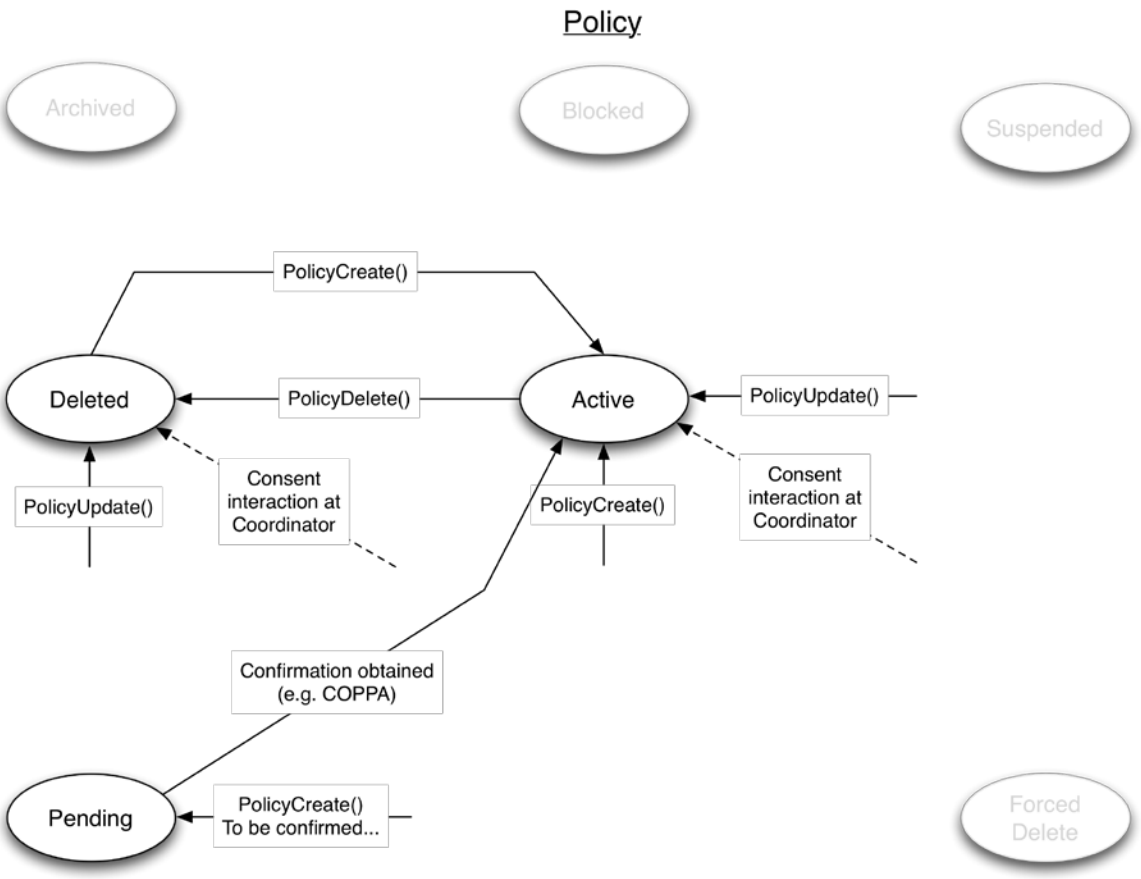


Figure 12: Policy Status Transitions

6 Assets: Metadata, ID Mapping and Bundles

An asset is a digital representation of content (films, television programs, video games, electronic books, etc.); it is described to the system and its users using *metadata*—data about the data.

Note: All of the assets tend to be stable over time and as such are easily cacheable.Nodes SHOULD use this information to establish a local cache of Basic/Digital/Logical Asset information. This supports conditional requests using OnOrAfter and HTTP directives for quick and efficient queries to the Coordinator to verify the asset information. Nodes SHOULD always query the Coordinator to verify the local cache.

6.1 Metadata Functions

DECE metadata schema documentation may be found in the *DECE Metadata Specification* [DMeta]. Metadata is created, updated and deleted by Content Providers, and may be retrieved by the Web Portal, Retailers, LASPs and DSPs.

The Coordinator SHALL enforce scheme-independent requirements for identifiers defined in [DSystem] section 5.5. The Coordinator MAY support scheme-specific requirements for identifiers defined in [DSystem] Section 5.5 and associated referenced specifications.

6.1.1 MetadataBasicCreate() and MetadataDigitalCreate()

6.1.1.1 API Description

These functions are used to create basic or digital asset metadata at the Coordinator.

6.1.1.2 API Details

Path:

```
[BaseURL]/Asset/Metadata/Basic

[BaseURL]/Asset/Metadata/Basic/{ContentID}

[BaseURL]/Asset/Metadata/Digital

[BaseURL]/Asset/Metadata/Digital/{APID}
```

Methods: POST (without parameters) | PUT (with parameters)

Authorized Roles:

```
urn:dece:role:contentprovider[:customersupport]
```

Request Parameters:

APID is the Asset Physical identifier for a digital asset
ContentID is the content identifier for Content.

Security Token Subject Scope: None

Opt-in Policy Requirements: None

Request Body:

For a Basic Asset:

Element	Attribute	Definition	Value	Card.
BasicAsset		See **Table 20**	`dece:AssetMDBasic-type`	

For a Digital Asset:

Element	Attribute	Definition	Value	Card.
DigitalAsset		See **Table 15**	`dece:DigitalAsset Metadata-type`	

Response Body: None

6.1.1.3 Behavior

This creates a Basic Metadata or Digital Asset Metadata at the Coordinator. Content Providers SHALL conform to the requirements defined in [DPublish] and [DMeta], and the Coordinator will enforce the presence of the stated mandatory values.

These functions MAY return a *202 Accepted* HTTP status code, as additional processing of the created Resource may be required (for example, the verification and caching of image resources referenced in the metadata). If no processing is required and no errors are encountered with the request, a 200 OK HTTP status code will be returned, and the resource will be immediately available.

In some cases, such as viruses found, the Coordinator Customer Support Role may notify the Content Provider if an error is unrecoverable.

Whenever a new image resource is provided as part of a new or updated Basic Metadata, the Coordinator will perform several actions on the image resource. For each BasicMetadata/LocalizedInfo/ArtReference element:

- Fetch the image from the provided URL

- Scan the image for viruses, and quarantine as necessary

For the set of images provided in BasicMetadata/LocalizedInfo/ArtReference elements

- If necessary image assets are absent, create missing image assets. This SHALL be in accordance with [DMeta] Section 3.2.
- Publish all the image assets at Coordinator-controlled URLs
- Update the BasicMetadata/LocalizedInfo/ArtReference to reflect these new image locations

If the ArtReference is a Coordinator URL for an existing, different, asset, the Coordinator will establish a new copy of that existing image, to provide the new asset with its own distinct URL, in order to avoid inadvertant overwriting of images across assets. No additional virus scanning will be performed on such images.

During concurrent metadata creation or update, the Coordinator SHALL support multiple usages of the same Node–sourced image. This will permit Content Providers to, for instance, simultaneously register or update a set of episodic assets that reference the same image.

Note that it may take significant time to ingest images, especially if some resolutions need to be generated by the Coordinator. The Content Provider can determine status using the GET APIs described below.

Note: While re-provisioning ContentIDs, Coordinator SHALL increment the UpdateNum (to highest UpdateNum +1) if the submitted UpdateNum is lower than the previously provisioned values for the ContentId or if no UpdateNum is submitted. In such cases, Content Providers SHOULD perform MetadataBasicGet to retrieve the latest UpdateNum to be used in the subsequent update requests.

6.1.1.4 MetadataBasicUpdate() and MetadataDigitalUpdate()API Description

These functions are used to update a Basic Metadata or Digital Asset Metadata at the Coordinator. Updates consist of complete replacement of the metadata. There is no provision for updating individual data elements.

6.1.1.5 API Details

Path:

```
[BaseURL]/Asset/Metadata/Basic/{ContentID}

[BaseURL]/Asset/Metadata/Digital/{APID}
```

Methods: PUT

Authorized Roles:

```
urn:dece:role:contentprovider[:customersupport]
```

Request Parameters:

APID is the Asset Physical identifier for a digital asset
ContentID is the content identifier for a digital asset.

Security Token Subject Scope: None

Opt-in Policy Requirements: None

Request Body:

For a Basic Asset:

Element	Attribute	Definition	Value	Card.
BasicAsset		See **Table 20**	`dece:AssetMDBasic-type`	

For a Digital Asset:

Element	Attribute	Definition	Value	Card.
DigitalAsset		See **Table 15**	`dece:DigitalAsset Metadata-type`	

Response Body: None

6.1.1.6 Behavior

The entry matching the Asset identifier (ContentID or APID) identified in the resource endpoint is updated. Updates may be performed only by the Node that created the asset.

Content Providers SHALL conform to the requirements defined in [DPublish] section 3.1, and the Coordinator will enforce the presence of the stated mandatory values.

These functions MAY return a *202 Accepted* HTTP status code, as additional processing of the updated Resource may be required (for example, the verification and caching of image resources referenced in the metadata).

In some cases, such as viruses found, the Coordinator Customer Support Role may notify the Content Provider if an error is unrecoverable.

Whenever a new image resource is provided as part of a new or updated Basic Metadata, the Coordinator will perform several actions on the image resource. For each BasicMetadata/LocalizedInfo/ArtReference element:

- Fetch the image from the provided URL
- Scan the image for viruses, and quarantine as necessary

For the set of images provided in BasicMetadata/LocalizedInfo/ArtReference elements

- If necessary image assets are absent, create missing image assets. This SHALL be in accordance with [DMeta] Section 3.2.
- Publish all the image assets at Coordinator-controlled URLs
- Update the BasicMetadata/LocalizedInfo/ArtReference to reflect these new image locations

The Coordinator SHALL NOT process image resources when the ArtReference URL matches an ArtReference element from a MetadataBasicGet() request.

If an update request is made while a previous update is in `pending` status (that is, any required post-processing is still underway), the Coordinator will refuse to process the update request, and respond with an HTTP status code of *404 Not Found*.

Note that it may take significant time to ingest images, especially if some resolutions need to be generated by the Coordinator. The Content Provider can determine status using the GET APIs described below.

Note: MetadataBasicCreate and Update requests will accept RunLength values up to 32 bytes to allow all practical values of run length (especially when describing the run length of an entire series of episodic content). See Table 25.7

6.1.2 MetadataBasicGet, MetadataDigitalGet

6.1.2.1 API Description

These functions are used to retrieve a Basic Metadata or Digital Asset Metadata from the Coordinator.

6.1.2.2 API Details

Path:

```
[BaseURL]/Asset/Metadata/Basic/{ContentID}[?updatenum={UpdateNumber}]

[BaseURL]/Asset/Metadata/Digital/{APID}
```

Methods: GET

Authorized Roles:

```
urn:dece:role[:dece:customersupport]
urn:dece:role:coordinator:customersupport
urn:dece:role:portal[:customersupport]
urn:dece:role:retailer[:customersupport]
urn:dece:role:accessportal[:customersupport]
urn:dece:role:lasp[:customersupport]
urn:dece:role:contentprovider[:customersupport]
urn:dece:role:dsp[:customersupport]
```

Request Parameters:

APID is the Asset Physical identifier for a digital asset

ContentID is the content identifier for a digital asset.

UpdateNumber is an optional query parameter indicating the specific version of the Basic Asset.

> UpdateNumber is only allowed for the Content Provider that created this resource. If a Node provides UpdateNumber, an HTTP status 403 *Forbidden* is returned.

Security Token Subject Scope: None

Opt-in Policy Requirements: None

Request Body: None

Response Body: The Basic or Digital asset metadata (see below for more details on possible responses).

6.1.2.3 Behavior

Requests for Digital Assets simply return the Digital Asset resource. No special response status apply.

The response to a GET query on a Basic Asset metadata varies based on the requester's Role (i.e., whether the requester is the creating Content Provider or another Node). The response will also depend on whether the resource was just created or updated and whether it is being post-processed at the moment of the request.

For newly created Basic Metadata, the table below describes the possible responses based on the requester's Role and the progress of the post-processing:

Request URL Form	Allowed Role(s)	Response		
		post-processing completed	post-processing not completed	post-processing failed (image error)
GET ../{ContentID}	All Roles	HTTP 200 *OK* <BasicAsset> </BasicAsset>	HTTP 404 *Not Found*	HTTP 404 *Not Found*
GET ../{ContentID} ?UpdateNum=1	Creating Content Provider	HTTP *200 OK* <BasicAsset UpdateNum=1> </BasicAsset>	HTTP *200 OK* <BasicAsset UpdateNum=1> <ResourceStatus> …pending</> </BasicAsset>	HTTP *409 Conflict* <ErrorList> Errors </ErrorList>

Table 13: Responses for newly created Basic Assets

Following *n* successful updates on a Basic Asset, and a new update request *m*, the table below describes the possible responses based on the requester's Role and the progress of the post-processing. In the following table 'n' and 'm' represent numbers, such as '0', '1' or '2', where 'm' is greater than 'n'.

Request URL Form	Allowed Role(s)	Response		
		post-processing completed	post-processing not completed	post-processing failed (image error)
GET ../{ContentID}	All Roles	HTTP 200 *OK* <BasicAsset UpdateNum=m> </BasicAsset>	HTTP 200 *OK* <BasicAsset UpdateNum=n> </BasicAsset>	HTTP *200 OK* <BasicAsset UpdateNum=n> </BasicAsset>
GET ../{ContentID} ?UpdateNum=m	Creating Content Provider	HTTP *200 OK* <BasicAsset UpdateNum=m> </BasicAsset>	HTTP *200 OK* <BasicAsset UpdateNum=m> <ResourceStatus> <Current CreatedBy="…" CreationDate="xxx" ModificationDate="yyy" ModifiedBy="…"> <Value>…:pending</> </Current> </ResourceStatus> </BasicAsset>	HTTP *409 Conflict* <ErrorList> Errors </ErrorList>

Table 14: Responses for updated Basic Assets

If an HTTP status code *409 Conflict* is returned, the Content Provider can resubmit a corrected message using the prior updateNum value (the one that failed), or they can increment the updateNum values as they see fit.

6.1.3 MetadataBasicDelete(), MetadataDigitalDelete()

These APIs allow the Content Provider Role to delete basic and digital asset metadata.

6.1.3.1 API Description

These functions are all based on the same template: a single Content identifier (either APID or ContentID) is provided in the URL, and the status of the identified metadata is set to *deleted*.

6.1.3.2 API Details

Path:

```
[BaseURL]/Asset/Metadata/Basic/{ContentID}

[BaseURL]/Asset/Metadata/Digital/{APID}
```

Method: DELETE

Authorized role: `urn:dece:role:contentprovider`

Request Parameters:

APID is an Asset Physical identifier for a digital asset.
ContentID is a content identifier for a digital asset.

Request Body: None

Response Body: None

6.1.3.3 Behavior

If metadata exists for the asset identified by the provided identifier (ContentID or APID), the status of the identified metadata is set to *deleted*.

Asset metadata may only be deleted by the creator of the digital asset or its proxy.

Metadata SHALL NOT be deleted if a reference to it exists (for example, in a bundle).

Furthermore, metadata SHALL NOT be deleted if the asset is referred to in a Rights Token in a User's Rights Locker. In these cases, the metadata MAY be updated, but not deleted.

6.1.4 MetadataBasicList()

6.1.4.1 API Description

This API call returns a list of Basic Assets.

6.1.4.2 API Details

Path:

```
[BaseURL]/Asset/Metadata/Basic/List
```

Method: GET

Authorized role:

```
urn:dece:role:accessportal[:customersupport]
urn:dece:role:contentprovider[:customersupport]
urn:dece:role:coordinator:customersupport
urn:dece:role:dece[:customersupport]
urn:dece:role:lasp:*[:customersupport]
urn:dece:role:portal[:customersupport]
urn:dece:role:retailer[:customersupport]
```

Security Token Subject Scope: None

Opt-in Policy Requirements: None

Request Parameters: None

Request Body: None

Response Body: BasicAssetList

6.1.4.3 Behavior

A collection containing references to <BasicAsset> elements in the system is returned. BasicAssets in deleted status SHALL NOT be returned.

The request SHOULD include the `urn:dece:type:viewfilter:lastmodifieddate` filter to ensure that the response is sorted in chronological order with the most recent elements placed at the beginning of the response. When combined with the `OnOrAfter` query parameter, the response will only contain elements newer that the provided date (see **3.15**).

6.1.5 MetadataDigitalList()

6.1.5.1 API Description

This API call returns a list of Digital Assets.

6.1.5.2 API Details

Path:

```
[BaseURL]/Asset/Metadata/Digital/List
```

Method: GET

Authorized role:

```
urn:dece:role:accessportal[:customersupport]
urn:dece:role:contentprovider[:customersupport]
urn:dece:role:coordinator:customersupport
urn:dece:role:dece[:customersupport]
urn:dece:role:lasp:*[:customersupport]
urn:dece:role:portal[:customersupport]
urn:dece:role:retailer[:customersupport]
```

Security Token Subject Scope: None

Opt-in Policy Requirements: None

Request Parameters: None

Request Body: None

Response Body: DigitalAssetList

6.1.5.3 Behavior

A collection containing references to <DigitalAsset> elements in the system is returned.

The request SHOULD include the `urn:dece:type:viewfilter:lastmodifieddate` filter to ensure that the response is sorted in chronological order with the most recent elements placed at the beginning of the response. When combined with the `OnOrAfter` query parameter, the response will only contain elements newer that the provided date (see **3.15**).

6.2 ID Mapping Functions

A *map* is a reference between the logical identifier for a digital asset (called the asset logical identifier, or ALID), and the physical identifier for a digital asset (called an asset physical identifier, or APID) of a particular file type (such as high-definition, ISO, 3-D, etc.). A *replaced asset* is a digital asset that has been replaced by an equivalent asset. A *recalled asset* is a digital asset that has been replaced with another digital asset, in a case where the original asset must nevertheless be maintained for downloading or streaming because a user has an outstanding to the asset.

6.2.1 MapALIDtoAPIDCreate(),MapALIDtoAPIDUpdate(), AssetMapALIDtoAPIDGet(), AssetMapAPIDtoALIDGet()

6.2.1.1 API Description

These functions create, update, and return the mapping between logical and physical assets.

6.2.1.2 API Details

Path:

```
[BaseURL]/Asset/Map/

[BaseURL]/Asset/Map/{Profile}/{ALID}

[BaseURL]/Asset/Map/{Profile}/{APID}
```

Methods: PUT | POST | GET

Authorized Roles:

For GET operations:

```
urn:dece:role:dece[:customersupport]
urn:dece:role:coordinator:customersupport
urn:dece:role:portal[:customersupport]
urn:dece:role:retailer[:customersupport]
urn:dece:role:accessportal[:customersupport]
urn:dece:role:lasp[:customersupport]
urn:dece:role:contentprovider[:customersupport]
```

For POST and PUT operations:

```
urn:dece:role:contentprovider[:customersupport]
```

Security Token Subject Scope:

`urn:dece:role:account` for GET requests from Devices

`None` for all other roles

Opt-in Policy Requirements: None

Request Parameters:

Profile is a profile from the `AssetProfile-type` enumeration.
APID is an Asset Physical identifier for a digital asset.
ALID is a logical identifier for a digital asset.

Request Body:

A PUT request message conveys the updated asset resource. A POST request message (to `[baseURL]/Asset/Map`) creates a new map, and includes the Asset resource.

Element	Attribute	Definition	Value	Card.
LogicalAsset *or* DigitalAsset		Describes the logical or digital asset, and includes the windowing details for the asset		
LogicalAsset		Mapping from logical to physical, based on profile	`dece:ALIDAsset-type`	1..n
LogicalAssetList		An enumeration of logical asset references associated with an Asset Map (response only)	`dece:LogicalAssetList-type`	0..n

Response Body:

A GET request message returns the Asset resource.

6.2.1.3 Behavior

When a POST operation is used (that is, when a *Create API is invoked), a map is created as long as the ALID is not already in a map for the given profile. When a PUT is used (that is, an *Update), the Coordinator looks for a matching ALID. If there is a match, the map is replaced. If no matching map is found, a map is created. Only the Node who created the asset may update the asset's metadata.

When a GET is used, the Asset is returned.

To determine a map's type, that is, whether the map is to or from an ALID, the provided asset identifier is inspected. An ALID-to-APID map, for example, provides the ALID in the request. Conversely, an APID-to-ALID map provides the APID in the request.

Because an APID may appear in more than one map, more than one ALID may be returned. Whether an ALID is mapped to one or more APIDs, the entire map is returned, because the APID or APIDs required to construct a complete response cannot be known in advance. In most cases, however, a single APIDGroup (containing `active` APIDs only) will be returned as the entire map.

Mapping APIDs to ALIDs will map any `active` APID as follows:

- All APIDGroup elements within the Map element (in the LPMap element) will be returned.
- Any `active` APID or ReplacedAPID will be returned.
- A RecalledAPID SHALL NOT be returned, unless the map does not contain any valid `active` APIDs or ReplacedAPIDs. The feature of returning the RecalledAPID in the case there are no Active or Replaced APIDs provides additional information (i.e., RecalledAPID/ReasonURL) about why the User is not getting the expected Container.

When an APID is mapped, the ALID identified in the ALID element in the LPMap element will be returned.

For requests containing an ALID, if the ALID's status is anything other than `active`, an error indicating that the map was not found will be returned.

6.2.2 LogicalAssetList()

6.2.2.1 API Description

This API call returns a list of Logical Assets.

6.2.2.2 API Details

Path:

```
[BaseURL]/Asset/Map/List
```

Method: GET

Authorized role:

```
urn:dece:role:accessportal[:customersupport]
urn:dece:role:dece[:customersupport]
urn:dece:role:retailer[:customersupport]
urn:dece:role:lasp:*[:customersupport]
urn:dece:role:coordinator:customersupport
urn:dece:role:portal[:customersupport]
urn:dece:role:contentprovider[:customersupport]
```

Security Token Subject Scope: None

Opt-in Policy Requirements: None

Request Parameters: None

Request Body: None

Response Body: LogicalAssetList

6.2.2.3 Behavior

A collection containing <LogicalAssetReference> elements in the system is returned. Logical Assets in deleted status SHALL NOT be returned.

The request SHOULD include the `urn:dece:type:viewfilter:lastmodifieddate` filter to ensure that the response is sorted in chronological order with the most recent elements placed at the beginning of the response. When combined with the `OnOrAfter` query parameter, the response will only contain elements newer that the provided date (see **3.15**).

6.2.3 LogicalAssetDelete()

6.2.3.1 API Description

This API allows the Content Provider and its Customer Support subrole to delete logical asset metadata.

6.2.3.2 API Details

Path:

[BaseURL]/Asset/Map/{Profile}/{ALID}

[BaseURL]/Asset/Map/{ALID}

Method: DELETE

Authorized role: `urn:dece:role:contentprovider:[customersupport]`

Request Parameters:

ALID is a Logical identifier for a digital asset.
Profile is a profile from the `AssetProfile-type` enumeration

Request Body: None

Response Body: None

6.2.3.3 Behavior

The identified logical asset is set to deleted status. When the API variant that does not take the Profile is used, all profiles associated with the Logical asset are deleted.

Assets SHALL NOT be deleted if a reference to it exists (for example, in a bundle OR if the asset is referred to in a Rights Token in a User's Rights Locker).

6.3 Bundle Functions

A *bundle* is a collection of metadata that describes an arbitrary collection of assets. It is analogous to a boxed set sold on store shelves; it may include feature films, audio tracks, electronic books, and other media (such as theatrical trailers, making-of documentaries, slide shows, etc.).

6.3.1 BundleCreate(), BundleUpdate()

These APIs are used to manage the metadata that defines a bundle of digital assets.

6.3.1.1 API Description

BundleCreate is used to create a bundle. BundleUpdate updates the bundle. The BundleUpdate API may be used to change the status of a bundle, which may have the one of several values: `active`, `deleted`, `pending`, or `other`.

The Coordinator SHALL require that active BasicMetadata resources exist for each LogicalAssetReference/ContentID instance and active LogicalAsset resources exist for each LogicalAssetReference/ALID instance.

6.3.1.2 API Details

Path:

```
[BaseURL]/Asset/Bundle

[BaseURL]/Asset/Bundle/{BundleID}
```

Methods: POST | PUT

Authorized Roles:

```
urn:dece:role:retailer[:customersupport]
urn:dece:role:contentprovider[:customersupport]
```

Request Body: The request body is the same for both BundleCreate and BundleUpdate.

Element	Attribute	Definition	Value	Card.
Bundle		Bundle	dece:BundleData-type	

Response Body: None

6.3.1.3 Behavior

When a POST operation is executed (for BundleCreate), a bundle is created. The BundleID is checked for uniqueness. The resource without the BundleID is used.

When a PUT operation is executed (for BundleUpdate), the Coordinator looks for a matching BundleID. If there is a match, the bundle is replaced. The resource which includes the BundleID is used.

Only urn:dece:role:coordinator:customersupport roles and the bundle's creator MAY update a Bundle's status.

6.3.2 BundleGet()

6.3.2.1 API Description

The BundleGet API is used to return bundle data.

6.3.2.2 API Details

Path:

```
[BaseURL]/Asset/Bundle/{BundleID}
```

Method: GET

Authorized Roles:

```
urn:dece:role:dece[:customersupport]
urn:dece:role:coordinator:customersupport
urn:dece:role:portal[:customersupport]
urn:dece:role:retailer[:customersupport]
urn:dece:role:accessportal[:customersupport]
urn:dece:role:lasp[:customersupport]
urn:dece:role:contentprovider[:customersupport]
```

Request Parameters: BundleID is the unique identifier for a bundle.

Request Body: None

Response Body:

Element	Attribute	Definition	Value	Card.
Bundle		Bundle	dece:BundleData-type	

6.3.2.3 Behavior

A bundle (matching the BundleID) is returned.

6.3.3 BundleDelete()

6.3.3.1 API Description

The BundleDelete API is used to set the bundle's status to *deleted*.

6.3.3.2 API Details

Path:

```
[BaseURL]/Asset/Bundle/{BundleID}
```

Method: DELETE

Authorized Roles:

```
urn:dece:role:contentprovider[:customersupport]
urn:dece:role:retailer[:customersupport]
```

Request Parameters: BundleID is the unique identifier for a bundle.

Request Body: None

Response Body: None

6.3.3.3 Behavior

The identified bundle's status is set to `deleted`. BundleDelete is discouraged, since bundles can only be deleted if they have never been referred to in a purchased or rented Rights Token.

Note: This API may be deprecated in future releases of this specification.

6.4 Metadata

Definitions of metadata are part of the md namespace, as defined the *DECE Metadata Specification* [DMeta].

6.4.1 DigitalAsset Definition

Common metadata does not use the APID identifier, so dece:DigitalAssetMetadata-type extends md:DigitalAssetMetadata-type with the following elements to support the APIs.

Element	Attribute	Definition	Value	Card.
DigitalAsset		Physical metadata for an asset	dece:DigitalAssetMetadata-type	

Table 15: DigitalAsset Definition

Not shown in table below, is that there is an xs:choice between {Audio, Video, Subtitle, Image and Interactive} and {ODMP}.

Element	Attribute	Definition	Value	Card.
dece:DigitalAssetMetadata-type		Physical metadata for an asset		
	APID	Asset Physical identifier	md:AssetPhysicalID-type	
	ContentID	Content identifier	md:contentID-type	
	UpdateNum	An increasing integer indicating the version of the resource. If absent, value is assumed to be 1 (one). The first update SHALL be indicated by 2 (two).	xs:positiveInteger	0..1
Audio		Metadata for an Audio Asset	md:DigitalAssetAudioData-type	0..n
Video		Metadata for a Video Asset	md:DigitalAssetVideoData-type	0..n
Subtitle		Metadata for Subtitles	md:DigitalAssetSubtitleData-type	0..n
Image		Metadata for Images	md:DigitalAssetImageData-type	0..n
interactive		Metadata for Interactive Assets	md:DigitalAssetInteractiveData-type	0..n

Element	Attribute	Definition	Value	Card.
ODMP		Description of an ODMP, as per [DDMP]	dece:DigitalAssetODMP-type	
ResourceStatus		Status of the resource. See section 17.2.	dece:ElementStatus-type	0..1

Table 16: DigitalAssetMetadata-type Definition

Element	Attribute	Definition	Value	Card.
DigitalAssetODMP-type		Physical metadata for an asset		
	DMPID	DMP Identifier associated with the described DMP. This identifier SHALL match the APID attribute of the encapsulating DigitalAsset element.	md:AssetPhysicalID	
	Version	ODMP Version as per [SMPTE2053]	xs:nonNegativeInteger	
Presentation		A description of a Presentation within DMP. An instance SHALL exist for each Presentation within the DMP.	dece:DigitalAssetPresentation-type	1..n
Application		A description of a Media Application within DMP. An instance SHALL exist for each Application within the DMP.	dece:DigitalAssetApplication-type	0..n
OtherContainer		Any other Containers within the DMP.	md:ContainerMetadata-type	0..n

Table 17: DigitalAssetODMP Definition

Element	Attribute	Definition	Value	Card.
DigitalAssetPresentation-type		Physical metadata for an asset		
	PresentationID	Presentation ID associated with this Presentation	md:AssetLogicalID-type	
	Version	Presentation Version as per [SMPTE2053]	xs:nonNegativeInteger	
MediaProfile		The DECE Media Profile associated with this Presentation	dece:AssetProfile-type	
APID		APID for each DCC associated with DMP, whether or not that DCC is in the ODMP.	md:AssetPhysicalID-type	1..n

Element	Attribute	Definition	Value	Card.
	isPresent	Indicates whether the DCC associated with this APID is present in the ODMP.	`xs:boolean`	
	forceDownload	Indicates whether the download manager SHALL download the DCC associated with this APID immediately following ODMP download.	`xs:boolean`	0..1

Table 18: DigitalAssetPresentation Definition

Element	Attribute	Definition	Value	Card.
DigitalAssetApplication-type		Physical metadata for an asset		
	Version	Media Application Version as per [SMPTE2053]	`xs:nonNegativeInteger`	
APID		APID for each Media Application element associated with DMP, whether or not that Media Application element is in the ODMP.	`md:AssetPhysicalID-type`	1..n
	isPresent	Indicates whether the Media Application element associated with this APID is present in the ODMP.	`xs:boolean`	
	forceDownload	Indicates whether the download manager SHALL download the DCC associated with this APID immediately following ODMP download.	`xs:boolean`	0..1
PresentationID		Each Presentation associated with this Media Application. If absent, Media Application is associated with all Presentations in the DMP. If present, the Coordinator SHALL verify that this identifier matches a presentation described in the same ODMP element.	`md:AssetLogicalID-type`	0..n

Table 19: DigitalAssetApplication Definition

6.4.1.1 Digital Asset Status Transitions

The possible Status values are: `active`, `pending` and `deleted`.

6.4.2 BasicAsset Definition

The BasicAsset element extends the `md:BasicMetadata-type`.

Element	Attribute	Definition	Value	Card.
BasicAsset			`dece:AssetMDBasic-type`	
BasicData		Basic Metadata	`md:MDBasicDataType`	
ResourceStatus		Status of the resource. See section 17.2.	`dece:ElementStatus-type`	0..1

Table 20: BasicAsset Definition

6.4.2.1 Basic Asset Status Transitions

The possible Status values are: `active`, `pending`, `deleted`, and `other`.

6.4.3 DigitalAssetList Definition

The DigitalAssetList element is a list of DigitalAsset element references.

Element	Attribute	Definition	Value	Card.
DigitalAssetList			`dece: DigitalAssetList-type`	
	ViewFilterAttr	Response filtering information, see section **17.5**	`dece:ViewFilterAttr-type`	
DigitalAssetReference			`dece: DigitalAssetReference-type`	0..n

Table 21: DigitalAssetList Definition

Element	Attribute	Definition	Value	Card.
DigitalAssetReference			dece:DigitalAssetReference-type	
	APID	Identifier of a DigitalAsset element. The value of this identifier is the APID attribute of the DigitalAsset.	**6.4.3.1** md:AssetPhysicalID-type	
	CurrentStatus	The same value as the of the resource's //ResourceStatus/Current/Value element (typically active, pending or deleted).	dece:StatusValue-type	
	DatedElementAttrGroup		dece:DatedElementAttrGroup-type	

Table 22: DigitialAssetReference Definition

6.4.4 BasicAssetList Definition

The BasicAssetList element is a list of BasicAsset element references.

Element	Attribute	Definition	Value	Card.
BasicAssetList			dece:BasicAssetList-type	
	ViewFilterAttr	Response filtering information, see section **17.5**	dece:ViewFilterAttr-type	
BasicAssetReference			dece: BasicAssetReference-type	0..n

Table 23: BasicAssetList Definition

Element	Attribute	Definition	Value	Card.
BasicAssetReference			dece:BasicAssetReference-type	
	ContentID	Identifier of a BasicAsset element. The value of this identifier is the ContentID attribute of the BasicAsset.	md:ContentID-type	

Element	Attribute	Definition	Value	Card.
	CurrentStatus	The same value as the of the resource's `//ResourceStatus/Current/Value` element (typically `active`, `pending` or `deleted`).	`dece:StatusValue-type`	
	DatedElement AttrGroup		`dece:DatedElement AttrGroup-type`	

Table 24: BasicAssetReference Definition

6.5 Mapping Data

6.5.1 Mapping Logical Assets to Content IDs

Every Logical Asset SHALL map to a single ContentID. Every ContentID MAY map to more than one Logical Asset.

6.5.1.1 LogicalAssetReference Definition

Element	Attribute	Definition	Value	Card.
LogicalAsset Reference		Logical Asset to Content identifier map	`dece:LogicalAssetReference-type`	
ALID		Asset Logical identifier	`md:AssetLogicalID-type`	
ContentID		Content identifier associated with the Logical Asset	`dece:ContentID-type`	

Table 25: LogicalAssetReference Definition

6.5.2 Mapping Logical to Digital Assets

A Logical Identifier maps to one or more Digital Assets for each available Profile.

6.5.2.1 LogicalAsset Definition

Mappings may be from an ALID to one or more APIDs. Maps are defined within one or more AssetFulfillmentGroups, identified by a FulfillmentGroupID and carry a serialized version identifier.

Distinct AssetFulfillmentGroup instances SHALL exist for DCCs of different versions as indicated by LatestContainerVersion. An AssetFulfillmentGroup MAY contain DCCs with different versions. LatestContainerVersion SHALL reflect the most current DCC version.

Distinct AssetFulfillmentGroup instances SHALL exist for DMPs as indicated by LatestContainerVersion.

An AssetFulfillmentGroup SHALL NOT contain a DigitalAssetGroups that references a DCC and a DigitalAssetGropus than references a DMP.

APIDs are grouped in DigitalAssetGroup elements. If no APIDs have been replaced or recalled (as described in DigitalAssetGroup-type Definition, below), then there should be only one group. If APIDs have been replaced or recalled, the digital asset grouping indicates which specific APIDs replace which specific APIDs. The grouping (as opposed to an ungrouped list) provides information that allows Nodes to know which specific replacements need to be provided.

Logical Assets can include a description of one or more restrictions on the Physical Assets, which inform DSPs and LASPs when and where they cannot Download, Stream, License or Fulfill Discrete Media. The Coordinator SHALL NOT enforce these restrictions. See [DSystem] 7.4.5.

APIDs can map to more than one ALID, but this mapping is not supported directly; it is handled by creating several APID-to-ALID maps.

Element	Attribute	Definition	Value	Card.
LogicalAsset		Asset mapping from logical to physical	`dece:ALIDAsset-type`	
	Version	version number, increasing monotonically with each update	`xs:int`	0..1
	ALID	Asset Logical identifier for Asset	`md:AssetLogicalID-type`	
	MediaProfile	Media Profile for Asset	`dece:AssetProfile-type`	
	ContentID		`md:ContentID-type`	
	Assent StreamAllowed	Indicates whether Streaming is enabled for LASPs without need of licensing from the Content Provider	`xs:boolean`	
	AssentStream Loc	The location of the AssentStream content. This value SHALL NOT be set unless AssentStreamAllowed is set to TRUE.	`xs:anyURI`	0..1
AssetFulfillmentGroup		A collection of DigitalAssetGroups	`dece:AssetFulfillment Group-type`	1..n
AssetRestriction		Regional and temporal Information about restrictions on Download, Licensing, Streaming and Discrete Media Fulfillment.	`dece:AssetRestriction-type`	0..n

Table 26: LogicalAsset

6.5.2.2 LogicalAssetList Definition

The LogicalAssetList element is a list of LogicalAssetReference elements.

Element	Attribute	Definition	Value	Card.
LogicalAssetList			dece:LogicalAssetList-type	
	ViewFilterAttr	Response filtering information, see section **17.5**	dece:ViewFilterAttr-type	
LogicalAssetReference		LogicalAssetReference element, see section **6.5.1**	dece:LogicalAssetReference-type	0..n

Table 27: LogicalAssetList Definition

Element	Attribute	Definition	Value	Card.
LogicalAssetReference			dece:LogicalAssetReference-type	
	ALID	Identifier of a LogicalAsset element. The value of this identifier is the ALID attribute of the logical asset.	md:AssetLogicalID-type	
	ContentID	Identifier of a BasicAsset element. The value of this identifier is the ContentID attribute of the BasicAsset.	md:ContentID-type	
	CurrentStatus	The same value as the of the resource's //ResourceStatus/Current/Value element (typically active, pending or deleted).	dece:StatusValue-type	
	DatedElementAttrGroup		dece:DatedElementAttrGroup-type	

Table 28: LogicalAssetReference Definition

6.5.2.3 APID Grouping Scenarios

This section intentionally left blank.

6.5.2.4 AssetFulfillmentGroup Definition

Element	Attribute	Definition	Value	Card.
AssetFulfillmentGroup			`dece:AssetFulfillmentGroup-type`	
	FulfillmentGroupID	The unique identifier for a fulfillment group	`xs:string`	0..1
	LatestContainerVersion	The highest version of all Container versions (no validation is required)	`xs:string`	0..1
	LatestSpecVersion	The version of the spec describing elements herein.	`xs:string`	0..1
DigitalAssetGroup		Map details	`dece:DigitalAssetGroup-type`	1…n

Table 29: AssetFulfillmentGroup

LatestContainerVersion SHALL be included and contains the highest version of each DCC referenced within the group. The version of the DCC is the major_brand in a Container's 'ftyp' Box as defined in [DMedia], Section 2.3.1.

LatestSpecVersion SHALL be encoded as following

- 'DMedia-v1.0.7' or absent when no DigitalAssetGroup instances contain DCCs only compliant with [DMedia] versions later than 1.0.7
- 'DMedia-v1.1' When DigitalAssetGroup instances contain DCCs only compliant with [DMedia] versions later than 1.0.6. It is anticipated that in the future, newer DCCs might be assigned a later version.
- 'DDMP-v1.0' When DigitalAssetGroups contain DMPs. At some point, newer DMPs might be assigned a later version.

6.5.2.5 DigitalAssetGroup Definition

A DigitalAssetGroup is a list of APIDs with identification of their state (`active`, `replaced`, or `recalled`).

The meaning of APID state identification is as follows:

- APIDs in an ActiveAPID element are `active` and current. DCCs associated with APIDs in a DigitalAssetGroup with CanDownload='true' SHALL be downloaded and licensed in accordance

with applicable policies. Content associated with other APIDs SHOULD be streamed or otherwise fulfilled in accordance with DigitalAssetGroup attributes and applicable policies.

- APIDs in the ReplacedAPID element have been replaced by the APIDs in the ActiveAPID element. That is, ReplacedAPID elements refer to Containers that are obsolete but still may be downloaded, licensed, streamed or otherwise fulfilled in accordance with DigitalAssetGroup attributes and applicable policies. APIDs in the ActiveAPID element are preferable. ReplacedAPIDs SHOULD NOT be downloaded, licensed, streamed or otherwise fulfilled. An APID SHALL NOT be placed in ReplacedAPID unless the corresponding APID has been placed in ActiveAPID.

- APIDs in RecalledAPIDs SHALL NOT be downloaded, licensed, streamed or otherwise fulfilled, with the exception that the RecalledAPID MAY be licensed if the LicensingAllowed attribute is set to 'true'. Normally, there will always be at least one ActiveAPID. However, for the contingency that an APID is recalled and there is no replacement, there may be one or more RecalledAPID elements. When the APID is a single audio track, LicensingAllowed attribute MAY be ignored. Note that this constraint is included because audio tracks are currently licensed along with video tracks, and in a DMP it is impractical to determine whether the recalled audio track is present. Expected behavior on recalled audio tracks is generally consistent with LicensingAllowed='true'. It is recommended that LicensingAllowed be set to 'true' to ensure consistency.

Exactly one of DiscreteMediaFulfillmentMethods, CanDownload and CanStream SHALL be included. The intended use of Assets in the AssetGroup is designated by the DiscreteMediaFulfllmentMethods, CanDownload and CanStream attributes. A downloadable DCC is indicated by CanDownload. If an Asset is suitable for streaming (e.g., a CFF Container with streamable media), CanStream is set to 'true'. DiscreteMediaFulfillmentMethods signals Assets suitable for Discrete Media Fulfillment; for example, `urn:dece:type:discretemediaformat:dvd:cssrecordable` for a burnable DVD.

APIDs in a DigitalAssetGroup SHALL correspond with acceptable uses indicated by the CanDownload, CanStream and DiscreteMediaFulfillmentMethods attributes. In particular, only DCCs can be included when CanDownload is set to 'true'.

No more than one instance of a DigitalAssetGroup within an AssetFulfillmentGroup SHALL have the same attribute value. For example, there cannot be more than one DigitalAssetGroup with CanDownload='true'.

Note that an APID may exist in more than one DigitalAssetGroup, and these APIDs might be classified differently. For example, an APID whose DCC is found to be noncompliant might be in a RecalledAPID element in a DigitalAssetGroup with the attribute CanDownload='true'; while that same APID was in a DigitalAssetGroup of with attribute CanStream='true' in the ActiveAPID element.

APIDs usage within an AssetFulfillmentGroup SHALL NOT conflict. For example, an APID cannot be in more than one of ActiveAPID, ReplacedAPID and RecalledAPID elements.

A DigitalAssetGroup SHALL have exactly one of DiscreteMediaFulfillmentMethods, CanDownload, CanStream and IsDMP.

Element	Attribute	Definition	Value	Card.
DigitalAssetGroup		Assets defined as a part of the Logical Asset, expressed as a map	`dece:DigitalAssetGroup-type`	
	DiscreteMediaFulfillmentMethods	One Discrete Media Fulfillment usage for APIDs in this map. It identifies which methods the APID can fulfill.	`xs:NMTOKENS`	0..1
	CanDownload	It is acceptable to download a Container associated with the APID if the ActiveAPID is not yet available. If FALSE or absent, the Container SHALL NOT be downloaded. The purpose of this attribute is to describe possible usage of the container (format). It does not express any window-related authorization.	`xs:boolean`	0..1
	CanStream	It is acceptable to stream a Container associated with the APID if the ActiveAPID is not yet available. If FALSE or absent, the Container SHOULD NOT be streamed. The purpose of this attribute is to describe possible usage of the container (format). It does not express any window-related authorization.	`xs:boolean`	0..1
	IsDMP	Indicates that this DigitalAssetGroup contains APIDs referring to DMPs.	`xs:boolean`	
ActiveAPID		Active Asset Physical identifier for Physical Assets associated with ALID	`md:AssetPhysicalID-type`	0..n
ReplacedAPID		Replaced Asset Physical identifier for Physical Assets associated with ALID	`md:AssetPhysicalID-type`	0..n
RecalledAPID		Recalled Asset Physical identifier for Physical Assets associated with ALID	`dece:RecalledAPID-type`	0..n

Table 30: DigitalAssetGroup Definition

6.5.2.6 RecalledAPID Definition

Element	Attribute	Definition	Value	Card.
RecalledAPID			`dece:RecalledAPID-type`	
	ReasonURL	An attribute of RecalledAPID, which contains a Content Provider-supplied URL to a page explaining why the request for this asset cannot be fulfilled.	`xs:anyURI`	0..1
	LicensingAllowed	Indicates that an already downloaded Container can be licensed. If 'true', licensing is allowed for the associated APID. If 'false' or absent licensing is not allowed. This only applies to DigitalAssetGroups with CanDownload set to 'true'. Behavior associated with this attribute is undefined if the APID refers to a DCC containing a single audio track.	`xs:boolean` `default 'false'`	0..1

Table 31: RecalledAPID Definition

6.5.2.7 AssetRestriction Definition

An Asset Restriction is a period of time in a particular region during which policies are applied with respect to downloading, streaming or Discrete Media Fulfillment. This is the mechanism for implementing blackout windows. Region, Start and End describe the location and timeframe of the restriction. Asset release is controlled by the restriction.

Restrictions are one of the following:

Restriction	Definition
urn:dece:contentrestriction:nodownload	Download not allowed (all forms)
urn:dece:contentrestriction:nodownload:legacy	Download not allowed for legacy devices
urn:dece:contentrestriction:nodownload:dcc	Download not allowed for DCCs
urn:dece:contentrestriction:nolicensing	Licensing not allowed
urn:dece:contentrestriction:nostream	Streaming not allowed
urn:dece:contentrestriction:nodiscretemedia	Discrete Media Fulfillment not allowed (all types)

Restriction	Definition
urn:dece:contentrestriction:nodiscretemedia:packaged	Discrete Media Fulfillment not allowed for packaged media
urn:dece:contentrestriction:nodiscretemedia:packaged:hd	Discrete Media Fulfillment not allowed for packaged HD
urn:dece:contentrestriction:nodiscretemedia:css	Discrete Media Fulfillment not allowed for CSS burnable
urn:dece:contentrestriction:nodiscretemedia:cprmsd	Discrete Media Fulfillment not allowed for CPRM SD

Following is the element definition.

Element	Attribute	Definition	Value	Card.
AssetRestriction			`dece:AssetRestriction-type`	
Region		Region to which the window applies. If absent, then restrictions are world-wide.	`md:Region-type`	0..n
Start		Date and time at which restriction starts. If absent, the start period is immediate. Time in UTC.	`xs:dateTime`	0..1
End		Date and time at which restriction ends. If absent, there is not end period; that is, all time following `Start`. Time in UTC.	`xs:dateTime`	0..1
Restriction		Policies define what is not allowed.	`xs:anyURI`	1..n

Table 32: AssetRestriction Definition

6.5.3 MediaProfile Values

The simple type `AssetProfile-type` defines the set of MediaProfile values used within DECE. The base type is `xs:anyURI`, and the values are described in the following table.

MediaProfile Value	Description
`urn:dece:type:mediaprofile:pd`	Portable Definition
`urn:dece:type:mediaprofile:sd`	Standard Definition
`urn:dece:type:mediaprofile:hd`	High Definition
`urn:dece:type:mediaprofile:uhd`	Ultra High Definition

Table 33: MediaProfile Values

6.6 Bundle Data

A bundle consist of a list of ContentID-to-ALID maps (`dece:BundleData-type`) and optional information to provide logical grouping to the Bundle in the form of composite resources (`md:CompObj-type`). In its simplest form, the Bundle is one or more ContentID-to-ALID maps along with a BundleID and a text description. The semantics of the bundle consists of the rights associated with the ALID and described by metadata. The Bundle refers to Rights Tokens, so there is no need to

include Profile information in the Bundle: that information exists in a Rights Token. A Bundle uses the Composite Resource mechanism (`md:CompObj-type`, as defined in [DCMeta]) to create a tree-structured collection of content identifiers, with optional descriptions and metadata.

6.6.1 Bundle Definition

The Bundle structure is described in the following table.

Note: This specification introduces a new element to the bundle structure (`ContentID`), that allows Content Publishers and Retailers to describe basic metadata for the Bundle. The version attribute of the schema has been incremented, however the namespace of the schema has not changed. Coordinator clients should ensure that they have updated their schema accordingly, and gracefully handle the presence of this element.

Element	Attribute	Definition	Value	Card.
Bundle			`dece:BundleData-type`	
	BundleID	Unique identifier for the Bundle	`dece:EntityID-type`	
ContentID		Identifier of a BasicAsset element. Allows Content Publishers and Retailers to reference Basic metadata for the Bundle..	`dece:EntityID-type`	0..1
DisplayName		A localizable string used for display purposes	`dece:LocalizedStringAbstract-type`	1…n
LogicalAsset Reference		A set of Logical Asset references	`dece:LogicalAsset Reference-type`	1…n
CompObj		Information about each asset component	`md:CompObj-type`	0..1
Resource Status		Status of element	`dece:ElementStatus-type`	0..1

Table 34: Bundle Definition

6.6.2 LogicalAssetReference Definition

The LogicalAssetReference is used to map ALID to ContentID

Element	Attribute	Definition	Value	Card.
LogicalAssetReference			`dece:LogicalAsset Reference-type`	

Element	Attribute	Definition	Value	Card.
ContentID		The unique identifier for a basic asset in the Bundle	`md:ContentID-type`	
ALID		Asset logical identifier	`md:AssetLogicalID-type`	

Table 35: LogicalAssetReference Definition

6.6.3 Bundle Status Transitions

The possible Status values are: `active`, `pending`, `deleted`, and `other`.

7 Rights

The Coordinator is an entitlement registry service. Its primary resources are entitlements expressed as Rights, which are an indication to API Clients that Users have acquired the rights to the digital assets identified in a Rights Token.

7.1 Rights Functions

Rights Lockers and Rights Tokens are `active` only if their status (ResourceStatus/Current) is set to `urn:dece:type:status:active`. Rights Lockers and Rights Tokens are accessible to API Clients according to the "API Invocation by Role" table in Appendix A which also specifies which representation of the resource is provided in a response.

The Coordinator SHALL NOT allow the number of DiscreteMediaRights within a given Rights Token to exceed the number determined by the Ecosystem parameter `DISCRETE_MEDIA_LIMIT`.

7.1.1 Rights Token Visibility

In general, the retailer that created a Rights Token (called the *issuer*) can access a Rights Token that it issued, regardless of the status of the Rights Token. For Rights Tokens issued by other retailers, however, a retailer can view only the Rights Tokens whose status is set to `active`.

The following table lists the Roles, the status of the Rights Tokens that are visible to the Role, and whether the Role may read (R), write (W), or read and write (RW) the values of Rights Token properties. It also describes the visibility of the Rights Tokens for the listed roles.

Role	Rights Token Status	R/W	Visibility
retailer:issuer	All	RW	All Rights Tokens created by the issuer are visible
retailer:issuer:customersupport	All	RW	All Rights Tokens created by the issuer are visible
coordinator:customersupport	All	R	All Rights Tokens in the Rights Locker are visible, regardless of status or issuer
Web Portal	`Active`, `Pending`	R	Rights Tokens with the specified statuses are visible
All other roles	`Active`, `Pending`	R	Only `active` and `pending` Rights Tokens are visible

Table 36: Rights Token Visibility by Role

7.1.2 RightsTokenCreate()

7.1.2.1 API Description

The RightsTokenCreate API is used to add a Rights Token to a Rights Locker.

7.1.2.2 API Details

Path:

```
[BaseURL]/Account/{AccountID}/RightsToken
```

Method: POST

Authorized Roles:

`urn:dece:role:retailer[:customersupport]`

Security Token Subject Scope: `urn:dece:role:user`

Opt-in Policy Requirements: None

Request Body:

Element	Attribute	Definition	Value	Card.
RightsTokenData		A fully populated Rights Token. All required information SHALL be included in the request.	`dece:RightsTokenData-type`	1

Response Body: None

7.1.2.3 Behavior

This creates a Right for a given Logical Asset Media Profile(s) for a given Account. The Rights token is associated with the Account, the User, and the Retailer.

The Node SHALL NOT set the value of the `RightsTokenID` element, which is established by the Coordinator.

RightsTokenCreate() MAY be invoked for an Account with `pending` status.

If no error conditions occur, the Coordinator SHALL respond with an HTTP 201 status code (*Created*) and a Location header containing the URL of the created resource.

Once created, the Rights token SHALL NOT be physically deleted, only flagged in the ResourceStatus element with a `<Current>` Status value of 'deleted'. Modifications to the Rights token SHALL be noted in the History element of the ResourceStatus Element.

Nodes implementing this API interface SHOULD NOT conclude any commerce transactions (if any), until a successful Coordinator response is obtained, as a token creation may fail due to Parental Controls or other factors.

Rights are associated with content by their identifiers ContentID and ALID. These identifiers SHALL be verified by the Coordinator when the RightsToken is created. The corresponding LogicalAsset and BasicAsset properties SHALL also be validated by the Coordinator when the RightsToken is created.

Nodes SHALL create all RightsToken media profiles which apply. For example, a RightsToken providing the HD media profile must also include the media profile for SD. [DSystem] defines which media profiles are required for a given purchased media profile.

Nodes SHALL create all necessary RightsTokens when creating Bundles or other composite content.

The DiscreteMediaRightsRemaining SHALL NOT be included with the creation of a Rights Token. This field is used by the Coordinator for response values only, and is calculated based on the available DiscreteMediaRightsTokens as defined in section 16.

The Coordinator SHALL require that:

- The ALID attribute value is a valid identifier, with a corresponding LogicalAsset resource in `active` status,
- The ContentID attribute value is a valid identifier with a corresponding BasicMetadata resource in `active` status,
- When SoldAs is present
    - All ContentID elements in the Rights Token's SoldAs element contain a valid identifier with a corresponding BasicAsset resource in `active` status,
    - The identifier in the RightsTokenData/@ContentID attribute exists in one instance of SoldAs/ContentID list, or within the Bundle referenced by SoldAs/BundleID
    - If SoldAs contains a BundleID:
        - The BundleID is a valid identifier and corresponds to a Bundle resource in `active` status (the 'referenced Bundle'),

- RightsTokenData/@ALID and RightsTokenData/@ContentID attributes correspond with ALID and ContentID in one instance of a LogicalAssetReference element in the referenced Bundle.

Upon successful creation, the Coordinator SHALL set the RightToken status to `active`.

7.1.3 RightsTokenDelete()

7.1.3.1 API Description

This API changes a rights token to an inactive state. It does not actually remove the rights token, but sets the status element to 'deleted'.

7.1.3.2 API Details

Path:

```
[BaseURL]/Account/{AccountID}/RightsToken/{RightsTokenID}

[BaseURL]/RightsToken/{RightsTokenID}
```

Method: DELETE

Authorized Roles:

```
urn:dece:role:retailer[:customersupport]
urn:dece:role:portal[:customersupport]
urn:dece:role:dece:customersupport
urn:dece:role:coordinator:customersupport
```

Security Token Subject Scope: `urn:dece:role:user` except for the `urn:dece:role:retailer[:customersupport]`role, where use of a Delegation Security Token is optional.

Opt-in Policy Requirements: None

Request Parameters:

RightsTokenID is the unique identifier for a rights token
AccountID is the unique identifier for an Account

Request Body: None

Response Body: None

7.1.3.3 Behavior

ResourceStatus is updated to reflect the deletion of the right. Specifically, the status value of the <Current> element within the ResourceStatus element is set to *deleted*. The prior <Current> Status gets moved to the ResourceStatus/History.

When the deletion is the result of a user request, the Node SHALL include a valid Delegation Security Token. Whenever a Delegation Security Token is included in the request, the Coordinator SHALL send an email notification to all the Users in the Account.

The URI path that excludes the {AccountID} parameter is exclusively for the use of Retailer (Issuer only) and its Customer Support subrole. Attempts to delete Rights Tokens created by other retailers shall result in a 403 Forbidden HTTP Status response.

Nodes SHALL NOT include a Delegation Security Token when using the URI path that excludes the {AccountID} parameter. If a Delegation Security Token is included in this case, it will be ignored by the Coordinator. If a Node has a valid, active Delegation Security Token, it SHOULD use the API form that includes the {AccountID} parameter, unless the Node's intent is to correct a Right.

RightsToken deletion without AccountID and Delegation Security Token is reserved for cases when a Retailer needs to correct or update an existing Right. For example, removing an incorrectly created Right, updating Media Profile or StreamWebLoc values, removing an accidentally duplicated Right or a Right with a reversed transaction (refund, credit card decline etc.). No email notification is sent when Rights are deleted using the URI path that excludes the {AccountID} parameter.

7.1.4 RightsTokenGet()

This function is used for the retrieval of a Rights token given its identifier. The following rules are enforced:

Role[4]	Issuer	Security Context	Applicable Policies	LockerView AllConsent	RightsToken	Notes
DECE	N/A	Account	N/A	Always TRUE	RightsTokenFull	
DECE: CS	N/A	Account	N/A	Always TRUE	RightsTokenFull	3, 7
Coordinator: CS	N/A	Account	N/A	Always TRUE	RightsTokenFull	3, 7
Web Portal	N/A	User	ParentalControl (BlockUnratedContent, RatingPolicy), AllowAdult	Always TRUE	RightsTokenFull	1
Web Portal CS	N/A	Account	N/A	Always TRUE	RightsTokenFull	1

Role[4]	Issuer	Security Context	Applicable Policies	LockerView AllConsent	RightsToken	Notes
Retailer	Y	User or None[6]	ParentalControl (BlockUnratedContent, RatingPolicy), AllowAdult	N/A	RightsTokenInfo or RightsTokenFull (when a Security Token is not included)	1, 2, 7
Retailer	N	User	LockerViewAllConsent, ParentalControl (BlockUnratedContent, RatingPolicy), AllowAdult	FALSE	RightsToken not available	1
				TRUE	RightsTokenInfo	
Retailer: CS	Y	Account or None[6]	N/A	N/A	RightsTokenInfo or RightsTokenFull (when a Security Token is not included)	2, 3, 7
Retailer: CS	N	Account	LockerViewAllConsent	FALSE	RightsToken not available	3
				TRUE	RightsTokenInfo	
Access Portal	N/A	User	LockerViewAllConsent, ParentalControl (BlockUnratedContent, RatingPolicy), AllowAdult	FALSE	RightsToken not available	1
				TRUE	RightsTokenInfo	
Access Portal: CS	N/A	Account	LockerViewAllConsent	FALSE	RightsToken not available	3
				TRUE	RightsTokenInfo	
Linked LASP	N/A	Account	N/A	Always TRUE	RightsTokenBasic	1
Linked LASP CS	N/A	Account	N/A	Always TRUE	RightsTokenBasic	3
Dynamic LASP	N/A	User	ParentalControl (BlockUnratedContent, RatingPolicy), AllowAdult	Always TRUE	RightsTokenBasic	1
Dynamic LASP CS	N/A	Account	N/A	FALSE	RightsTokenBasic	3
				TRUE	RightsTokenInfo	
				TRUE	RightsTokenInfo	

[1] Requires a valid Security Token issued to entity

[2] Rights Tokens are returned regardless of Rights Token Status

[3] Customer Support security context will only be at the Account level
(using one of the Security Tokens issued to the corresponding entity)

[4] Relative URN based in `urn:dece:role:*`

[5] The following elements in PurchaseInfo SHALL NOT be included in the response: NodeID, RetailerTransaction, and TransactionType

[6] Issuing Retailers MAY omit the Security Token for RightsTokenGET, RightsTokenUpdate and RightsLockerDataGet APIs.

[7] Allowed to invoke the URI endpoint that excludes the {AccountID} parameter.

Table 37: Rights Token Access by Role

7.1.4.1 API Description

The retrieval of the Rights token is constrained by the rights allowed to the retailer and the user who is making the request.

7.1.4.2 API Details

Path:

```
[BaseURL]/Account/{AccountID}/RightsToken/{RightsTokenID}

[BaseURL]/RightsToken/{RightsTokenID}
```

Method: GET

Authorized Roles:

```
urn:dece:role:dece[:customersupport]
urn:dece:role:coordinator:customersupport
urn:dece:role:portal[:customersupport]
urn:dece:role:retailer[:customersupport]
urn:dece:role:accessportal[:customersupport]
urn:dece:role:lasp[:customersupport]
```

Security Token Subject Scope:

`urn:dece:role:user`, or None (see below)

Ignored, if submitted for the endpoint form without AccountID

Opt-in Policy Requirements:

```
urn:dece:type:policy:LockerViewAllConsent
urn:dece:type:policy:ParentalControl:*
```

Request Parameters: RightsTokenID is the unique identifier for a Rights Token

Request Body: None

Response Body: RightsToken

RightsToken SHALL contain one of the following: `RightsTokenBasic`, `RightsTokenInfo`, `RightsTokenData` or `RightsTokenFull`. For more information about these objects, see section 7.2.

7.1.4.3 Behavior

The request for a Rights Token is made on behalf of a User. The Rights Token data is returned in accordance with Table 37: Rights Token Access by Role.

Nodes SHALL NOT include a Delegation Security Token when using the URI path that excludes the {AccountID} parameter. If invoked by a Retailer or its Customer Support subrole, only Rights Tokens created by the invoking Retailer Node or its Customer Support specialization are accessible. Attempts to access Rights Tokens created by other retailers shall result in a 403 Forbidden HTTP Status response.

If a Node has a valid, active Delegation Security Token, it SHOULD use the API form that includes the {AccountID} parameter.

It is strongly recommended that Nodes use the ETag in subsequent conditional requests (using If-Match or If-None-Match) as specified in section 3.5.

7.1.5 RightsTokenDataGet()

7.1.5.1 API Description

This method allows for the retrieval of a list of Right tokens selected by ALID. The list may contain a single element.

7.1.5.2 API Details

Path:

For the list of Rights tokens based on an ALID:

```
[BaseURL]/Account/{AccountID}/RightsToken/ByMedia/{ALID}
```

Method: GET

Authorized Roles:

```
urn:dece:role:dece[:customersupport]
urn:dece:role:coordinator:customersupport
urn:dece:role:portal[:customersupport]
urn:dece:role:retailer[:customersupport]
urn:dece:role:accessportal[:customersupport]
urn:dece:role:lasp[:customersupport]
```

Security Token Subject Scope:
`urn:dece:role:user.`

Opt-in Policy Requirements:
In accordance with Table 37: Rights Token Access by Role.

Request Parameters:

`ALID` is the logical identifier for a digital asset.

Response Body:

A list of one or more Rights Tokens.

7.1.5.3 Behavior

A request is made for a list of Rights Tokens. This request is made on behalf of a User.

The Rights Token data is returned in accordance with Table 37: Rights Token Access by Role

When requesting by ALID, Rights tokens that contain the ALID for that Account are returned. There may be zero or more.

Limited data is returned on Rights tokens that were created by Retailers other than the requestor.

7.1.6 RightsLockerDataGet()

RightsLockerDataGet() returns the list of all the Rights tokens. This operation can be tuned via a request parameter to return actual Rights tokens with or without metadata or references to those tokens.

7.1.6.1 API Description

The Rights Locker data structure, namely RightsLockerData-type information is returned.

7.1.6.2 API Details

Path:

```
[BaseURL]/Account/{AccountID}/RightsToken/List[?response={responseType}]
```

Method: GET

Authorized Roles:

```
urn:dece:role:dece[:customersupport]
urn:dece:role:coordinator:customersupport
urn:dece:role:portal[:customersupport]
urn:dece:role:retailer[:customersupport]
urn:dece:role:accessportal[:customersupport]
urn:dece:role:lasp[:customersupport]
```

Security Token Subject Scope:

`urn:dece:role:user` or None

Opt-in Policy Requirements:

```
urn:dece:type:policy:LockerViewAllConsent
```

Request Parameters: `response (optional)`

By default, that is if no request parameter is provided, the operation returns a list of Rights Token references. When present, the `response` parameter can be set to one of the following values:

token – return the actual Rights tokens
reference – return references to the Rights tokens (`RightsTokenReference-type`) - (default setting)
metadata – return the Rights tokens metadata (`RightsTokenDetails-type`)
download – return only the RightsTokenLocation portion of the Rights Token (<xs:element name="RightsTokenLocation" type="dece:RightsTokenLocation-type"/>)

For example:

```
[BaseURL]/Account/{AccountID}/RightsToken/List?response=reference
```

will instruct the Coordinator to only return a list of references to the Rights Tokens.

In addition, appropriate filter parameters defined in section 3.15 may be included.

Request Body: None

Response Body:

Element	Attribute	Definition	Value	Card.
RightsTokenList			`dece:RightsLockerData-type`	

7.1.6.3 Behavior

The request for Rights Locker data is made on behalf of a User.

In order to prevent operational issues such as timeouts, the Coordinator returns a maximum of DCOORD_PAGINATION_THRESHOLD Rights Tokens (or references) in a single response. Requests by users with lockers that have more than DCOORD_PAGINATION_THRESHOLD Rights Tokens will return the first DCOORD_PAGINATION_THRESHOLD tokens and include the ViewFilterAttr group attributes (see section 17.5) indicating that additional Rights Tokens are available. See Section 3.16 for information on retrieving resources in groups.

RightsDataLockerGet responses may include a `true` value in the `FilterMoreAvailable` attribute indicating a partial Rights Locker response.

The aforementioned request pagination controls apply whether or not a Delegation Security Token was included in the request.

Similarly, the `response` parameters defined in 7.1.6.2 MAY be used whether or not a Delegation Security Token was included in the request.

RightsDataLockerGet response ordering must be deterministic, and for this reason, in addition to applying any sort that may be included in the request, a secondary sort based on the RightsTokenID values is applied to the response. If no FilterClass parameters (see section **3.15**) are included in the request, the Coordinator SHALL apply the FilterClass `urn:dece:type:viewfilter:lastmodifieddate`.

When a call to RightsLockerDataGet results in multiple pages, it is possible for the Locker to be updated between page requests. In such circumstance, one or more Rights Tokens will be referenced more than once across the pages, so Nodes SHOULD assume that any multi-page RightsLockerDataGet response may contain repeated tokens. If the Node subscribes to push notification, it will be notified of the update and be able to make a second RightsLockerDataGet call to obtain the updated Rights Token(s). Otherwise the Node MAY choose to make a second RightsLockerDataGet call, using the OnOrAfter parameter or a conditional request (etag), to determine if an update occurred.

If the request does not include a Delegation Security Token, the Coordinator SHALL exclude from the response the Rights Token not issued by the requester.

7.1.7 RightsTokenUpdate()

7.1.7.1 API Description

This API allows limited fields of the Rights token to be updated. Precisely which fields are updated depends on Role.

7.1.7.2 API Details

Path:

```
[BaseURL]/Account/{AccountID}/RightsToken/{RightsTokenID}

[BaseURL]/RightsToken/{RightsTokenID}
```

Method: PUT

Authorized Roles:

`urn:dece:role:retailer[:customersupport]`
`urn:dece:role:dece:customersupport`
`urn:dece:role:coordinator:customersupport`

Security Token Subject Scope:

`urn:dece:role:user` or None

If present, it must match to a User in `active` or `pending` status.

Ignored if submitted for the endpoint form without AccountID

Opt-in Policy Requirements:

Request Parameters: None

Request Body:

Element	Attribute	Definition	Value	Card.
//RightsToken/RightsTokenFull		A fully populated RightsTokenFull object.		

The update request SHALL match the current contents of the rights token except for the items being updated.

Retailers may only update rights token that were purchased through them (that is, the NodeID in PurchaseInfo matches that retailer's NodeID). If a Security Token is presented, updates are made on

behalf of a User, so only Rights viewable by that User may be updated. Only the following fields may be updated by the Retailer named in //PurchaseInfo/NodeID:

Element or Attribute	Constraints
@ALID[1]	Update
@ContentID[1]	Update
SoldAs	Update
RightsProfiles/PurchaseProfile	Add, update, delete elements
RightsProfiles/PurchaseProfile/@MediaProfile	Add, update, delete elements (e.g. change from HD to SD)
RightsProfiles/PurchaseProfile/DiscreteMediaRightsRemaining	Not directly changeable (calculated by Coordinator from corresponding DiscreteMediaRightsToken)
RightsProfiles/PurchaseProfile/DiscreteMediaRightsRemaining/@ FulfillmentMethod	Not directly changeable (calculated by Coordinator from corresponding DiscreteMediaRightsToken)
RightsProfiles/PurchaseProfile/CanDownload	Update
RightsProfiles/PurchaseProfile/CanStream	Update
LicenseAcqBaseLoc	Add, update, delete
FulfillmentWebLoc	Add, update, delete
FulfillmentManifestLoc	Add, update, delete
StreamWebLoc	Add, update, delete
PurchaseInfo	Purchase info should not be updated unless the retailer needs to correct an initial error.
PurchaseInfo/NodeID	Not changeable (future policy review)
PurchaseInfo/RetailerTransaction	Update
PurchaseInfo/PurchaseAccount	Update. If this value is changed, the Retailer SHALL update the PurchaseUser element as well.
PurchaseInfo/PurchaseUser	Update (must be in Purchase Account). The UserID supplied MAY be different than the User identified in the Delegation Security Token.
PurchaseInfo/PurchaseTime	Update
PurchaseInfo/TransactionType	Update
@RightsLockerID	Not changeable. Its value is created and managed by the Coordinator.

[1] Asset identifiers should almost never be updated. The system relies on these identifiers to link Rights Tokens to content, define hierarchical metadata structures, map logical assets to digital (physical) assets etc. A Content Provider may wish to change an Asset identifier if a mistake was made but even then it may be preferable to leave the identifier as is rather than correct it.

Table 38: Allowed Resource Changes for RightsTokenUpdate

Any element retrieved by a GET, including those "Not directly changeable" ones, SHALL be included in an update request by the Retailer. However, elements marked as "Not directly changeable" in the table above are ignored (left intact) in an update request, leaving the original value at the Coordinator. For example, DiscreteMediaRightsRemaining information is managed exclusively by the Coordinator and is ignored during an UPDATE.

If a request includes changes to other fields, that is, for which changes are not allowed, no changes to such fields will be made, and an error will be returned.

The Rights Token status SHALL NOT be set to `deleted` using this API. The RightsTokenDelete API should be used instead.

An update to a Rights Token may have secondary consequences on Discrete Media Rights, and the Coordinator shall verify that the number of available Discrete Media Rights matches the updated DiscreteMediaRightsRemaining. If the Coordinator is unable to adjust the number of Discrete Media Rights Tokens, an error is returned. Discrete Media Rights are discussed in section 16.

7.1.7.3 Behavior

The Rights Token is updated. This is a complete replacement, so the update request must include all data.

The Coordinator SHALL require that:

- The ALID attribute value is a valid identifier, with a corresponding LogicalAsset resource in `active` status,
- The ContentID attribute value is a valid identifier with a corresponding BasicMetadata resource in `active` status,
- When SoldAs is present
    - All ContentID elements in the Rights Token's SoldAs element contain a valid identifier with a corresponding BasicAsset resource in `active` status,
    - The identifier in the RightsTokenData/@ContentID attribute exists in one instance of SoldAs/ContentID list, or within the Bundle referenced by SoldAs/BundleID
    - If SoldAs contains a BundleID:

- The BundleID is a valid identifier and corresponds to a Bundle resource in `active` status (the 'referenced Bundle'),

RightsTokenData/@ALID and RightsTokenData/@ContentID attributes correspond with ALID and ContentID in one instance of a LogicalAssetReference element in the referenced Bundle.

When the update is the result of a user request, the Node SHALL include a valid Delegation Security Token. Whenever a Delegation Security Token is included in the request, the Coordinator SHALL send an email notification to all the Users in the Account.

The URI path that excludes the {AccountID} parameter is exclusively for the use of Retailer (Issuer only) and its Customer Support subrole. Attempts to update Rights Tokens created by other retailers shall result in a 403 Forbidden HTTP Status response.

Nodes SHALL NOT include a Delegation Security Token when using the URI path that excludes the {AccountID} parameter. If a Delegation Security Token is included in this case, it will be ignored by the Coordinator. If a Node has a valid, active Delegation Security Token, it SHOULD use the API form that includes the {AccountID} parameter, unless the Node's intent is to correct a Right.

RightsToken updates without an AccountID and Delegation Security Token is reserved for cases when a Retailer needs to correct or update an existing Right. No email notification is sent when Rights are updated using the URI path that excludes the {AccountID} parameter.

7.1.8 DownloadPlaybackLicenseReporting()

7.1.8.1 API Description

The DownloadPlaybackLicenseReporting API is used to record licensing at the Coordinator.

7.1.8.2 API Details

Path:

```
[bhost]/Account/{AccountID}/RightsToken/{RightsTokenID}/DPLicense
```

Method: POST

Authorized Roles:

```
urn:dece:role:retailer[:customersupport]
```

Security Token Subject Scope: None

Opt-in Policy Requirements: None

Request Parameters:

`RightsTokenID` is the unique identifier for a rights token
`AccountID` is the unique identifier for an Account

Request Body:

Element	Attribute	Definition	Value	Card.
License		Defines the issued playback license that is being reported	`dece:PlaybackLicense-type`	1

Response Body: None

7.1.8.3 Behavior

Nodes can use this API to report the playback-licenses issued. This API is available only at the batch 'b' host. Nodes SHALL include the following information in the request.

`LicenseIssuanceDateTime`

`RequestingUserID`

`RightsTokenID`

Additionally, Nodes can also indicate the `Mediaprofile` for which the license was issued. The Coordinator shall allow all valid mediaprofiles as defined in section 6.5.3.

The following are Retailer-defined strings and may be included when reporting.

`LicensingNodeID – Defaults to invoking node.`

`NodeLicenseID`

`PlaybackDeviceName`

`LicensingAuthorityName`

Coordinator SHALL validate that the request body is syntactically valid, the `RightsTokenID` identified in the request body matches the one in the request URL and SHALL respond with an HTTP 202 status code (*Accepted*).

During post-processing, Coordinator SHALL validate that the User identified in the `RequestingUserID` and the RightsToken identified in the `RightsTokenID` match existing User and RightsToken resources

respectively. Coordinator SHALL not validate the status of these resources or their current association with one another. Errors identified during post-processing are written to the logs.

Note that the new resource is not accessible via a GET request at the Coordinator and therefore not visible to Nodes.

The Coordinator will respond with a 503 status code (*Service Unavailable*) when the request cannot be fulfilled due to service availability failures. Nodes will have to retry the request at a later time.

7.1.9 RightsTokenListCreate()

7.1.9.1 API Description

The RightsTokenListCreate API is used to a list of Rights Tokens to a Rights Locker in a single API call.

7.1.9.2 API Details

Path:

```
[BaseURL]/Account/{AccountID}/RightsToken/List
```

Method: POST

Authorized Roles:

`urn:dece:role:retailer[:customersupport]`

Security Token Subject Scope: `urn:dece:role:user`

Opt-in Policy Requirements: None

Request Body:

Element	Attribute	Definition	Value	Card.
RightsTokenList		RightsTokenList containing a list of RightsTokenData elements . All required information SHALL be included in the request.	dece:RightsLockerData-type	1

Response Body: None

7.1.9.3 Behavior

Nodes SHALL include at least one fully populated RightsToken in the payload. The list can contain a maximum of 60 fully populated RightsTokens. Coordinator SHALL return an appropriate error when the payload contains zero or more than 60 RightsToken elements.

The Coordinator SHALL require that:

- The AccountID path parameter is a valid identifier, with a corresponding Account resource in active status; the AccountID attribute value in the request body matches the AccountID in the request URL
- The RightsLockerID attribute value is a valid identifier, with a corresponding RightsLocker resource and matches the RightsLockerID associated with the AccountID specified in the request URL

In addition, all the validation requirements that apply to discrete RightsTokenCreate apply to this flavor as well.

The request body is processed and when the first validation error occurs, further evaluations are skipped and errors are returned. For example, Retailer posts a RightsTokenList payload in which RightsToken 3 has invalid CID and RightsToken 17 has invalid ALID set. Validation fails for CID 3, further evaluations are skipped and error Invalid CID – CID 3 is returned.

Coordinator will not add or change transaction type. (Billing will work as always, based on transaction type per token.)

No DMR operations will be included. Retailers should utilize the DMRFulfill API for the same.

Successful requests will result in an HTTP 201 Created response, with the Location header returning the URL of the RightsLockerDataGet API including the OnOrAfter query parameter set to the date and time when Coordinator received the RightsTokenListCreate request and the byReference query parameter (which is the default, but included for clarity). Upon successful creation, the Coordinator SHALL set the RightToken/s to active status.

The Coordinator SHALL ensure the atomicity of this API call. In other words, the only possible outcomes of this operation are that either all the RightsTokens requested have been created, or no resource was created as a result of the API call. If any individual Rightstoken cannot be created, the entire list fails, and error/s returned.

7.2 Rights Token Resource

A Rights Token represents a User's entitlement to a digital asset resource. Rights Tokens are defined in four structures to accommodate the various authorized views of the Rights Token. Each succeeding structure inherits the data elements of the preceding data structure, as depicted in the following diagram.

Figure 13: Rights Token Resource

- **RightsTokenBasic** identifies the digital assets contained in the Rights Token, and the rights profiles associated with the digital assets represented by the Rights Token.
- **RightsTokenInfo** extends RightsTokenBasic to include fulfillment details related to licensing, downloading, and streaming the digital asset represented by the Rights Token.
- **RightsTokenData** extends RightsTokenInfo to include details about the User's purchase of the Rights Token, and the visibility constraints on the Rights Token.
- **RightsTokenFull** extends RightsTokenData to a complete view of the Rights Token's data, including the Rights Locker where the Right Token can be accessed by the User, as well as the Rights Token status and status history.
- **RightsTokenDetails** provides an asset metadata populated version of the rights tokens in a list (Locker), instead of the purchase profile centric view.. Clients may select this response variant by means of the `response=metadata` query parameter.
- **RightsTokenLocation** provides devices with a means of obtaining only the download information for a Rights Token. Clients may select this response variant by means of the `response=download` query parameter.

7.2.1 RightsToken Definition

Element		Attribute	Definition	Value	Card.
RightsToken				dece:RightsTokenObject-type	
		RightsTokenID	An identifier (unique to an Account and a Node) for the RightsToken, created by the Coordinator. Nodes SHALL NOT create nor alter the RightsTokenID.	dece:EntityID-type	0..1
One of:	RightsTokenBasic		Representation of the Rights Token (based on Policies and other properties of the Rights Token, and the associated Account, User, and API Client)	RightsTokenBasic-type	
	RightsTokenInfo			RightsTokenInfo-type	
	RightsTokenData			RightsTokenData-type	
	RightsTokenFull			RightsTokenFull-type	
	RightsTokenDetails			RightsTokenDetails-type	
	RightsTokenLocation			RightsTokenLocation-type	
PolicyList				dece:PolicyList-type	0..1

Table 39: RightsToken Definition

7.2.2 RightsTokenBasic Definition

Element	Attribute	Definition	Value	Card.
RightsTokenBasic			dece:RightsTokenBasic-type	
	ALID	The logical asset identifier for a RightsToken	md:AssetLogicalID-type	
	ContentID	The content identifier for the digital asset associated with the RightsToken	md:ContentID-type	
SoldAs		Retailer-specified product information (see Table 41)	dece:RightsSoldAs-type	0..1
RightsProfiles		The list of transaction profiles for the RightsToken	dece:RightsProfileInfo-type	
ResourceStatus		See section 17.2		0..1

Table 40: RightsTokenBasic Definition

7.2.3 SoldAs Definition

Element		Attribute	Definition	Value	Card.
SoldAs				dece:RightsSoldAs-type	
DisplayName			The localized display name defined by the retailer	dece:LocalizedString Abstract-type	0..1
	ProductID		"ProductID" is any identifier used to identify a product associated with this Rights Token. DECE has no defined use for this element, so it may be used at Retailer's discretion.	xs:string	0..1
One of:	ContentID		The content identifier for the digital asset associated with the RightsToken, based on how the retailer sold the asset (this MAY be different from the RightsTokenBasic/ ContentID). The Coordinator SHALL verify ContentIDs with established BasicAsset@ ContentIDs.	md:ContentID-type	1…n
	BundleID			dece:EntityID-type	0..1

Table 41: SoldAs Definition

7.2.4 RightsProfiles Definition

This structure describes the details of the purchase associated with a Rights Token.

Element	Attribute	Definition	Value	Card.
RightsProfiles			dece:RightsProfilesInfo-type	
PurchaseProfile		See Table 43	dece:PurchaseProfile-type	0..n

Table 42: RightsProfiles Definition

7.2.5 PurchaseProfile Definition

Element	Attribute	Definition	Value	Card.
PurchaseProfile			dece:PurchaseProfile-type	
	MediaProfile	The digital asset profile (see Table 15)	dece:AssetProfile-type	

Element	Attribute	Definition	Value	Card.
	HighDynamicRange	Boolean indicated the Right includes High Dynamic Range (HDR). Default value is "false". This profile parameter may only be "true" in PurchaseProfiles having a MediaProfile of `urn:dece:type:mediaprofile:hd` or `urn:dece:type:mediaprofile:uhd`	`xs:boolean`	0..1
	HighFrameRate	Boolean indicated the Right includes High Frame Rate (HFR). Default value is "false". This profile parameter may only be "true" in PurchaseProfiles having a MediaProfile of `urn:dece:type:mediaprofile:hd` or `urn:dece:type:mediaprofile:uhd`	`xs:boolean`	0..1
	WideColorGamut	Boolean indicated the Right includes Wide Color Gamut. Default value is "false". This profile parameter may only be "true" in PurchaseProfiles having a MediaProfile of `urn:dece:type:mediaprofile:hd` or `urn:dece:type:mediaprofile:uhd`	`xs:boolean`	0..1
	NextGenAudio	Boolean indicated the Right includes Next Generation Audio. Default value is "false". This profile parameter may only be "true" in PurchaseProfiles having a MediaProfile of `urn:dece:type:mediaprofile:hd` or `urn:dece:type:mediaprofile:uhd`	`xs:boolean`	0..1
	ThreeD	Boolean indicated the Right includes 3D. Default value is "false". This profile parameter may only be "true" in PurchaseProfiles having a MediaProfile of `urn:dece:type:mediaprofile:hd` or `urn:dece:type:mediaprofile:uhd`	`xs:boolean`	0..1
DiscreteMediaRightsRemaining		The collection of Discrete Media Rights available in the Rights Token. The maximum quantity is determined by the defined Ecosystem parameter DISCRETE_MEDIA_LIMIT (specified in [DSystem]). Changes to existing DiscreteMediaRights must be made using the functions specified in section 16.1.	`dece:DiscreteMediaRightsRemaining-type`	0..1
CanDownload		Boolean indicator of whether the RightsToken allows downloading (defaults to TRUE)	`xs:boolean`	
CanStream		Boolean indicator of whether the RightsToken allows streaming (defaults to TRUE)	`xs:boolean`	

Table 43: PurchaseProfile Definition

7.2.6 DiscreteMediaRights Definition

The DiscreteMediaRightsRemaining type is an enumeration of Discrete Media Rights within a RightsToken. A NULL set, or the absence of this element, is an indication that no discrete media rights are present.

Element	Attribute	Definition	Value	Card.
DiscreteMediaRightsRemaining		Indicates any available DiscreteMediaRights, as calculated by the Coordinator (see section 16.1).	`dece:DiscreteMediaRightsRemaining-type` extends `xs:positiveInteger`	
	FulfillmentMethod	Indicates which fulfillment methods are allowed given this Right.	`xs:NMTokens`	0..1

Table 44: DiscreteMediaRightsRemaining Definition

7.2.7 RightsTokenInfo Definition

RightsTokenInfo-type extends the `RightsTokenBasic-type` definition, and adds the following elements:

Element	Attribute	Definition	Value	Card.
RightsTokenInfo			`dece:RightsTokenInfo-type`	
LicenseAcqBaseLoc		The base location from which the LAURL to fulfill DRM License requests can be constructed. See Section 12.2.2 in [DSystem]	`xs:anyURI`	0..1
FulfillmentWebLoc		The network location from which the desired DCC of the Right can be obtained. See Section 11.1.2 in [DSystem]. This value MAY be omitted if fulfillment is not required.	`dece:ResourceLocation-type`	0…n
FulfillmentManifestLoc		The network location from which the fulfillment manifest can be obtained. See Section 11.1.3 in [DSystem]. This value MAY be omitted if fulfillment is not required.	`dece:ResourceLocation-type`	0…n

Element	Attribute	Definition	Value	Card.
StreamWebLoc		Identifies one or more Streaming endpoint URI's associated with the identified Media Profile. This value SHALL be included if at least one purchase profile includes CanStream as true. The Coordinator will return an error if StreamWebLoc is not completed appropriately.	dece:ResourceLocation-type	0..n

Table 45: RightsTokenInfo Definition

7.2.8 RightsTokenLocation Definition

Element	Attribute	Definition	Value	Card.
RightsTokenLocation			dece:RightsTokenLocation-type	
	ALID	The Logical Asset ID for the RightsToken	dece:EntityID-type	
	ContentID	The Content ID for the RightsToken	dece:EntityID-type	
LicenseAcqBaseLoc		The base location from which the LAURL to fulfill DRM License requests can be constructed. See Section 12.2.2 in [DSystem]	xs:anyURI	0..1
FulfillmentWebLoc		The network location from which the desired DCC of the Right can be obtained. See Section 11.1.2 in [DSystem]. This value MAY be omitted if fulfillment is not required.	dece:ResourceLocation-type	0…n
FulfillmentManifestLoc		The network location from which the fulfillment manifest can be obtained. See Section 11.1.3 in [DSystem]. This value MAY be omitted if fulfillment is not required.	dece:ResourceLocation-type	0…n
StreamWebLoc		Identifies one or more Streaming endpoint URI's associated with the identified Media Profile. This value MAY be omitted if streaming is not required.	dece:ResourceLocation-type	0..n

7.2.9 ResourceLocation Definition

Element	Attribute	Definition	Value	Card.
ResourceLocation-type				
	MediaProfile	The media profile specific download location	`xs:anyURI`	0..1
Location		A network-addressable URI	`xs:anyURI`	
Preference		An integer that indicates the retailer's preference, if more than one Location is provided. Higher integers indicate a lower preference. Clients MAY choose any Location based on its own deployment characteristics. The Web Portal shall select the `Location` URL with the lowest provided `Preference` value (or a randomly selected `Location` if no `Preference` is indicated) when displaying a Right.	`xs:int`	0..1

Table 46: ResourceLocation Definition

7.2.10 RightsTokenData Definition

RightsTokenData-type extends the RightsTokenInfo-type with the following elements:

Element	Attribute	Definition	Value	Card.
RightsTokenData			`dece:RightsTokenData-type extends dece:RightsTokenInfo-type`	
PurchaseInfo		See Table 48	`dece:RightsPurchaseInfo-type`	

Table 47: RightsTokenData Definition

7.2.11 PurchaseInfo Definition

Element	Attribute	Definition	Value	Card.
PurchaseInfo			`dece:RightsPurchaseInfo-type`	
NodeID		The identifier of the retailer that sold the RightsToken	`dece:EntityID-type`	0..1
RetailerTransaction		A retailer-supplied string which may be used to record an internal retailer transaction identifier. This element is only visible to the Retailer that created the Right.	`xs:string`	0..1
PurchaseAccount		The Account identifier URI that the RightsToken was initially issued to	`dece:EntityID-type`	

Element	Attribute	Definition	Value	Card.
PurchaseUser		The User identifier URI under which the Right was initially issued to the Account	`dece:EntityID-type`	
PurchaseTime		The date and time the Right was issued by the Retailer	`xs:dateTime`	
TransactionType		A DECE-defined code provided by the Retailer to indicate the type of the transaction (for example an EST transaction or a disc-to-digital transaction). The Retailer SHALL only use values published by DECE. This element is only visible to the Retailer that created the Right.	`xs:string`	0..1

Table 48: PurchaseInfo Definition

`TransactionType` information is used for DECE billing purposes and analysis purposes.

7.2.12 RightsTokenFull Definition

`RightsTokenFull-type` is a `RightsTokenData-type` with additional metadata information and the RightsLockerID.

Element	Attribute	Definition	Value	Card.
RightsTokenFull			`dece:RightsTokenFull-type extends RightsTokenData-type`	
RightsLockerID		The system-wide unique identifier for a Rights Locker where a given token resides	`dece:EntityID-type`	

Table 49: RightsTokenFull Definition

7.2.13 RightsTokenDetails Definition

`RightsTokenDetails-type` provides a metadata populated response for the Rights Token. The data is determined by the Coordinator based on the associated BasicAsset metadata. The definition column in the following table describes the mapping to the corresponding BasicAsset elements.

To determine which language the response should provide, the Coordinator first consults any provided Accept-Lang HTTP Header, then consults the preferred language (if any) associated with the User of the request, then consults to default language identified in the corresponding BasicAsset's LocalizedInfo, and finally, resorts to English (en).

RatingSet selection is performed as a best effort by the Coordinator. If the User associated with the request has a Country specified in their profile, the Coordinator will include the rating systems associated with the applicable Geography Policy (see Appendix F). If such a determination cannot be made, the Coordinator may use any method to determine the appropriate RatingSet (or include them all). Should a full list of Ratings be required by the client, they may obtain them via the BasicAsset itself, where all ratings are returned.

Note: This structure, RightsTokenDetails, is slated for deprecation. It is recommended that implementations avoid its use. Recommend usage is RightsTokenInfo plus BasicMetadata queries. Future implementation may include a modified version of this element.

Element	Attribute	Definition	Value	Card.
RightsTokenDetails			`dece:RightsTokenDetails-type`	
	ALID	The Logical Asset identifier of the Right	`dece:EntityID-type`	
	ContentID	The ContentID of the Right	`dece:EntityID-type`	
	Language	The language the metadata is presented in. Corresponds to the [MLMeta] use of the Language attribute in `md:MDBasicDataType` See note above on language selection.	`xs:language`	
TitleDisplay60		Corresponds to the BasicData/LocalizedInfo/TitleDisplay60 element	`xs:string`	
ArtReference		Corresponds to the BasicData/LocalizedInfo/ArtReference element	`xs:anyURI`	0..n
Summary190		Corresponds to the BasicData/LocalizedInfo/Summary190 element	`xs:string`	
Genre		Corresponds to the BasicData/LocalizedInfo/Genre element	`xs:string`	0..n
RunLength		Corresponds to the BasicData/RunLength element	`xs:duration`	0..1
WorkType		Corresponds to the BasicData/WorkType element	`xs:string`	

Element	Attribute	Definition	Value	Card.
RatingSet		Corresponds to the BasicData/RatingSet element	md:ContentRating-type	0..1

Table 50: RightsTokenDetails-type

7.2.14 RightsTokenList Definition

Element		Attribute	Definition	Value	Card.
RightsTokenList				dece:RightsLockerData-type	
		Group: dece:ViewFilterAttr-type	Response filtering information, see section **17.5**	dece:EntityID-type	
		RightsLockerID	The system-wide unique identifier for a Rights Locker where a given token resides	dece:EntityID-type	
		AccountID	The unique identifier for the Account	dece:EntityID-type	
One of:	RightsTokenReference		Rights Token identifier augmented with creation/update date information	dece:RightsTokenReference-type	0..n
	RightsToken		Rights Token object. See 7.2.1	dece:RightsTokenObject-type	0..n

Table 51: RightsLockerData-type Definition

The RightsTokenReference-type is defined as follows:

Element	Attribute	Definition	Value	Card.
	RightsTokenID		dece:EntityID-type	
	ContentID		md:ContentID-type	
	CurrentStatus	See section **17.2**	dece:DatedEntityElementAttrGroup-type	
	DatedElementAttrGroup-type			

Table 52: RightsTokenReference-type Definition

Element	Attribute	Definition	Value	Card.
	DatedElementAttrGroup-type		dece:DatedEntityElementAttrGroup-type	
	CreatedDate	Creation date of the resource	xs:dateTime	0..1
	UpdatedDate	Last update date of the resource	xs:dateTime	0..1

Table 53: DatedEntityElementAttrGroup-type Definition

7.2.15 License-type Definition

Element	Attribute	Definition	Value	Card.
License			dece:License-type	
LicensingNodeID		The node that issued the playback license. Defaults to the node that reported the license at the Coordinator.	dece:EntityID-type	
LicenseIssuanceDateTime		Date and time when the playback license was issued	xs:dateTime	
RequestingUserID		The User that initiated the license reporting.	dece:EntityID-type	
RightsTokenID		Identifier of the RightsToken that holds the asset for which the license was issued.	dece:EntityID-type	
MediaProfile		MediaProfile for the RightsToken	dece:AssetProfile-type	0..1
NodeLicenseID		Unique DRM Identifier	xs:string	0..1
PlaybackDeviceName		Retailer-defined string	xs:string	0..1
LicensingAuthorityName		Retailer-defined string	xs:string	0..1
SubDividedGeolocation		Identifies an approximate geographic location of the stream client, when available.	dece:SubDividedGeolocation-type	0..1

Table 544: License-type Definition

7.2.16 Rights Token Status Transitions

The possible Status values are: `active`, `pending`, `deleted`, and `other`.

7.2.17 Rights De-Identification Process

Rights Tokens with a status of `urn:dece:type:status:deleted` for DCOORD_DEIDENTIFY_RIGHT_THRESHOLD, SHALL be modified by the Coordinator as follows.

- //PurchaseInfo/PurchaseAccount SHALL be changed to a unique anonymized value to dissociate the Account from the Rightstoken
- //PurchaseInfo/PurchaseUser SHALL be changed to a unique anonymized value to dissociate the User from the Rightstoken
- The AccountID and requesting UserID SHALL be changed to a unique anonymized value for all Discrete Media tokens and Discrete Media Token History associated with the RightsToken.

The AccountID and requesting UserID SHALL be changed to a unique anonymized value for all stream data and stream history associated with the RightsToken.

8 License Acquisition

Note: Licensing of content to devices is now exclusively the responsibility of Retailers, as allowed by agreements by Content Providers. The Coordinator no longer facilitates content licening.

9 Domains

Note: Management of DRM systems is now exclusively the responsibility of Retailers, as allowed by agreements by Content Providers. The Coordinator no longer facilitates content licening or Domain management.

10 Legacy Devices

Note: This section 10 is not currently implemented and subject to change..

A product or application that is not a compliant DECE Device (as specified in [DSystem]) but is allowed to have Content delivered to it by a Retailer is considered a Legacy Device.

10.1 Legacy Device Functions

Because nothing can be assumed of a Legacy Device's compatibility with the DECE ecosystem, it is envisioned that a single Node will: manage the Legacy Device's content in a proprietary fashion and act as a proxy between the Legacy Device and the Coordinator. The Coordinator must nonetheless be able to register a Legacy Device in the Account so that Users can see the corresponding information in the Web Portal. To enable this, a set of simple functions is defined in the subsequent sections.

10.1.1 LegacyDeviceCreate()

10.1.1.1 API Description

This function creates a new Legacy Device and adds it to the Account provided a Device slot is available.

10.1.1.2 API Details

Path:

```
[BaseURL]/Account/{AccountID}/LegacyDevice
```

Method: POST

Authorized Roles: `urn:dece:role:retailer[:customersupport]`

Request Parameters: None

Security Token Subject Scope:

```
urn:dece:role:user:class:standard
urn:dece:role:user:class:full
```

Applicable Policy Classes: N/A

Request Body:

Element	Attribute	Definition	Value	Card.
LegacyDevice			dece:DeviceInfo-type	

Response Body: None

10.1.1.3 Behavior

The Coordinator first verifies that the maximum number of Legacy Devices has not been reached and the maximum number of total Devices has not been reached. If not, the Legacy Device information is stored in the Account and the associated identifier created, if required.

The DeviceID can be used, in conjunction with the Node's DeviceManagementURL, to calculate the Node's endpoint for servicing a Legacy Device by postpending the parameter deviceID=[DeviceID] the the DeviceManagementURL. If the DeviceManagementURL includes other query parameters, the deviceID parameter is appended with the "&" (ampersand) reserved character, otherwise a new query segment is postpended. For example:

```
https://devices.example.com/manage?deviceID=82937dahdiaj93
https://devices.example.com/manage?type=x-type&deviceID=82937dahdiaj93
```

10.1.2 LegacyDeviceDelete()

10.1.2.1 API Description

10.1.2.2 API Details

Path:

```
[BaseURL]/Account/{AccountID}/LegacyDevice/{DeviceID}
```

Method: DELETE

Authorized Roles:

```
urn:dece:role:retailer[:customersupport]
urn:dece:role:dece:customersupport
urn:dece:role:coordinator:customersupport
```

Request Parameters:

AccountID is the unique identifier for an Account
DeviceID is the unique identifier for a Device

Security Token Subject Scope:

```
urn:dece:role:user:class:standard
urn:dece:role:user:class:full
```

Applicable Policy Classes: N/A

Request Body: None

Response Body: None

10.1.2.3 Behavior

Only the Node that created the Legacy Device may delete it (besides the customer support roles as defined above).

10.1.3 LegacyDeviceUpdate()

10.1.3.1 API Description

10.1.3.2 API Details

Path:

```
[BaseURL]/Account/{AccountID}/LegacyDevice/{DeviceID}
```

Method: PUT

Authorized Roles:

```
urn:dece:role:retailer[:customersupport]
```

Request Parameters: None

Security Token Subject Scope:

```
urn:dece:role:user:class:standard
urn:dece:role:user:class:full
```

Applicable Policy Classes: N/A

Request Body:

Element	Attribute	Definition	Value	Card.
LegacyDevice			dece:DeviceInfo-type	

Response Body: None

10.1.3.3 Behavior

The Rights Locker verifies that the device identifier corresponds to a known (that is existing) Device resource. If so it replaces the data with the element provided in the request. Only the Node that created the Legacy Device may update it.

11 Streams

Streams allow a User to view the content of digital assets (to which the User is entitled by virtue of a Rights Token in the Account's Rights Locker). They are not artifacts in the same way that DVDs are, rather they are real-time representations of digital content.

11.1 Stream Functions

Stream resources provide reservation and stream information to authorized Roles.

11.1.1 StreamCreate()

11.1.1.1 API Description

A LASP SHALL call StreamCreate() to request a streaming session lease for specified content on behalf of an Account or User.

A LASP NEED NOT wait for a Coordinator response before starting the associated streaming session.

The Coordinator grants authorization to create a stream by responding with a HTTP *201 Created* status that includes the newly created stream resource in the HTTP Location header. The stream lease that is created includes an expiration timestamp (`Expiration`).

If the Coordinator responds with any HTTP response other than *201 Created, 408 Request Timeout or 5xx Service Errors*, the LASP SHALL NOT begin the streaming session, or if the LASP has started the streaming session the LASP SHALL terminate the streaming session.

LASP streaming sessions are global to an account and are not allowed to exceed the duration defined by the Ecosystem parameter DYNAMIC_LASP_AUTHENTICATION_DURATION (specified in [DSystem]), without re-authentication.

The requesting Node MAY supply a `TransactionID` to the Coordinator for its' own use.

The Coordinator must verify the following criteria to grant the request:

- The Account possesses the Rights Token.
- The number of active LASP sessions is less than the number determined by the defined Ecosystem parameter LASP_SESSION_LIMIT (as defined in [DSystem] Section 16)
- The User has requisite stream creation privileges and meets the Parental Control policy requirements. (This requirement only applies to the `urn:dece:role:lasp:dynamic` Role.)

If granted, The Coordinator SHALL establish an initial stream lease ExpirationDateTime of RENEWAL_MAX_ADD from the time this API is invoked.

11.1.1.2 API Details

Path:

```
[BaseURL]/Account/{AccountID}/Stream
```

Method: POST

Authorized Roles:

```
urn:dece:role:lasp:linked[:customersupport]
urn:dece:role:lasp:dynamic[:customersupport]
```

Security Token Subject Scope:

For Dynamic LASP: `urn:dece:role:user`

For Linked LASP: `urn:dece:role:account`

Opt-in Policy Requirements: None

Request Parameters: AccountID is the unique identifier for an Account

Request Body:

Element	Attribute	Definition	Value	Card.
Stream		Defines the stream that is being requested	`dece:Stream-type`	

The Node SHALL NOT include the `Stream/@StreamHandleID` in the request.

Response Body: None

If no error conditions occur, the Coordinator SHALL respond with an HTTP 201 status code (*Created*) and a Location header containing the URL of the created resource.

The resulting resource, when created, will include the {streamhandleid}, and is considered a DECE assigned identifier, whose syntax will be:

```
<STREAMHANDLEID> ::= "urn:dece:streamhandleid:" <streamhandleiduniquepart>
```

where <streamhandleiduniquepart> is defined as one or more characters that are in the set 'unreserved' as defined in [RFC3986], Section 2.3.

11.1.1.3 Behavior

The RightsTokenID in the request SHALL be for the content being requested.

When invoked by a Dynamic LASP, the `RequestingUserID` element SHALL be supplied. A Linked LASP MAY provide the RequestingUserID element. If provided, the Coordinator SHALL match its value with the User associated with the presented Delegation Security Token.

Prior to enabling a stream, the Coordinator validates that an Account has a Right to stream as determined by the existence of an `active` Rights Token associated with that ALID in the associated Account.

The Coordinator SHALL maintain stream description parameters for all streams, both active and inactive (see Table 56 for details). The Coordinator will establish the initial stream parameters `ResourceStatus`, `ExpirationDateTime`, and `StreamHandleID`.

The Coordinator SHALL set Account/ActiveStreamCount to reflect the number of available streams.

A newly created stream SHALL NOT have an expiration date and time that exceeds the expiration date and time of the provided Security Token.

11.1.2 StreamListView(), StreamView()

11.1.2.1 API Description

This API supports LASP, Web Portal and CS functions. The data returned is dependent on the Role making the request.

11.1.2.2 API Details

Path:

```
[BaseURL]/Account/{AccountID}/Stream/{StreamHandleID}

[BaseURL]/Account/{AccountID}/Stream/List
```

Method: GET

Authorized Roles:

```
urn:dece:role:portal[:customersupport]
urn:dece:role:lasp:linked[:customersupport]
urn:dece:role:lasp:dynamic[:customersupport]
urn:dece:role:coordinator:customersupport
urn:dece:role:dece:customersupport
```

```
urn:dece:role:retailer[:customersupport]
urn:dece:role:accessportal[:customersupport]
```

Security Token Subject Scope:

For Linked LASP: `urn:dece:role:account`

otherwise: `urn:dece:role:user`

Opt-in Policy Requirements: urn:dece:type:policy:ManageAccountConsent as described in Section 11.1.6.

Request Parameters:

AccountID is the unique identifier for an Account
StreamHandleID is the unique identifier for an active Stream.

Request Body: None

Response Body:

When StreamHandleID form of the invocation URL is used, Stream is returned.

Element	Attribute	Definition	Value	Card.
Stream			`dece:Stream-type`	

When the '/List' form of the invocation URL is used, StreamList is returned.

Element	Attribute	Definition	Value	Card.
StreamList			`dece:StreamList-type`	

11.1.2.3 Behavior

A Node makes this request on behalf of an authorized User, and the Coordinator's response depends on the requestor:

Stream Visibility SHALL be in accordance with Table in 11.1.67.

If the requestor is a Role other than LASP Customer Support StreamList responses for streams that refer to Content that are not visible to a User based on their Parental Control settings SHALL contain

StreamClientNickname, if present, and, SHALL contain a RightsTokenID of `urn:dece:stream:generic`.

If the requestor is not a member of the same Organization as the Stream creator, the following information SHALL NOT be returned:

- //Stream/TransactionID
- //Stream/SubDividedGeolocation

The above restriction does not apply to the `urn:dece:role:portal` Role.

As User IDs are Node-specific, `RequestingUserID` is returned in a form suitable for the requesting Node.

The Coordinator will retain stream information for a configurable period, which SHALL NOT be less than DCOORD_STREAM_INFO_MIN_RETENTION. Stream resources created beyond that date range will not be available using any API.

The sort order of the response SHALL be based on the Streams' created datetime value, in descending order.

11.1.3 Checking for Stream Availability

StreamList provides the `AvailableStreams` attribute, to indicate the number of available streams, as not all active streams are necessarily visible to the LASP. Nevertheless, it is possible that, depending on a delay between a StreamListView() and StreamCreate() message, additional streams may be created by other Nodes. LASPs should account for this condition in their implementations, but SHALL NOT use StreamCreate() as a mechanism for verifying stream availability.

11.1.4 StreamDelete()

11.1.4.1 API Description

The LASP uses this message to inform the Coordinator that the content is no longer being streamed to the user. The content could have been halted due to completion of the content stream, user action to halt (rather than pause) the stream, or a time out occurred exceeding the duration of streaming content policy.

Streams which have expired SHALL have their status set to `deleted` state upon expiration by the Coordinator.

11.1.4.2 API Details

Path:

```
[BaseURL]/Account/{AccountID}/Stream/{StreamHandleID}
```

Method: DELETE

Authorized Roles:

`urn:dece:role:lasp:linked[:customersupport]`
`urn:dece:role:lasp:dynamic[:customersupport]`

Security Token Subject Scope:
For Dynamic LASP: `urn:dece:role:user`
For Linked LASP: `urn:dece:role:account`

Opt-in Policy Requirements: None

Request Parameters:

AccountID is the unique identifier for an Account
StreamHandleID is the unique identifier for an active stream.

Request Body: None

Response Body: None

11.1.4.3 Behavior

The Coordinator records the status of the Stream in the <Current> status element as *deleted*, indicating that the stream is inactive. The <AdminGroup> element of ResourceStatus is updated with the current date and time and the identifier of the Node that closed the stream.

A Stream may only be deleted by the Node which created it (or by any customer support Node).

Deleted streams are maintained for a period of time at the discretion of the Coordinator, but not less than DCOORD_STREAM_INFO_MIN_RETENTION.

11.1.5 StreamRenew()

If a LASP has a need to extend a lease on a stream reservation, they may do so via the StreamRenew() request.

The HTTP HEAD Method is not supported on this URL.

11.1.5.1 API Description

The LASP uses this message to inform the Coordinator that the expiration of a stream needs to be extended.

The Coordinator will support this API at the [pHost] form of the URL.

11.1.5.2 API Details

Path:

```
[BaseURL]/Account/{AccountID}/Stream/{StreamHandleID}
```

Method: PUT

Authorized Roles:

```
urn:dece:role:lasp:linked[:customersupport]
urn:dece:role:lasp:dynamic[:customersupport]
```

Security Token Subject Scope:
For Dynamic LASP: `urn:dece:role:user`
For Linked LASP: `urn:dece:role:account`

Opt-in Policy Requirements: None

Request Parameters:

`AccountID` is the unique identifier for an Account
`StreamHandleID` is the unique identifier for an active stream.

Request Body:

The Stream object `dece:Stream-type` is provided in the request, incorporating the requested new ExpirationDateTime.

Element	Attribute	Definition	Value	Card.
Stream			`dece:Stream-type`	

Response Body:

If no error conditions occur, the Coordinator SHALL respond with an HTTP `200 OK` status code. The updated Stream is included in the body of the response (`<Stream-type>`).

11.1.5.3 Behavior

A LASP MAY request up to DCOORD_STREAM_RENEWAL_MAX_ADD hours for the identified StreamHandle.

The Coordinator SHALL determine the allocated ExpirationDateTime value based on a number of parameters including the following rules:

- Streams may only be renewed for a maximum of DCOORD_STREAM_MAX_TOTAL hours. New streams must be created once a stream has exceeded the maximum time allowed.
- Stream lease renewals SHALL NOT exceed the date time of the expiration of the Security Token provided to this API.
- The ExpirationDateTime shall satisfy the following constraints..
  a) Must not extend the prior ExpirationDateTime by more than DCOORD_STREAM_RENEWAL_MAX_ADD hours.
  b) Must not extend the prior ExpirationDateTime by more than DCOORD_STREAM_MAX_TOTAL hours from the timepoint of stream creation.

The coordinator shall apply the following validation constraints on the elements of the supplied Stream object.

- The StreamHandleID shall match to that of an existing active stream for the Account i.e. must match the one in the URL.
- The following elements shall be ignored by the coordinator and the values for these elements of stream object held at the Coordinator shall be unchanged.
  a) RequestingUserID
  b) RightsTokenID
- The SubDividedGeolocation may change as long as the value complies with general requirements (syntax constraints).

- The ResourceStatus value shall be ignored and shall be set by Coordinator according to system determination.
- The coordinator shall modify the Stream object after satisfying validation constraints with that supplied in the PUT request.

If the Stream has already reached DCOORD_STREAM_MAX_TOTAL (prior to this renewal), a new Stream must be created and the Coordinator SHALL respond with the `StreamRenewalMaximumTimeReached` error code.

If Dynamic LASPs require renewal of a stream that exceeds the Security Token expiration, such LASPs SHALL request a new Security Token. The Coordinator MAY allow a renewal up to the validity period of the Security Token.

LASPs SHOULD keep an association between their local Stream accounting activities, and the expiration of the Coordinator Stream resource. Since most LASP implementations support pause/resume features, LASPs will need to coordinate the Stream lease period with the Coordinator, relative to any pause/resume activity. LASPs SHALL NOT provide streaming services beyond the expiration of the Stream resource.

11.1.6 Batch Stream Reporting

In some circumstances, LASPs may be required to report Stream activity but not utilize the Coordinator to enforce the LASP_SESSION_LIMIT. In such cases, the batch stream reporting API is used to notify the Coordinator of historical stream activity of an Account.

11.1.6.1 API Details

Path:

```
[bhost][versionPath]/Account/{AccountID}/Stream
```

Method: POST

Authorized Roles:

```
urn:dece:role:lasp:linked[:customersupport]
urn:dece:role:lasp:dynamic[:customersupport]
```

Security Token Subject Scope:

None. If a Delegation Security Token is provided, it is ignored.

Opt-in Policy Requirements: None

Request Parameters: AccountID is the unique identifier for an Account

Request Body:

Element	Attribute	Definition	Value	Card.
Stream		Defines the stream that is being requested	`dece:Stream-type`	

The `StreamHandleID` attribute SHALL NOT be included. Its value will be set by the Coordinator.

The `RequestingUserID` and `RightsTokenID` elements are mandatory.

The `ExpirationDateTime` element SHOULD NOT be included. If included, its value will be discarded by the Coordinator.

Unlike other APIs, the `//ResourceStatus/Current` SHALL be included in the request. The `CreationDate` and `DeletionDate` attributes indicate the Streams start date/time and ending date/time. The `Value` element SHALL be set to `urn:dece:type:status:deleted`.

Response Body: None

11.1.6.2 Behavior

The `RightsTokenID` identified in the request body SHALL be for the content being reported.

Coordinator SHALL validate that the request body is syntactically valid, reported datetime range is valid and SHALL respond with an HTTP 202 status code (*Accepted*).

During post-processing, Coordinator SHALL validate that the User identified in the `RequestingUserID` and the RightsToken identified in the `RightsTokenID` match existing User and RightsToken resources respectively. Coordinator SHALL not validate the status of these resources or their current association with one another. Errors identified during post-processing are written to the logs.

Unlike the real-time StreamCreate API defined in Section 11.1.1, no resource will be created. The Coordinator simply records the Stream activity for reporting purposes.

The Coordinator will respond with a 503 status code (*Service Unavailable*) when the request cannot be fulfilled due to service availability failures. Nodes will have to retry the request at a later time.

11.1.7 Stream Visibility Rules

The following table describes the rules the Coordinator SHALL enforce to determine Stream visibility and access to Stream API calls.

Role	Stream Creator	Same Org.	MAC	StreamListView, Stream View		StreamRenew	StreamDelete
				Active	Deleted		
LASP/CS	YES	YES	N/A	●	●	●	●
	NO	YES		●	●		●
		NO	YES	●			
			NO				
Non-LASP/CS	NO	N/A	YES	●			
			NO				
Web Portal	N/A	N/A	N/A	●			

Legend

- Role
  - 'LASP/CS' designates LASP and the associated Customer Support Role.
  - 'Non-LASP/CS' represents Authorized Roles other than LASPs and LASP Customer Support Roles.
- 'Stream Creator' is whether or not the requesting Node is the Node that created the stream.
- 'Same Org.' indicates whether the requesting Node is in the same organization as the Stream Creator Node.
- 'MAC' refers to a granted Manage Account Consent.
- 'N/A' means the condition is not applicable.

Notes

- A 'Stream Creator' is implicitly in the 'Same Org.'
- Non-LASPs cannot be Stream Creators
- Streams reported via the Batch Stream Reporting are NOT returned in StreamListView API

11.2 Stream Types

11.2.1 StreamList Definition

The StreamList element describes a list of Streams. Streams are bound to Accounts, not to Users.

Element	Attribute	Definition	Value	Card.
StreamList			dece:StreamList-type	
	ActiveStreamCount	Number of active streams	xs:int	0..1
	AvailableStreams	Number of additional streams possible	xs:int	0..1
Stream			dece:Stream-type	0..n

Table 55: StreamList Definition

11.2.2 Stream Definition

The Stream element describes a stream, which may be active or inactive.

Element	Attribute	Definition	Value	Card.
Stream			dece:Stream-type	
	StreamHandleID	Unique identifier for the stream. It is unique to the Account, so it does not need to be handled as an identifier. The Coordinator must ensure it is unique.	dece:EntityID-type	0..1
StreamClientNickname		An optional human readable string representing the customer's stream client that may be used to aid a User or Customer Support function.	xs:string	0..1
RequestingUserID		The User that initated the Stream.	dece:EntityID-type	0..1

Element	Attribute	Definition	Value	Card.
RightsTokenID		Identifier of the RightsToken that holds the asset being streamed. This provides information about what stream is in use (particularly for customer support)	`dece:EntityID-type`	
TransactionID		Transaction information provided by the LASP to identify its transaction associated with this stream. A TransactionID need not be unique to a particular stream (that is, a transaction may span multiple streams). Its use is at the discretion of the LASP	`xs:string`	0..1
ExpirationDateTime			`xs:dateTime`	0..1
SubDividedGeolocation		Identifies an approximate geographic location of the stream client, when available.	`dece:SubDividedGeolocation-type`	0..1
ResourceStatus		Whether or not stream is considered active (that is, against the count).	`dece:ElementStatus-type`	0..1

Table 56: Stream Definition

11.3 Stream Status Transitions

The possible Status values are: `active` and `deleted`.

12 Account Delegation

12.1 Types of Delegations

Account delegation (or "linking") is the process of granting Nodes access to certain information from the Coordinator on behalf of Users without an explicit Coordinator login. LASPs (both Linked and Dynamic), Access Portal and Retailers are able to request such delegation.

The policy classes defined in section 5.5 enable specific APIs for the Node or Nodes identified in the Policy. These privileges are identified by consent policies established at the Account and User levels. Delegations are obtained by establishing a Delegation Security Token between the Coordinator and the Node or Nodes, as specified in [DSecMech].

In order for a Node to demonstrate that delegation has occurred, it SHALL present the Delegation Security Token using the REST binding and Delegation Security Token profile specified in [DSecMech].

Delegations occur between Nodes and the Coordinator, and may either be at the Account level, or the User level, depending on the Role of the Node being linked. These linkages may be revoked, at any time, by the User or the Node. The appropriate Delegation Security Token Profile defined in [DSecMech] SHALL specify the mechanisms for the creation and revocation of these delegations.

Nodes MAY be notified using the Delegation Security Token specific mechanism when a link is deleted, but Nodes should assume delegations may be revoked at any time and gracefully handle error messages when attempting to access a previously linked User or Account.

The Coordinator provides interfaces are provided to facilitate the collection of consent and the provisioning of Policies within the Coordinator.

LASPs (both Linked and Dynamic), Access Portal and Retailers SHALL support at least one Delegation Security Token profile defined in [DSecMech]. Support for the UserValidationTokenCreate API method defined in section 14.1.7.4 is optional for these Roles.

12.1.1 Delegation for Rights Locker Access

Retailers, Dynamic LASPs and Linked LASPs can be granted the right to access an Account's Rights Locker. The default access is for a Retailer Node to only have access to Rights tokens created by that Retailer Node. A LASP Node always has rights to all Rights Tokens (although with restricted detail). For example, if Retailer X creates Rights token X1 and Retailer Y creates Rights token Y1, X can only access X1 and Y can only access Y1.

Policies, established by a full-access user, enable a Retailer Node to obtain access to the entire Rights Locker, governed by the scope of the Security Token issued. The Authorization Matrix provided in

[7] Allowed to invoke the URI endpoint that excludes the {AccountID} parameter.

Table 37 details the nature of the policies which control the visibility of rights tokens in the Rights Locker. Linked LASPs (role: `urn:dece:role:lasp:linked`) only link at the Account level, and have limited access to the entire Rights Locker as detailed in the matrix.

Access shall be granted in the context of specific Users associated with the Delegation Security Token for retailers and DSPs This is established through policies established at the Coordinator at both the User and Account level. Rights Tokens which include ViewControl settings remain unavailable to Users who are not identified within the Rights Tokens. More specifically, if a User is not included in the list of AllowedUser elements, Rights tokens with that User will not be visible to the Node. In the case where the AllowedUser list is null, Rights tokens Access Rights SHALL be accessible to all users.

12.1.2 Delegation for Account and User Administration

The Coordinator allows Nodes to create and administer Users in an Account when those Nodes have both `urn:dece:type:policy:ManageAccount` and `urn:dece:type:policy:EnableManageUserConsent` policies enabled, and one or more Users within the Account have enabled the `urn:dece:type:policy:ManageUserConsent` policy.

12.1.3 Delegation for Linked LASPs

The Linked LASP linking process allows a Linked LASP to stream Content for an Account without requiring a User to login on the LASP Client receiving the stream. Linked LASP delegation differs from other delegations only in that:

There is a limit to the number of Linked LASPs associated with an Account as specified in [DSystem] Section 16.

Delegation Security Tokens are evaluated at the Account level (as apposed to the User level, as with most Security Token uses)

The lifespan of a Delegation Security Token to a Linked LASP is effectively unbounded. Delegation Security Token profiles specify the actual longevity, and the lifespan must be present in the Delegation Security Token itself.

The effect of Account level policy evaluation of Delegation Security Tokens during API invocation eliminates the incorporation of any User level Policies within the Account. For example, Parental Control and ManageUserConsent policies are not consulted by the Coordinator, and will therefore have no

influence on the construction of the response to the API request. Section 5.5.2 specifies the User level policies that would be ignored in these circumstances.

Linked LASPs, like dynamic LASPs, are not assumed to have a license to all DECE content, so not everything in the Rights Locker will be streamable.

12.2 Initiating a Delegation

To initiate a delegation and establish a Delegation Security Token between the Node and the Coordinator, Nodes shall utilize the Delegation Security Token specific mechanisms defined in [DSecMech] or as defined in this section. Currently defined Delegation Security Token Profiles require that Nodes initiate the link. That is, delegations cannot be initiated by the Web Portal, because the Web Portal does not maintain lists of Nodes.

12.3 Revoking a Delegation

Users and Nodes may revoke a delegation at any time, and mechanisms should be provided both by the Node, as well as the Web Portal. Delegation token profiles specified in [DSecMech] shall specify one or more mechanisms to provide for revocation of delegations initiated by either party.

A delegation SHALL be revocable at any time by User request through the Web Portal. Nodes may provide a mechanism for a User to request revocation of delegations.

12.3.1 Authorization

Upon linking, the Coordinator provides the Node with an appropriate Delegation Security Token, as defined in [DSecMech] that can subsequently be used to access Coordinator APIs on behalf of the User. The Coordinator SHALL verify that the Delegation Security Token presented to the API is well-formed, valid, and issued to the Node presenting the token. If the presented token is invalid, the Coordinator shall respond with an error response appropriate for the token employed, and defined in the token profile of [DSecMech].

13 Accounts

An Account represents a group of system Users, and their ability to access Rights Tokens in the Account's Rights Locker. The conventional model for an Account is a family living under the same roof, but in fact an Account's Users may be unrelated and geographically dispersed.

The maximum allowed `active` User count is determined by the defined Ecosystem parameter ACCOUNT_USER_LIMIT (specified in [DSystem] section 16). Users which are in `deleted`, `mergedeleted`, `forcedeleted or deidentified` status SHALL NOT be considered when calculating the total number of users within an Account.

The Account object maintains information about the DisplayName and Country for the Account, as well as its status. It is also the resource to which the account-level policies, discussed in section 5.5.1 are applied.

Unless otherwise noted, APIs evaluated at the Account level SHALL be rejected when the targeted Account's status is not `active`. Note that RightsTokenCreate() MAY be invoked for an Account with `pending` status as documented under that API.

13.1 Account Functions

The Account functions ensure that an Account is always in a valid state. The AccountUserCreate function creates the Account and the Rights Locker along with the first user. Several Account creation use cases begin with a user's identification of content to be acquired. Invocation of the AccountUserCreate API is then followed by the user obtaining a Rights Token (that is, invocation the RightsTokenCreate API).

Once created, an Account cannot be directly removed from the system by invoking an API. Instead the AccountDelete API changes the status of the Account to `urn:dece:type:status:deleted`. This allows Account deletion to be reversed (by changing the Account status to `urn:dece:type:status:active`). The status of the associated resources (such as Rights Tokens and Users) remains unchanged. Furthermore, the Account SHALL be considered `active` when it is in any status other that `deleted`, `forcedeleted`, `mergedeleted or deidentified`.

During its lifecycle, an Account's status undergoes changes from one status to another (for example, from `urn:dece:type:status:pending` to `urn:dece:type:status:active`). The Status element (in the ResourceStatus element) may have the following values.

Account Status	Description
`urn:dece:type:status:active`	Account is active (the normal condition for an Account)
`urn:dece:type:status:archived`	Account is inactive but remains in the database

urn:dece:type:status:blocked	Account has been blocked, possibly for an administrative reason
urn:dece:type:status:deleted	Account has been deleted
urn:dece:type:status:forcedeleted	An administrative delete was performed on the Account.
urn:dece:type:status:other	Account is in a non-active, but undefined state
urn:dece:type:status:pending	Account is pending but not fully created
urn:dece:type:status:mergedeleted	Indicates that the resource was force deleted as part of the merge process
urn:dece:type:status:suspended	Account has been suspended for some reason
urn:dece:type:status:deidentified	Indicates the resource was deidentified to remove PII.

Table 57: Account Status Enumeration

The possible Status values are: `active`, `pending`, `deleted`, `forcedeleted`, `blocked`, `suspended` , `mergedeleted, and deidentified`.

13.1.1 Inactive and Userless Accounts:

Accounts for which no Users are created shall be removed after the age of the Account exceeds DCOORD_USERLESS_ACCOUNTS_THRESHOLD..

An Account which has acquired Rights Tokens but has had no new Rights Tokens added or removed and had no streaming activity for a period of DCOORD_INACTIVITY_THRESHOLD is considered to be inactive. Users (who are not deleted, forcedeleted, mergedeleted, deidentified) associated with this type of inactive Account shall be notified by email with an explanation of how the Account may be closed. Such Accounts are not deleted, however.

13.1.2 Account De-Identification Process:

An Account which has been in status `deleted`, `forcedeleted`, or `mergedeleted` for a period of DCOORD_DEIDENTIFY_ACCOUNT_THRESHOLD or longer shall be modified to have all personally identifiable information removed from the Account and from all Users in the Account. The following adjustments are made to the Account:

- //Account/DisplayName shall be changed to DCOORD_DEIDENTIFIED
- //Account/ResourceStatus/Current/value shall be changed to deidentified

In addition to the account resource being de-identified, all the users under the account will be de-identified per Section 14.1.1.

13.1.3 Periodic removal of test accounts and related data

The Coordinator shall reserve portions of the namespaces for Accounts, Usernames, and User email addresses as indicators of test Accounts. An Account, Username, or User email address which begins with DCOORD_TEST_INDICATOR may result in an Account, and all related data, being physically removed by the Coordinator. The criteria for removal are:

1. Account DisplayName begins with DCOORD_TEST_INDICATOR
2. Username begins with DCOORD_TEST_INDICATOR
3. User email address begins with DCOORD_TEST_INDICATOR

An Account, and all related data, may be removed by the Coordinator if the combination of (1) together with either of (2) or (3) applies to that Account.

A Coordinator batch process will run regularly to remove these accounts.

Nodes that create test Accounts and Users but wish to ensure the Coordinator does not delete them may use the DCOORD_TEST_PRESERVE_INDICATOR prefix for the DisplayName of the Account.

13.1.4 AccountCreate()

This API call is retired and AccountUserCreate() (see section 13.1.8) SHALL be used instead.

13.1.5 AccountUpdate()

13.1.5.1 API Description

The AccountUpdate API is used to update an Account entry. The AccountUpdate API can be used to modify the Account's DisplayName and Country properties when the Web Portal role is composed with a full-access user access level. Account data can be also be updated by Nodes on behalf of a properly authenticated full-access User. The Coordinator SHALL generate an e-mail notice to all full-access Users indicating that the Account has been updated.

13.1.5.2 API Details

Path:

```
[BaseURL]/Account/{AccountID}
```

Method: PUT

Authorized Roles:

```
urn:dece:role:dece:customersupport
urn:dece:role:coordinator:customersupport
urn:dece:role:lasp[:customersupport]
urn:dece:role:portal[:customersupport]
urn:dece:role:retailer[:customersupport]
```

Request Parameters: AccountID is the unique identifier for an Account

Request Body: Account

Element	Attribute	Definition	Value	Card.
Account			dece:Account-type	

Security Token Subject Scope: urn:dece:role:user:class:full

Opt-in Policy Requirements:

urn:dece:type:policy:ManageAccountConsent

Response Body: None

13.1.5.3 Behavior

The AccountUpdate can be used to modify the Account's DisplayName and Country properties when the Web Portal role is composed with a full-access user access Level.

13.1.6 AccountDelete()

13.1.6.1 API Description

The AccountDelete API deletes an Account. It changes the status of the Account to urn:dece:type:status:deleted. This allows Account deletion to be reversed (by changing the Account status to urn:dece:type:status:active). None of the statuses of any of the Account's associated elements (for example, Users or Rights Tokens) SHALL be changed.

Account deletion may be initiated only by a full-access User belonging to that Account. This has the effect of making the Account delete reversible (that is, it is possible to return the Account's status to urn:dece:type:status:active). In order for any resource within an Account to be considered active (or any other non-deleted status), the Account SHALL be active.

When Account deletion has been completed, any outstanding Security Tokens issued to any and all Users belonging to the deleted Account are invalidated.

13.1.6.2 API Details

Path:

```
[BaseURL]/Account/{AccountID}
```

Method: DELETE

Authorized Roles:

```
urn:dece:role:coordinator:customersupport
urn:dece:role:dece:customersupport
urn:dece:role:lasp[:customersupport]
urn:dece:role:portal[:customersupport]
urn:dece:role:retailer[:customersupport]
```

Request Parameters: AccountID is the unique identifier for an Account

Request Body: None

Response Body: None

Security Token Subject Scope: `urn:dece:role:user:class:full`

Opt-in Policy Requirements:

`urn:dece:type:policy:ManageAccountConsent`

13.1.6.3 Behavior

AccountDelete updates the status to *deleted*. Nothing else is modified. Upon invocation of AccountDelete(), the Coordinator SHALL invalidate all Security Tokens associated with the Account's Users. The Coordinator MAY send Security Token revocation requests, as defined for the applicable Security Token Profile, to the Nodes associated with these Security Tokens.

The Coordinator SHALL provide e-mail notification to all Full Access Users in the Account indicating that the Account has been deleted.

Additional email notifications will additionally result as a side effect of the deletion of each User in the Account (see section 14.1.6).

13.1.7 AccountGet()

13.1.7.1 API Description

This API is used to retrieve Account descriptive information.

13.1.7.2 API Details

As with many Coordinator GET operations, the entire XML object is returned to the requesting API Client.

Path:

```
[BaseURL]/Account/[{AccountID}]
```

Method: GET

Authorized Roles:

```
urn:dece:role:accessportal[:customersupport]
urn:dece:role:dece[:customersupport]
urn:dece:role:coordinator:customersupport
urn:dece:role:dece
urn:dece:role:lasp[:customersupport]
urn:dece:role:portal[:customersupport]
urn:dece:role:retailer[:customersupport]
```

Request Parameters: AccountID is the unique identifier for an Account (optional)

Security Token Subject Scope: `urn:dece:role:user`

Opt-in Policy Requirements:

None

Request Body: None

Response Body: Account

Element	Attribute	Definition	Value	Card.
Account			`dece:Account-type`	1

13.1.7.3 Behavior

The GET request has no parameters and returns the Account object.

13.1.8 AccountUserCreate()

13.1.8.1 API Description

This API call is used to create both an Account and its first User. Additionally, it allows the creation of User-level policies TermsOfUse, UserLinkConsent and ManageUserConsent as part of the composite call.

13.1.8.2 API Details

Path:

```
[BaseURL]/Account
```

Method: POST

Authorized role:

```
urn:dece:role:coordinator:customersupport
urn:dece:role:dece:customersupport
urn:dece:role:lasp:dynamic[:customersupport]
urn:dece:role:lasp:linked[:customersupport]
urn:dece:role:portal[:customersupport]
urn:dece:role:retailer[:customersupport]
```

Request Parameters: None

Request Body:

Element	Attribute	Definition	Value	Card.
Account (including a single User in the UserList element) (may include policylist inside the UserList element)			dece:Account-type	1

Response Body: None

Security Token Subject Scope: None

Opt-in Policy Requirements: None

Response Body: None

13.1.8.3 Behavior

AccountUserCreate creates both the Account, the necessary Rights Lockers and the first User in the Account. Additionally, the user-level consents TermsOfUse, UserLinkConsent and ManageUserConsent can be created, if these policies are included as part of the request body. Upon successful creation, an HTTP Location header in the response provides a reference to the newly created User (this resource URL contains both the identifier of the User and the Account).

The Account SHALL be set to `pending` and the User status SHALL be set to `blocked:tou` until the `termsofuse` policy is created, at which point both resources are set to `active`.

Nodes SHALL supply a value for the //Account/DisplayName and //Account/Country. Nodes MAY utilize the initial User's //User/GivenName value or the initial User's Username value for the DisplayName.The provided Account SHALL also contain a UserList element with a single User (not by reference). The User data required is the same as for a regular UserCreate().

Nodes MAY use the UserID attribute of the <User> element to provide the NodeSpecifiedUserID. NodeSpecifiedUserID is defined as a set of hexadecimal characters of length 32 characters. The value may be padded on the left with zeroes to account for the the required length. If the supplied value is less than 32 characters, it will rejected by the Coordinator. The UserID SHALL be unique for the Node. If a Node submits a UserID that collides with an existing UserID (for that Node) then the Coordinator SHALL return an appropriate error.

The relevant policies SHALL be enforced by the Coordinator.

The Account-level policy ManageAccountConsent is automatically set to TRUE, and applied to the Account, to facilitate the creation of the first User.If ManageUserConsent policy is requested as part of the composite operation, then Coordinator automatically sets the Account-level policy ManageAccountConsent for the Account (for the same Node or Org)

Nodes may request the creation of certain user-level consents as part of the request body. A general constraint is that TOU must be set prior to any other consent being set. As long as TermsOfUse policy is included in the request, this contraint is satisfied.

Note: Requests containing only the UserLinkConsent policy and/or ManageUserConsent but not the TermsOfUse policy are not permitted and Coordinator shall respond with an appropriate error. Other User-level consents or Account-level consents shall not be included in the request.

As the UserID is not known at the time of the request, Nodes SHALL NOT set the RequestingEntity for TermsOfUse policy and Resource for UserLinkConsent and ManageUserConsent policies.

Multiple RequestingEntities can be set for ManageUserConsent policy.

The Coordinator SHALL ensure the atomicity of this API call. In other words, the only possible outcomes of this operation are that either a new Account and its first User (and policies, if requested) have been created, or no resource was created as a result of the API call.

Note: An Account DisplayName, Username, or User email address which begins with DCOORD_TEST_INDICATOR may result in an Account, and all related data, being physically removed by the Coordinator. Refer to section 13.1.3 for more information.

13.2 Merging Accounts

The Coordinator provides two special APIs, AccountMergeTest() and AccountMerge() that together provide the ability to merge two distinct Accounts into one Account.

The merge process involves two Accounts:

- The Surviving Account (the Account that will be merged into, and will remain active after the merge has been completed),
- The Retired Account (the Account that resources will be copied from, into the Surviving Account, and will be deleted after the merge has been completed)

During the merge process, the Account FAUs choose which account is the Surviving Account, and which is the Retired Account..

13.2.1 Basic Process for Performing a Merge

The following sequence defines the merge process.

1. Authentication and Acknowledgement.
    - Full Access User (FAU) 1 in one Account authenticates to the Node, and indicates the intention to merge with a second Account (which Account is unknown at this stage).
    - The Node indicates to FAU 1 that this process is irreversible and the User must acknowledge that they want to proceed.
    - Within the same browser, FAU 2 in the other Account authenticates to the same Node.

- The Node indicates to FAU 2 that the merge process is irreversible and the User must acknowledge that they want to proceed.

2. Merge Choices.
   The following proceeds until the User has selected a merge scenario that is valid or the User aborts the merge process.

   - The Node provides the User the ability to identify the following (the merge scenario)

     - Which User(s) will be retained (at least one of FAU1 and FAU2 MUST be retained).

   - The Node allows the User to review the contents of each Account, and warns the User of any potential issues that may prevent a successful merge (for example, exceeding ACCOUNT_USER_LIMIT).

   - The Node performs the AccountMergeTest API with the two Accounts to confirm the merge can complete successfully or identify errors.

   - If any errors occur, the Node indicates the required corrective action(s) to the FAUs, and allows the User to return to defining the merge scenario.

3. The Node indicates to the FAUs that the merge can now be performed (and is irreversible) and receives final confirmation.

4. The Node invokes the AccountMerge API

5. The Coordinator determines whether the Accounts can be merged. This is essentially equivalent to AccountMergeTest.

6. If the merge is valid, the Coordinator performs the following actions on resources

   - All the Rights Tokens are moved from the Retired Account to the Surviving Account.

   - DiscreteMediaRights are copied along with corresponding Rights Tokens, including existing DiscreteMediaRight leases. This allows the lease timing and other factors to be retained properly. When a lease is moved to the Surviving Account, the previous lease resource location will no longer be available, nor will the associated Delegation Security Token be active. However, when an attempt is made to renew, release or consume a lease, the Coordinator will respond with the `SecTokenMergeReplacementRequired` error. This will indicate to the Node that the DiscreteMediaRight has moved (in addition to the need to obtain a replacement

Security Token). The corresponding {DiscreteMediaRightID} Resource URL parameter will remain unchanged after the Account Merge has completed, however, the {AccountID} parameter will reflect the AccountID of the Surviving Account.

- The retained Users in the Retired Account are moved to the Surviving Account.
- Users in the Surviving Account that are to be removed have their statuses updated to `urn:dece:type:status:mergedeleted`.
- Users in the Retired Account that are to be removed have their AccountID changed to the Surviving Account's ID. These Users are then deleted (using UserDelete()) and their statuses updated to `urn:dece:type:status:mergedeleted`.
- If set in the Retired Account, the `urn:dece:type:policy:ManageAccountConsent` policy SHALL be carried over to the Surviving Account in an `Active` status.
- Active Streams from the Retired Account have their statuses updated to `urn:dece:type:status:deleted`.
- The Coordinator performs an AccountDelete on the Retired Account and updates the Account Status to `urn:dece:type:status:mergedeleted`.

7. If the merge is valid,
    - The Node acquires fresh Delegation Security Tokens for all Users that were moved from the Retired Account to the Surviving Account. This is necessary because the `AccountID` and `UserID`s for the moved Users will have changed (note that all consent policies will be preserved during the merge process).

13.2.2 Common Requirements for Account Merge APIs

Merging involves the combination of resources of two Accounts. This includes Users and Rights. Policies from the Surviving Account are retained while Policies of each remaining User are retained regardless of which Account they were from.

The merge process SHALL require that at least one of the two Users represented by the presented Delegation Security Tokens remains active in the Surviving Account.

The merge process SHALL copy the entire Rights Locker. All Rights Tokens are maintained, regardless of whether the Account already has Rights for a given Logical Asset (ALID).

The merge process SHALL invalidate all outstanding Delegation Security Tokens for all Users from the Retired Account.

For Users that are moved from the Retired Account to the Surviving Account, the merge process SHALL copy all `active` Policies associated with said Users. This includes both consent Policies as well as Parental Control Policies.

Users whose status is `deleted`, `forcedeleted` or `mergedeleted` NEED NOT be included in the `//AccountMerge/UserReference` element. If included, the Coordinator SHALL ignore those and not moved them to the Surviving Account.

The outcome of the merge SHALL be a fully valid Account (that is, it meets all of the requirements for being a valid Account).

The merge process SHALL NOT be performed unless the countries of the Accounts associated with the merge are identical (e.g. the `/Account/Country` values match).

Merge SHALL comply with any Geography-specific constraints and requirements as defined in [DGeo]. Geography requirements may prohibit the movement of Users below the DGEO_CHILDUSER_AGE. This may occur when geo-political systems prohibit such an action. Moving such Users will require manual re-entry of the child Users into the Surviving Account.

Users under the DGEO_CHILDUSER_AGE who have an associated Connected Legal Guardian (see section 5.5.2.5) SHALL NOT be moved to the Surviving Account unless the Connected Legal Guardian is also moved to the Surviving Account.

Outstanding streams in the Retired Account SHALL be deleted.

Delegation Security Tokens presented by Customer Support Roles SHALL be evaluated at the User-level for the Account Merge API methods.

13.2.3 AccountMergeTest()

13.2.3.1 API Description

Provides a mechanism to allow a Node to test the validity of the merge of two Accounts prior to performing a final merge of those Accounts by proposing a new merged Account. If the new Account would be valid, the invocation is successful. If the new Account would be invalid, error conditions are returned to instruct the Node regarding what changes are necessary. For example, the resulting number of Users meet ecosystem parameter restrictions. Furthermore, if all required preconditions are not met, an error response will indicate which required preconditions were not met.

If AccountMergeTest() succeeds, and nothing has changed, it should be expected that AccountMerge() will be successful.

13.2.3.2 API Details

Path:

```
[BaseURL]/Account/{SurvivingAccountID}/Merge/Test/{RetiredAccountID}
```

Method: POST

Authorized Roles:

urn:dece:role:dece:customersupport
urn:dece:role:coordinator:customersupport
urn:dece:role:portal[:customersupport]
urn:dece:role:retailer[:customersupport]
urn:dece:role:lasp[:customersupport]
urn:dece:role:accessportal[:customersupport]

Node-based Access Control: Yes

Request Parameters:

SurvivingAccountID is the unique identifier for the Account that will be merged into
RetiredAccountID is the unique identifier for an Account that will be merged into the SurvivingAccountID

Security Token Subject Scope:

urn:dece:role:user:class:full (see section 13.2.6)

Opt-In Policy Requirements:

urn:dece:type:policy:ManageAccountConsent

Request Body: AccountMerge

Response Body: None or ErrorList

Element	Attribute	Definition	Value	Card.
AccountMerge			dece:AccountMerge-type	

13.2.3.3 Request Behavior

The Node SHALL have a Delegation Security Token for both Users involved in the merge process. The incorporation of two Delegation Security Tokens into this API request differs from a normal API invocation, as two Users are involved in the process. See section 13.2.6 for details. The Node SHALL present the two Delegation Security Tokens for authentication within the time period specified by DCOORD_MERGE_SESSION_AGE.

The request SHALL include an `AccountMerge` resource that represents the desired Coordinator actions to perform to complete the merge. This will include:

- An enumeration of each User in both Accounts, as `UserReference` elements, indicating the requested `ResourceDisposition` for each User after the merge (that is, indicating which Users to keep, and which Users to delete via the `StatusValue` element).

The following `StatusValue` values may be used for the Users in the merge request:

- `urn:dece:type:status:Active` : indicates that the resource should be preserved after the merge.
- `urn:dece:type:status:mergedeleted` : indicates that the resource should be force deleted as part of the merge process.

13.2.3.4 Response Behavior

The Coordinator will evaluate the submission to ensure the results of the request will result in a fully compliant Account. If the request does not meet the requirements provided in section **13.2** an ErrorList response will be returned, indicating with the following error codes what actions are required in order to complete the merge successfully.

The HTTP response status `200 OK` will signal a successful test.

In addition to normal API failures, the following errors are particular to the merge process:

- AccountActiveUserCountReachedMaxLimit : the resulting number of Users will exceed the ACCOUNT_USER_LIMIT. Error will be of form: "`AccountActiveUserCountReachedMaxLimit:`" + <userexceeded> where <userexceeded> is the number of users in excess of ACCOUNT_USER_LIMIT.
- AccountUserAgeRequirementNotMet : a User remains in the Account who cannot be moved as a result of a restriction on `Country` of the Accounts. For example, when a Child User moves

without their associated Connected Legal Guardian. Error will be of form: "AccountUserAgeRequirementNotMet:" + <userID> where <userID> is the User that caused the error condition. There can be multiple instances.

- SameAccount : SurvivingAccountID refers to the same Account as RetiredAccountID. A Merge can only be performed between two distinct Accounts.

An example of an AccountMergeTest submission:

```
<AccountMerge xmlns="http://www.decellc.org/schema/2015/03/coordinator">

<!-- Proposed Merged User actions -->
<UserList>
   <!-- delete this User as part of the Merge action -->
   <UserReference ResourceDisposition="urn:dece:type:status:mergedeleted">
      urn:dece:userid:user1fromaccountB
   </UserReference>

   <!-- retain this User as part of the Merge action -->
   <UserReference ResourceDisposition="urn:dece:type:status:active">
       urn:dece:userid:user2fromaccountB
   </UserReference>

   <!-- retain this User as part of the Merge action -->
   <UserReference ResourceDisposition="urn:dece:type:status:active">
      urn:dece:userid:user3fromaccountA
   </UserReference>

   <!-- delete this User as part of the Merge action -->
   <UserReference ResourceDisposition="urn:dece:type:status:mergedeleted">
       urn:dece:userid:user2fromaccountA
   </UserReference>
</UserList>

</AccountMerge>
```

13.2.4 AccountMerge()

13.2.4.1 API Description

Provides a mechanism to allow a Node to perform a final merge of two Accounts. The outcome of this merge is a single unified Account containing all of the resources of both Accounts based on the instruction set of the API invocation. The submission process is identical to AccountMergeTest.

13.2.4.2 API Details

Path:

```
[BaseURL]/Account/{SurvivingAccountID}/Merge/{RetiredAccountID}
```

Method: POST

Authorized Roles:

urn:dece:role:dece:customersupport
urn:dece:role:coordinator:customersupport
urn:dece:role:portal[:customersupport]
urn:dece:role:retailer[:customersupport]
urn:dece:role:lasp[:customersupport]
urn:dece:role:accessportal[:customersupport]

Node-based Access Control: Yes. Nodes SHALL NOT use this API without permission from DECE. Note: Node-based Access Control can be policy-based or Coordinator-enforced.

Request Parameters:

`SurvivingAccountID` is the unique identifier for the Account that will be merged into

`RetiredAccountID` is the unique identifier for an Account that will be merged into the `SurvivingAccountID`

Security Token Subject Scope:

urn:dece:role:user:class:full (see section 13.2.6)

Opt-In Policy Requirements:

`urn:dece:type:policy:ManageAccountConsent`

Request Body: `AccountMerge`

Response Body: None or `ErrorList`

13.2.4.3 Request Behavior

A Node SHALL inform the User that Account Merge is irreversible and obtain acknowledgement prior to invoking AccountMerge().

A Node SHOULD have already performed a successful AccountMergeTest() prior to the use of this API.

The Node SHALL have a Delegation Security Token for both Users involved in the merge process. The incorporation of two Delegation Security Tokens into this API request differs from a normal API invocation, as two Users are involved in the process. See section 13.2.6 for details. The Node SHALL present the two Delegation Security Tokens for authentication within the time period specified by DCOORD_MERGE_SESSION_AGE.

13.2.4.4 Response Behavior

AccountMerge() performs all tests of AccountMergeTest() prior to making any changes. If there are any error conditions resulting from these tests, no changes are made to either Account and error conditions are returned as they would be for AccountMergeTest(). If successful, the Coordinator SHALL create a `dece:AccountMergeRecord` resource in the Surviving Account to document the changes done in both Accounts.

The Account is modified in accordance with requirements in Section **13.2**.

If the merge is successfully performed, an HTTP `200 OK` status response (with no body) will be returned.

If the merge cannot be successfully performed, an HTTP `403 Forbidden` status response with a complete ErrorList body will be returned. The ErrorList will detail all of the pre-conditions that must be met to achieve a successful merge.

Any error returned by AccountMergeTest() can also be returned by AccountMerge().

13.2.5 AccountMergeUndo()

API Description

This API allows a Merge to be undone given constraints. This API is only available to Customer Support sub Roles. AccountMergeUndo() SHALL NOT be allowed once any change has been made to the Surviving Account. Examples of changes are new or updated Users and new or updated Rights Tokens.

API Details

Path:

```
[BaseURL]/Account/{SurvivingAccountID}/Merge/Undo
```

Method: POST

Authorized Roles:

```
urn:dece:role:dece:customersupport
urn:dece:role:coordinator:customersupport
urn:dece:role:portal:customersupport
urn:dece:role:retailer:customersupport
urn:dece:role:lasp:customersupport
urn:dece:role:accessportal:customersupport
```

Node-based Access Control: Yes. Nodes SHALL NOT use this API without permission from DECE. Note: Node-based Access Control can be policy-based or Coordinator-enforced.

Request Parameters:

`SurvivingAccountID` is the unique identifier for the Account that was merged into.

Security Token Subject Scope:

urn:dece:role:user:class:full (see section 13.2.6)

Note: Security Tokens presented by Customer Support Nodes are usually evaluated at the Account level. This API is an exception to that.

Opt-In Policy Requirements:

`urn:dece:type:policy:ManageAccountConsent`

Request Body: `None`

Response Body: None or `ErrorList`

Request Behavior

The Node SHALL have a Delegation Security Token for a Full Access User in the Surviving Account.

Response Behavior

MergeUndo occurs on the most recent Merge as indicated by most recent MergeRecord/DateTimeofMerge element.

The Coordinator SHALL NOT allow a Merge Undo beyond the earlier of either:

- The date calculated by adding DCOORD_MERGE_UNDO_PERIOD to the Merge date.
- If present, the `UndoExpiration` date attribute of the relevant MergeRecord resource.

The Coordinator SHALL move `active` Users from the Retired Account to the Restored Account, based on the `MergeRecord/MovedUserReference` elements. Deleted UserLinkConsents, ManageUserConsents and UserDataUsageConsents are not restored.

The MergeUndo process SHALL invalidate all outstanding Delegation Security Tokens for all Users from the Surviving Account that are returned to the Retired Account.

The Coordinator SHALL return Rights Tokens from the Retired Account back to the Restored Account. This will be done based on the `RightsPurchaseInfo/PurchaseAccount` element of the Rights Token.

The Coordinator SHALL change the state of the Restored Account to `active`.

The HTTP response status `200 OK` will signal a successful Merge Undo.

In addition to normal API failures, the following errors are particular to the merge undo process:

- MergeUndoTimeLimitExceeded: More time has elapsed since the Merge than DCOORD_MERGE_UNDO_PERIOD (or, if present, when the `UndoExpiration` date attribute has passed).
- UndoDoesNotMeetPolicy: Defined policies does not meet Undo policies.
- SurvivingAccountHasBeenModified: changes have been made to the Surviving Account since the Merge happened.

13.2.6 Special Requirements for Security Tokens for Merge

Because the merge APIs require two Users to be involved in the transaction, both Delegation Security Tokens SHALL be provided in the HTTP header. This is accomplished by including the same HTTP header parameter twice, one for each Delegation Token, unless defined otherwise by the Delegation Security Token Profile.

For example, for the SAML Token Profile defined in [DSecMech], a Node includes two HTTP `Authorization` headers to include both Delegation Security Tokens.

Users who were in the Retired Account will have all outstanding Delegation Security Tokens revoked (to all Nodes). The Security Token Service defined in section 8 of [DSecMech] provides a special allowance to facilitate the exchange of Delegation Security Tokens for Users of Retired Accounts. This allowance is also extended to Users moved back to the Retired Account subsequent to an AccountMergeUndo.

All applicable APIs will support the Error Code `SecTokenMergeReplacementRequired` which is exclusively used to indicate that the Delegation Security Token Service must be used to exchange an old Delegation Security Token with a new one due to a merge event.

13.3 Account-type Definition

The Account-type data element is the top-level element for an Account and is identified by an AccountID. The AccountID is created by the Coordinator, and is of type `dece:EntityID-type`. Its content is left to implementation, although it SHALL be unique within a particular Coordinator-Node context.

Element	Attribute	Definition	Value	Card.
Account			`dece:Account-type`	
	AccountID	Unique identifier for an Account	`dece:EntityID-type`	0..1
DisplayName		Display name for the Account If set as DCOORD_TEST_INDICATOR, then the Account may be removed by Coordinator. Refer Section 13.1.3	`xs:string`	
Country		Only authorized countries as defined in [DGeo] Section 2.2 SHALL be valid values for this element. The Coordinator validates this value and SHALL return an error if the Country value is not authorized or is invalid.	`dece:Country` (defined as `xs:string`)	
RightsLockerID		Reference to the Account's Rights Locker. Currently, only one Rights Locker is allowed.	`xs:anyURI`	0..n
UserList		A collection of Users associated with the Account (see **Table 78**)	`dece:UserList-type`	0..1
PolicyList		A collection of Account Consent policies (see section 5.4.1	`dece:PolicyList-type`	0..1
MergeRecord		Information about Merges into this Account. This is only returned to Nodes with the Role urn:dece:role:dece:customersupport, urn:dece:role:coordinator:customersupport	`dece:AccountMergeRecord-type`	0..n
ResourceStatus		Status of the Account resource (see section 17.2)	`dece:ElementStatus-type`	0..1

Table 58: Account-type Definition

13.3.1 AccountMerge-type definition

AccountMergeUser-type is used to express the changes initiated in an Account Merge.

Element	Attribute	Definition	Value	Card.
AccountMerge-type				
UserReference		The unique identifier of the User. May be from either Account.	`extends dece:EntityID-type`	1..n
	ResourceDisposition		`dece:StatusValue-type`	

Table 59: AccountMerge-type Definition

13.3.2 AccountMergeRecord-type definition

AccountMergeRecord-type captures Merge information needed to perform and Undo.

Element	Attribute	Definition	Value	Card.
AccountMergeRecord-type				
	AccountMergeRecordID	Unique identifier for the AccountMergeRecord	`dece:EntityID-type`	
	UndoPoliciesMet	Is this Merge eligible for Undo? The Coordinator determines if policies will allow the Undo or if other conditions would preclude Undo, and returns the appropriate value.	`xs:boolean`	
	UndoExpiration	The date and time when Undo will not be allowed anymore. Note that other factors beyond time may preclude Undo.	`xs:dateTime`	0..1
DateTimeofMerge		The date and time when merge was completed	`xs:dateTime`	
MergeNodeID		The Node that initiated the Merge	`dece:EntityID-type`	
RetiredAccount		AccountID of the Retired Account	`dece:EntityID-type`	
MergeActorSurviving		The User from the Surviving Account who performed the Merge (FAU 1).	`dece:EntityID-type`	

Element	Attribute	Definition	Value	Card.
MergeActorRetired		The User from the Retired Account who performed the Merge (FAU 2).	`dece:EntityID-type`	
MovedUserReference		References to Users moved during the Merge.	`dece:EntityID-type`	0..n
UndoDateTime		The date and time when Undo was performed. If this element is present, then an Undo has occurred and the record is maintained for historical purposes.	`xs:dateTime`	0..1

Table 60: AccountMergeRecord-type Definition

13.4 Account Status Transitions

The possible Status values are: `active`, `pending`, `deleted`, `forcedeleted`, `blocked`, `suspended` , `mergedeleted`, and deidentified

14 Users

The User object is a representation of a human end-user of the Coordinator. It allows the users certain privileges when accessing system data and resources in the DECE ecosystem. Users belong to an Account.

14.1 Common User Requirements

Users which are in a `deleted`, `mergedeleted`, `forcedeleted` or `deidentified` status shall not be considered when calculating the total number of users slots used within an Account for the purposes of determining the Account's User quota.

The maximum allowed `active` User count is determined by the defined Ecosystem parameter ACCOUNT_USER_LIMIT (specified in [DSystem] section 16). At no time shall the Coordinator retain more than this number of Users in an Account.

If the sole Full Access User in an Account is being deleted or their User Level is being changed, and there are additional Users in the Account, the Coordinator SHALL return an error status code of `urn:dece:errorid:org:dece:LastFullAccessUserofAccountCannotBeDeleted`. In response, the requesting Node SHOULD recommend to the User that a new Full-Access User be created or a Basic- or Standard-Access User be promoted to Full Access to allow deletion of the other Full-Access User.

Legal Guardians

Geography Policies (see Appendix F) SHALL define Legal Guardian requirements, if any, for Users below the DGEO_AGEOFMAJORITY and/or the DGEO_CHILDUSER_AGE. In order to support the transfer of Guardianship of such a User, the `LegalGuardian` element has a cardinality of 0..n. The `LegalGuardian` element defines an attribute `status`, which provides an indication of the current and intended transferee Legal Guardian. At no time shall there be more than one `active` LegalGuardian for a User under the DGEO_AGEOFMAJORITY, if such is required.

Child Users

[DGEO] policies may or may not allow the creation of Child Users

14.1.1 User De-Identification Process:

A user account which has been in status `deleted`, `forcedeleted`, or `mergedeleted` for a period of DCOORD_DEIDENTIFY_USER_THRESHOLD or longer shall be modified to have all personally identifiable information removed from the user account. The following adjustments are made to the User resource:

- //User/GivenName shall be changed to DCOORD_DEIDENTIFIED
- //User/SurName shall be changed to DCOORD_DEIDENTIFIED
- Username shall be changed to a unique anonymized value with prefix DCOORD_DEIDENTIFIED
- Password, if present, shall be changed to DE-IDENTIFIED
- DisplayImage, if present, shall be physically removed
- ContactInfo, if present, shall be changed to DCOORD_DEIDENTIFIED except for /Address/Country
- DateOfBirth, if present, shall be unchanged
- //User/ResourceStatus/Current/value shall be changed to deidentified
- Email address in UserVerificationTokens shall be changed to DCOORD_DEIDENTIFIED

14.1.2 User Functions

Users are only created at the Coordinator, unless the Account-level policy EnableManageUserConsent is set to TRUE, which allows Node management of a User resource.

14.1.3 UserCreate()

14.1.3.1 API Description

Users may be created using the Web Portal or by a Node (for example, a LASP, Access Portal, or Retailer) if the Account-level policy EnableManageUserConsent is set to TRUE.

Node SHALL inform the user that a User will be created, why it is being created, and that an email notification will follow.

14.1.3.2 API Details

Path:

```
[BaseURL]/Account/{AccountID}/User
```

Method: POST

Authorized Roles:

```
urn:dece:role:accessportal[:customersupport]
urn:dece:role:coordinator:customersupport
urn:dece:role:dece:customersupport
urn:dece:role:portal[:customersupport]
urn:dece:role:retailer[:customersupport]
urn:dece:role:lasp:dynamic[:customersupport]
urn:dece:role:lasp:linked[:customersupport]
```

Request Parameters: AccountID is the unique identifier for an Account

Security Token Subject Scope:

`urn:dece:role:user:class:standard`
`urn:dece:role:user:class:full`
(with the exception of the first user associated with an Account,
when the security context SHALL be NULL)

Opt-in Policy Requirements:

`urn:dece:type:policy:EnableManageUserConsent` on the Account resource, with the exception of the first User which does not require this consent

Request Body:

Element	Attribute	Definition	Value	Card.
User		Information about the user to be created.	`dece:UserData-type`	

Response Body:

If no error conditions occur, the Coordinator responds with an HTTP 201 status code (*Created*) and a Location header containing the URL of the created resource.

14.1.3.3 Behavior

The first User created in an Account SHALL be of UserClass `urn:dece:role:user:class:full`. The required security context for the first user created in association with an Account SHALL be NULL. EnableManageUserConsent is not required for the creation of the first User in an Account.

A User's primary E-mail address MAY be attested as confirmed by the Node submitting the transaction.

A similar confirmation MAY be performed every time a User's PrimaryEmail address is updated.
Note that whether a User's primary E-mail address is validated or not has no impact on the User's status.

A creating user may promote a created user only to the same user privilege level equal to or less than that of the creating user. By default, the Role for new Users shall be the same Role as the creating User. A different Role can be provided when invoking this method.

When an Account has reached the DCOORD_MAX_USERS limit, the Coordinator SHALL return an error. The number of Users in an Account is calculated based on the sum of all `active`, `pending`, `blocked` (`tou` and `clg`) and `suspended` Users.

Nodes SHALL NOT provide a DateOfBirth when creating a user. The `Password` element within the `UserCredentials` element may be omitted. If it is omitted, the Coordinator SHALL generate a random password with sufficient entropy to ensure randomness, incorporate that value as part of the newly created resource, and internally track that the User's password value was determined by the Coordinator by setting the `IsRandom` attribute on the `Password` element to TRUE.

This randomly generated password SHALL meet the syntax requirements detailed in [DSecMech] section 6, with the following constraints:

- The randomly generated password SHALL be no less than 12 characters in length.
- The randomly generated password SHALL only consist of the numeric values 0-9 (UTF8 0x30 – 0x39) and alphabetic characters a-z and A-Z (UTF8 0x41 – 0x5A and 0x61 – 0x7A),

The Node creating a new User may have already verified a User's email address. A Node may indicate this fact to the Coordinator by populating the relevant attributes provided by the `VerificationAttr-group` attribute group, indicating the `ConfirmationEndpoint` used for verification and the date and time of the verification. The Node SHALL only indicate a verified email address if the Node has verified the email address in a manner equivalent to the Coordinator's email validation process below. See section 14.2.5.

A Node accepting an email address from a User for the purpose of this API SHOULD require the User to enter that email address twice and verify that they match to minimize user error.

As part of UserCreate(), a Node MAY attest to the Coordinator that email verification was performed by a third partyby setting the verificationEntity element to a URL representing the third party. For example, if a Retailer uses a third party email verification, that Retailer would include a URL that references that third party.

The resulting resource, when created, will include the {userid}, and considered a DECE assigned identifier, whose syntax will be:

```
<USERID> ::= "urn:dece:userid:" <useriduniquepart>
```

where <useriduniquepart> is defined as one or more characters that are in the set 'unreserved' as defined in [RFC3986], Section 2.3.

If the user's givenname and/or username if set as DCOORD_TEST_INDICATOR, then the user account may be removed by Coordinator.

Nodes that create test Users but wish to ensure the Coordinator does not delete them may use the DCOORD_TEST_PRESERVE_INDICATOR prefix for the GivenName, Username of EmailAddress of the User. If the User is located within an Account whose DisplayName does not include the DCOORD_TEST_PRESERVE_INDICATOR, all Users within that Account are subject to deletion by the Coordinator.

14.1.4 UserGet(), UserList()

14.1.4.1 API Description

User information may be retrieved either for an individual user or all users in an Account.

14.1.4.2 API Details

Path:

For UserGet, resulting in a single User:

```
[BaseURL]/Account/{AccountID}/User/{UserID}
```

For UserGet, in support of remote Node account creation (with the DataSharingConsent policy):

```
[BaseURL]/Account/{AccountID}/User/{UserID}/DataSharing
```

For UserList, resulting in a list of all users in an Account:

```
[BaseURL]/Account/{AccountID}/User/List[?response={responseType}]
```

Method: GET

Authorized Roles:

```
urn:dece:role:accessportal[:customersupport]
urn:dece:role:dece[:customersupport]
urn:dece:role:retailer[:customersupport]
urn:dece:role:lasp:*[:customersupport]
urn:dece:role:coordinator:customersupport
urn:dece:role:portal[:customersupport]
```

Request Parameters:

For UserGet:

`AccountID` – the unique identifier for an Account
`UserID` – the unique identifier for a User

For UserList:

`AccountID` – the unique identifier for an Account
`response` – optional. By default, that is if no request parameter is provided, the operation returns a list of Users by reference. When present, the `response` parameter can be set to one of the 2 following values:

- **node** – return the Users. Only the `urn:dece:role:dece:customersupport` Role can use this value.
- **reference** – return references to the Users (`UserReference`) – this is the default value.

For example: `[BaseURL]/Account/{AccountID}/User/List?response=reference` will instruct the Coordinator to only return a list of references to Users.

Security Token Subject Scope: `urn:dece:role:user`

Opt-in Policy Requirements:

For UserGet:

NoneFor UserList:

`urn:dece:type:policy:ManageAccountConsent`

Request Body: None

Response Body:

For a single User, response shall be the identified User resource.

For UserList(), the response shall be the UserList collection (UserReference form).

Element	Attribute	Definition	Value	Card.
User		See Table 62	`dece:User-type`	
UserList		See Table 78	`dece:UserList-type`	

14.1.4.3 Behavior

If no error conditions result, the Coordinator returns the User or UserList resource. Only Users whose status is not deleted (that is, not `urn:dece:type:status:archived`, `urn:dece:type:status:other`, `urn:dece:type:status:deleted` or `urn:dece:type:status:forcedeleted`) shall be returned to all invoking Roles, with the exception of the customer support Roles, who have access to all Users in an Account regardless of status.

The Policies applied to the User resource (stored in the `PolicyList` element) SHALL NOT be returned. Nodes may obtain the Parental Controls for the User using the PolicyGet() API.

The `Password` element will be returned only if the `IsRandom` attribute is `true`. When returned, the element will not be populated with the passwords value, and the `IsRandom` attribute will be included with the response set to 'true'.

The DateOfBirth element will not be returned in the response.

14.1.4.3.1 UserGet for Data Sharing

The requirements in this section only apply when UserGet is invoked with the DataSharing form of the endpoint; that is, the form used for remote user account creation.

When UserGet is invoked, `urn:dece:type:policy:DataSharingConsent` must be present and have been created less than DCOORD_DATA_SHARING_CONSENT_DURATION from the time of the UserGet request; otherwise, the Coordinator SHALL reject the request.

The response SHALL only contain the following elements (from the `User` Resource):

- //User/Name
- //User/DisplayImage
- //User/ContactInfo
- //User/Languages

The Coordinator SHALL include the `Cache-control: no-cache, no-store` directives in its response. This will prohibit HTTP caching.

No reference to Coordinator-hosted URLs SHALL be used. If the Node wants to use an image, it would de-reference any URL link included in the response (e.g. //DisplayImage/DisplayImageURL) and copy the data locally.

14.1.5 UserUpdate()

14.1.5.1 API Description

This API provides the ability for a Node to modify some User properties.

14.1.5.2 API Details

Path:

```
[BaseURL]/Account/{AccountID}/User/{UserID}
```

Method: PUT

Authorized Roles:

- `urn:dece:role:accessportal[:customersupport]`
- `urn:dece:role:retailer[:customersupport]`
- `urn:dece:role:lasp:linked[:customersupport]`
- `urn:dece:role:lasp:dynamic[:customersupport]`
- `urn:dece:role:portal[:customersupport]`
- `urn:dece:role:dece[:customersupport]`
- `urn:dece:role:coordinator:customersupport`

Request Parameters:

AccountID is the unique identifier for an Account

UserID is the unique identifier for a User

Security Token Subject Scope:

- `urn:dece:role:user:class:basic` (when managing their own User resource)
- `urn:dece:role:user:class:standard`
- `urn:dece:role:user:class:full`

Opt-in Policy Requirements:

For invoking Roles (except DECE, Web Portal, Coordinator, and all customer support Roles), the `urn:dece:type:policy:EnableManageUserConsent` policy must be TRUE for the Account

resource and urn:dece:type:policy:ManageUserConsent policy must be TRUE for the User resource.

Request Body:

Element	Attribute	Definition	Value	Card.
User			dece:UserData-type	

Response Body: None

14.1.5.3 Behavior

Only Users whose status is urn:dece:type:status:active MAY be updated by non-customer support Roles. Most Roles may only update a subset of a User resource. The following table shows which Roles may change which data elements.

Role	Data Element
urn:dece:role:accessportal[:customersupport] urn:dece:role:retailer urn:dece:role:retailer:customersupport urn:dece:role:lasp:linked urn:dece:role:lasp:linked:customersupport urn:dece:role:lasp:dynamic urn:dece:role:lasp:dynamic:customersupport	ContactInfo DisplayImage Languages Name UserClass
urn:dece:role:coordinator:customersupport urn:dece:role:dece urn:dece:role:dece:customersupport urn:dece:role:portal urn:dece:role:portal:customersupport	Entire User Resource

Table 61: User Data Authorization

A Node accepting an email address from a User for the purpose of this API SHOULD require the User to enter that email address twice and verify that they match to minimize user error.

The Coordinator SHALL provide e-mail notification to the effected User's primary email-address after a successful update has occurred.

14.1.5.4 Password Resets

Customer support Roles SHALL NOT update a user's Credentials/Password directly. Instead, they should invoke a password recovery process with the User at the Web Portal, as defined in section 14.2.7. Customer support Roles MAY update a User's primary e-mail address in order to facilitate e-mail-based password recovery defined in section 14.2.7. The Web Portal, Coordinator, and DECE customer support

Roles MAY update a User password directly. If a User changes a password, the Coordinator will clear any flag that may indicate that the Coordinator generated the password value, as provided for in section 14.1.2.

14.1.5.5 UserRecoveryTokens (Security Questions)

Note: This feature is no longer supported. It is retained here for historical purposes and potential re-introduction in the future.

UserRecoveryToken SHOULD NOT be used. This function is supported for backwards compatibility and may be reinstituted in the future, but its use should be considered deprecated

A UserRecoveryTokens resource maintains questions and their User-supplied answers, which can be used to recover forgotten User Credentials. Processing rules for UserRecoveryTokens are defined in section 14.2.7. These tokens SHALL NOT be used by the Web Portal in order to initiate a question-based password recovery procedure.

UserRecoveryTokens tokens MAY be used to authenticate a User through other communications channels, including voice. Customer support Roles that include voice-based support services SHOULD authenticate a User with these questions if present, in addition to any other knowledge authentication methods the Node may possess.

Customer Support Roles MAY employ UserRecoveryTokens to authenticate a customer who has supplied a username. In this case the Customer Support Role SHALL select one question from the set of user-answered questions and present it to the User through available channels (Web interface, online chat, e-mail, phone conversation, etc.).

The Customer Support Role SHALL then compare the answer to the original User-supplied answer, either programmatically (after removing punctuation and whitespace from both strings) or by human comparison, to determine if the customer is authorized to access the identified User and Account records.

Customer Support Roles SHALL NOT ask for password through any channel.

14.1.6 UserDelete()

14.1.6.1 API Description

This removes a User from an Account. The User's status is changed to *deleted*, rather than removed to provide an audit trail, and to allow restoration of a User that was inadvertently deleted.

14.1.6.2 API Details

Path:

```
[BaseURL]/Account/{AccountID}/User/{UserID}
```

Method: DELETE

Authorized Roles:

```
urn:dece:role:accessportal[:customersupport]
urn:dece:role:portal[:customersupport]
urn:dece:role:retailer[:customersupport]
urn:dece:role:lasp:*[:customersupport]
urn:dece:role:coordinator:customersupport
```

Request Parameters:

AccountID is the unique identifier for an Account

UserID is the unique identifier for a User

Security Token Subject Scope: `urn:dece:role:user:full`

Opt-in Policy Requirements:

For the Access Portal, LASP, and Retailer Roles, successful invocation requires that the Account-level policy `urn:dece:type:policy:EnableManageUserConsent` is TRUE on the Account resource and that the User-level policy `urn:dece:type:policy:ManageUserConsent` is TRUE on the User resource.

Request Body: None

Response Body: None

14.1.6.3 Requester Behavior

The Coordinator SHALL NOT allow the deletion of the last User associated with an Account. If User wants to close an Account entirely, then AccountDelete() SHALL be used.

The Coordinator SHALL NOT allow the deletion of the last full-access User associated with an Account. If the User being deleted is the only Full Access User, and there are additional Users in the Account, a new Full Access User SHALL be created, before the Coordinator will allow the deletion to occur. If the requestor wishes to remove the last remaining User in an Account, then the AccountDelete API SHALL be used instead.

Deletion of the invoking User identified in the presented Delegation Security Token SHALL be allowed.

The Coordinator SHALL invalidate any outstanding Delegation Security Tokens associated with a deleted User. The Coordinator MAY initiate the appropriate specified Delegation Security Token logout profile to any Node which possesses a Delegation Security Token.

User resources whose status is changed to `deleted` SHALL be retained by the Coordinator for at least as many days from the date of deletion as determined by the defined Ecosystem parameter DCOORD_DELETION_RETENTION. Deleted Users SHALL NOT be considered when calculating the number of Users in the Account.

The Coordinator SHALL provide e-mail notification to the effected User's primary email-address after a successful deletion has occurred.

14.1.7 UserValidationTokenCreate()

14.1.7.1 API Description

This API will be used by Nodes to request the DECE Coordinator to issue a new verification token of the token type specified in the request.

To minimize the impact of automated attacks to this API, including each `TokenType` variant, all Nodes, including the Web Portal, SHALL employ a reverse Turing test after the maximum allowable retries has been exceeded. This limit is defined as DCOORD_VALIDATION_TOKEN_RETRY_LIMIT attempts by a User within the DCOORD_VALIDATION_TOKEN_RETRY_TIMEOUT that would result in the invocation of this API. [DSECMECH] section 3.4.3 defines requirements for implementations of a reverse Turing test.

For example, a Node may provide password recovery capabilities within their web application, accessible to anonymous users. The user may attempt providing an e-mail address to the tool 3 times in a span of 15 minutes before being additionally challenged with a CAPTCHA.

Note: The terms validation and verification are used interchangeably in this section.

14.1.7.2 API Details

Path:

When a Security Token is available to the node:

```
[BaseURL]/Account/{AccountID}/User/{UserID}...
.../VerificationToken/{TokenType}
```

When a Security Token is not available to the node, or to request a Security Token to be established:

```
[BaseURL]/VerificationToken/{TokenType}?subject={UserIdentifier}[&responseType={SecurityTokenResponseType}]
```

Method: POST

Authorized Roles:

urn:dece:role:dece[:customersupport]
urn:dece:role:coordinator:customersupport
urn:dece:role:portal[:customersupport]
urn:dece:role:retailer[:customersupport]
urn:dece:role:lasp:dynamic[:customersupport]
urn:dece:role:lasp:linked[:customersupport]
urn:dece:role:accessportal[:customersupport]

Node-based Access Control: The Validated email form of this API employs Node-based acess control measures. Nodes SHALL NOT use that form of the API without permission from DECE. Note: Node-based Access Control can be policy-based or Coordinator-enforced. Other forms of this API do not include Node-based access control.

Request Parameters:

`AccountID` is the unique identifier for an Account
`UserID` is the unique identifier for a User
`TokenType` is the type of confirmation token request. Valid values defined below.
`Useridentifier` is the `PrimaryEmailAddress`. If `TokenType` is `urn:dece:type:token:ResetPassword` or `urn:dece:type:token:DelegationTokenRequest:VerifiedEmail`, this identifier may instead contain one of the `AlternateEmailAddress`. This identifier is used as the primary search criteria
`SecurityTokenResponseType` is the profile identifier of a suitable delegation token profile as defined in [DSecMech].

Security Token Subject Scope: `urn:dece:role:user` if present. See Behavior below for details.

Opt-in Policy Requirements:

None

Request Body: None or a Delegation Security Token Request (for the urn:dece:type:token:DelegationTokenRequest and urn:dece:type:token:DelegationTokenRequest:VerifiedEmail tokentypes)

Response Body: None

14.1.7.3 Behavior

The requestor provides a TokenType value of:

- `urn:dece:type:token:ValidateEmail` – instructs the Coordinator to send a new email address confirmation message to the specified User.
- `urn:dece:type:token:ResetPassword`- instructs the Coordinator to send a forgotten credential message to the specified User.
- `urn:dece:type:token:UnlockMe` - instructs the Coordinator to send an Account unlock message to the specified User. A locked account typically occurs after sequential authentication attempt failures.
- `urn:dece:type:token:DelegationTokenRequest`- instructs the Coordinator to initiate an email-based account linking exchange. See section 14.1.7.4 for details.
- `urn:dece:type:token:DelegationTokenRequest:VerifiedEmail`- instructs the Coordinator to immediately create a Delegation Security Token. See section 14.1.7.4.1 for details.

A Node SHALL include a Delegation Security Token for the associated User if that Node bears such a Delegation Security Token.

This API shall generate a new verification token of the requested token type for a given User. This operation shall invalidate any previously outstanding verification token of the requested token type associated with the User.

The Coordinator SHALL NOT allow Users below the DGEO_CHILDUSER_AGE to use the `urn:dece:type:token:ResetPassword` token type with the API variant not requiring a Delegation Security Token. That is, Child Users cannot do email-based Credential Recovery. Such Users will need to have their passwords reset at the Portal or an authorized Node by the applicable Connected Legal Guardian or the Child User themselves (either at the Portal or the API with the Connected Legal Guardian's Delegation Security Token or the Childs Delegation Security Token). An authorized Node is one for which the policy `urn:dece:type:policy:ManageUserConsent` has been established for the subject User.

If the supplied subject query parameter does not match one or more Users, the Coordinator shall respond with an `HTTP 404 Not Found` response code.

If the supplied subject query matches exactly one User and that User is in the `urn:dece:type:status:blocked` status, the Coordinator will update the User status to the previous status of the User, prior to generating an email communication.

If the supplied subject query matches (in the API variant without the Delegation Security Token) exactly one User and that User is below the DGEO_CHILDUSER_AGE, the Coordinator will not service the request to non-customer support roles, and will respond with an HTTP 403 Forbidden response code.

In the case of the `urn:dece:type:token:ResetPassword` parameter, the Coordinator will require that the User establish a password when the verification token is redeemed at the Coordinator. The update of a User's password shall follow the requirements of [DSecMech] section 6, and 14.1.4, but may match a previously established password.

Successful creation of a new verification token shall result in a new verification email message to be sent to the User, and the Coordinator shall response with an `HTTP 200 OK` response code. This email will include, at a minimum:

- The one-time-use verification token (to allow for cases when the URL above cannot be used, for example, within certain devices).
- The URL where the verification token can be submitted to complete the verification process.

The Coordinator will generate the verification token of a length and validity period such that verification token collisions are impossible. The length and validity period of verification tokens may be a function of actual or anticipated load, however they will not exceed DCOORD_VALIDATION_TOKEN_MAX_LENGTH (but will usually be DCOORD_VALIDATION_TOKEN_TYPICAL_LENGTH bytes). It will consist of the following Unicode code points:

- U+002D (HYPHEN-MINUS)
- U+0030 through U+0039 (0-9)
- U+0042 through U+005A (A-Z), matching is case insensitive

If the supplied subject query parameter matches more than one User at or above the DGEO_CHILDUSER_AGE, the Coordinator will be required to associate the supplied verification token with a set of Users that matched the API request, and SHALL present to the person undergoing a verification token confirmation:

- the Account DisplayName
- the User's GivenName and SurName

for each User that shares the same primary email address. Users below the DGEO_CHILDUSER_AGE shall not be included in this disambiguation step. For example: “John Smith (the Smith’s household)”.

Once the User has been uniquely identified, the Coordinator will redirect the User to a page for the User to perform the necessary action(s) associated with the `TokenType` provided in the original invocation.

Once the User has completed the action(s) associated with the `TokenType`, the Coordinator will redirect the User to their profile page at the Web Portal.

To mitigate the exposure of abuse by unauthenticated users at Node’s and the Portal, use of this API’s Security Token-less form is limited to DCOORD_VALIDATION_TOKEN_RETRY_LIMIT, which is calculated based on the supplied `UserIdentifier` API parameters irrespective of the Node associated with this API invocation.

If the DCOORD_VALIDATION_TOKEN_RETRY_LIMIT has been reached for the supplied `UserIdentifier`, the Coordinator will respond with an `HTTP 403 Forbidden` status code, and an errorID of `urn:dece:errorid:org:dece: ValidationTokenRetryLimitReached`. The Coordinator will reset the counter for each `UserIdentifier`, after DCOORD_VALIDATION_TOKEN_RETRY_TIMEOUT.

To minimize the impact of automated attacks to this API, when receiving this error, the Web Portal and Nodes SHALL employ a reverse Turing test in accordance with [DSECMECH] section 3.4.

14.1.7.4 Email-based Delegation Security Token Establishment

A Node may initiate an email-based process to establish a UserLinkConsent policy as defined in Section 5 and obtain a Delegation Security Token as defined in [DSecMech] by use of this API. It does so by indicating a `{tokentype}` parameter value of `urn:dece:type:token:DelegationTokenRequest` and supplying in the body of the HTTP request a fully formed Delegation Security Token request as defined in [DSecMech]. The specificities of the supplied HTTP request body are defined by the Delegation Security Token profiles implemented at the requesting Node (see section 5 of [DSecMech]). Responses by the Coordinator will use the same Security Token profile that the request was made with. For example, a SAML AuthNRequest submission to this API will result in a SAML Response to the Node.

Errors in the body of the API submission will result in security profile-specific error messages. Other errors will be handled in the same manner as other API invocations (that is, an `ErrorList` in the body of the response).

A validation token generated by the Coordinator for this token type SHALL be valid for no more than DCOORD_VALIDATION_DELEGATIONTOKEN_MAXLIFE, is valid for exactly one use and is unique

compared to other validation tokens within the DCOORD_VALIDATION_DELEGATIONTOKEN_MAXLIFE time span. Once a token of this type has expired, it shall be considered invalid if presented to the Coordinator, and a new token will be required, provided the DCOORD_VALIDATION_TOKEN_RETRY_LIMIT has not been reached.

The validation token generated by the Coordinator acts as an internal reference for correlating a User response to the corresponding request from a Node.

The requesting Node SHALL include a UserLinkConsentPolicy in the request.

If the UserLinkConsent Policy does not already exist for the Node and User, the Coordinator SHALL create a UserLinkCreate Policy for the invoking Node's Organization and the identified User.

If the UserLInkConsent Policy already exists for the Node and User, the Coordinator MAY overwrite the existing UserLinkConsentPolicy for that Node and User with the new UserLinkConsent Policy

Nodes MAY include other Policies as allowed by [DSecMech] section 5.

If the UserLinkConsentPolicy is not present in the request, then the Coordinator SHALL reject the request and return the HTTP status code *403 Forbidden.*

When a valid validation token is submitted to the Coordinator, the Coordinator SHALL create a UserLinkConsent policy for the invoking Node's Organization and the identified User.

The Node signals to the Coordinator which Delegation Security Profile it wants for the response by using that Profile's identification URN in the `SecurityTokenResponseType` request parameter. The Coordinator responds in accordance with the protocol defined in [DSecMech] for the requested Delegation Security Profile type.

14.1.7.4.1 Validated Email Requests

A Node SHALL NOT include an email address of the User in the SAML //Subject/NameID element if the Node has not previously verified the email address.This will result in a new SAML Assertion to be created immediately, without the need for the Coordinator to interact with the User prior to the issuance of a Delegation Security Token.

To use this form of the API, Nodes SHALL have an established authentication session with the User and SHALL have positively verified the email address associated with the User.

The Coordinator shall be capable of unambiguously identifying exactly one User based solely on Primary and secondary email addresses already provisioned. Ambigous matches will result in an error response.

The requesting Node SHOULD include a UserLinkConsentPolicy in the request. If UserLinkConsent policy is requested, Coordinator SHALL create a UserLinkConsent policy for the invoking Node's Organization and the identified User. The presence of the UserLinkConsent policy affects the duration of Delegation Security Tokens (see [DSecMech] 4.3.2.1).

Successful requests will result in an `HTTP 201 Created` response, with the `Location` header indicating the URL of the created SAML Assertion.

14.1.7.4.2 Email message-based Requests

If the request does not include a User email address in the request, the Coordinator sends an email message (the "account link request email") to the primary email address of the User identified by the {UserIdentifier} parameter of the API invocation. The email includes at a minimum a fully qualified URL that incorporates the validation token suitable for an [HTML4] compatible user agent, as well as the URL of the Coordinator validation resource and the validation token in plain text form.

The User may perform an HTTP GET (typically by clicking on an included link in the email message or by typing the validation resource into an HTML user agent) on one of the provided URLs.

Should a Node require a stateful mechanism for such an email-based exchange, it MAY request that session state be transferred to the email verification process, provided the requested Delegation Security Token Profile supports this capability. If provided in the original request and if supported by the Delegation Security Token profile, the Coordinator will include such session state information in its response to the Node.

For example, the SAML Delegation Security Token profile allows for the `RelayState` parameter to be included in a SAML response via the urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect and urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST bindings, defined in [SAML2BIND] and discussed in [DSecMech].

A prototypical sequence of events is depicted in Figure 14 below.

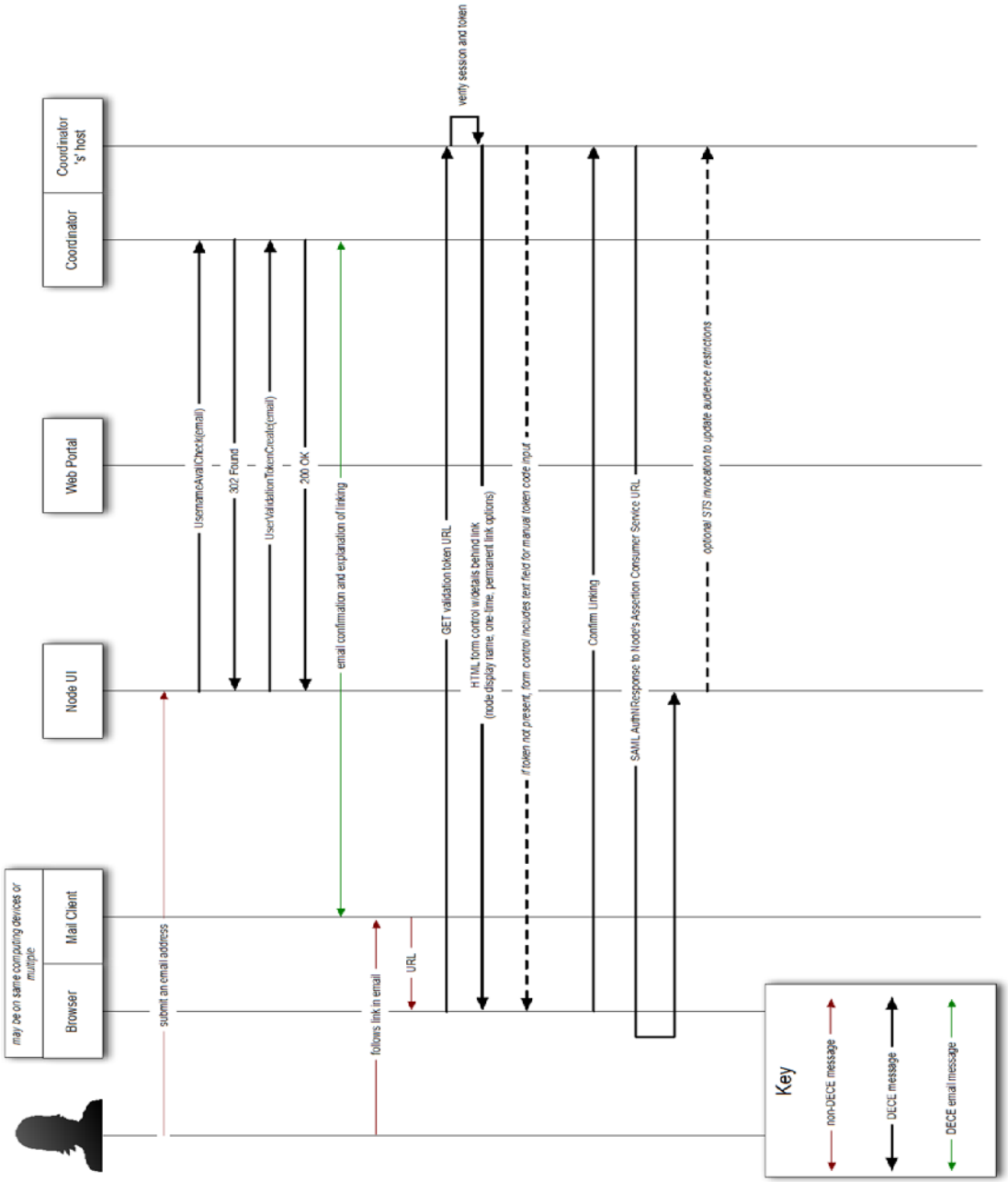


Figure 14 Example Email-based Delegation Token Establishment Flow

14.2 User Types

14.2.1 UserData-type Definition

The User Resource's construction will be heavily influenced by specific geo-political requirements. These requirements will be generally addressed in [DGeo] section 2, and may also be amended by specific Geography Policies outlined in the applicable [DGeo] Appendices. The criteria specified there include age restrictions for Roles, grace periods for the acceptance of Terms of Use (see section 5.5.2.3) and certain restrictions on the modification of properties of a User Resource.

Element	Attribute	Definition	Value	Card.
User				
	UserID	The Coordinator-specified or Node-specified User identifier, which SHALL be unique among the Node and the Coordinator.	`dece:EntityID-type`	0..1
	UserClass	The class of the User. Defaults to the class of the creating User	`dece:UserClass-type` (defined as an `xs:string`)	
Name		GivenName and Surname If GivenName is set as DCOORD_TEST_INDICATOR, then the user account may be removed by Coordinator.Refer Section 13.1.3	`dece:PersonName-type`	
DisplayImage		A chosen display image (or avatar) for the user.	`dece:DisplayImage-type`	0..1
ContactInfo		Contact information which includes the definion of the Users Country, which may be required depending on requirements defined in [DGeo].	See `UserContactInfo-type`	
Languages		Languages used by User	dece:`Languages-type`	0..1
DateOfBirth		The DateOfBirth date value of the User SHALL NOT be supplied by the Nodes.	`dece:DateOfBirth-type`	0..1
LegalGuardian		A reference to the identified Legal Guardian for the User. Usage SHALL be in accordance with [DGeo].	`dece:LegalGuardian-type`	0..n

Element	Attribute	Definition	Value	Card.
Credentials		The Security Tokens used by the User to authenticate to the Coordinator If UserName is set as DCOORD_TEST_INDICATOR, then the user account may be removed by Coordinator.Refer Section 13.1.3	`dece: UserCredentials-type`	
UserRecoveryTokens		A pair of security questions used for password recovery interactions between the Coordinator and the User. Two questions, identified by URIs are selected from a fixed list the Coordinator provides, and the User's `xs:string` answers. Matching is case insensitive; and punctuation and white space are ignored.	`dece: PasswordRecovery-type`	0…1
PolicyList		Collection of policies applied to the User	`dece:PolicyList-type`	0..1
ResourceStatus		Indicates the status of the User resource. See section 17.2.	`dece: ElementStatus-type`	0..1

Table 62: UserData-type Definition

The DateOfBirth-type allows for the expression of either:

- A full date expression (i.e., YYYY-MM-DD) or a date expressed with a granularity of month (i.e., YYYY-MM)
- A NULL value, with the boolean attribute `MeetsAgeOfMajority` indicating if the User meets the applicable geographies criteria (as defined by [DGeo]). For example, <DateOfBirth MeetsAgeOfMajority="true"/>

Element	Attribute	Definition	Value	Card.
DateOfBirth			`Extends dece:DayOptionalDate-type`	
	MeetsAgeOfMajority	As allowed by [DGeo], this flag may be used to indicate the User meets the DGEO_AGE_OF_MAJORITY requirement.	`xs:boolean`	0..1

Table 63: DateOfBirth-type definition

The simple type DayOptionalDate-type extends the date datatype to allow the omition of the day value in a date expression

Element	Attribute	Definition	Value	Card.
DayOptionalDate-type			union: xs:date or xs:gYearMonth	

Table 64: DayOptionalDate-type Definition

The DisplayImage-type allows for either the submission of the raw image data, or a reference URL to the image.

Element	Attribute	Definition	Value	Card.
DisplayImageURL		A fully qualified URL to the User's display image.	dece:AbstractImageResource-type	(choice)
DisplayImageData		A base 64 encoded image to incorporate into the User resource. The Coordinator shall store and assign the supplied image a URL for incorporation into other User resource requests as DisplayImageURL	xs:base64Binary in accordance with [RFC2045]	(choice)

Table 65: DisplayImage-type Definition

14.2.2 UserContactInfo Definition

Element	Attribute	Definition	Value	Card.
UserContactInfo			dece:UserContactInfo-type	
PrimaryE-mail			dece:ConfirmedCommunicationEndpoint-type	
AlternateE-mail			dece:AlternateEmail-type	0..5
Address			dece:ConfirmedPostalAddress-type	0..1
TelephoneNumber			dece:ConfirmedCommunicationEndpoint-type	0..1
MobileTelephoneNumber			dece:ConfirmedCommunicationEndpoint-type	0..1

Table 66: UserContactInfo Definition

14.2.3 ConfirmedPostalAddress-type Definition

Element	Attribute	Definition	Value	Card.
ConfirmedPostalAddress-type			dece: ConfirmedPostalAddress-type	
	VerificationAttr-group	See Table 69	dece: VerificationAttr-group	
PostalAddress		An optional street address.	xs:string	0…n
PostalCode		An optional postal code.	xs:string	0…1
Locality		An optional Locality (e.g. City)	xs:string	0…1
StateOrProvince		An optional state or province name.	xs:string	0…1
Country		Only authorized countries as defined in [DGeo] Section 2.2 SHALL be valid values for this element. The Coordinator validates this value and SHALL return an error if the Country value is not authorized or is invalid. This value SHALL conform to values as specified in [ISO3166-1].	xs:string	1

14.2.4 ConfirmedCommunicationEndpoint Definition

Element	Attribute	Definition	Value	Card.
Confirmed Communication Endpoint			dece:Confirmed Communication Endpoint-type	
	VerificationAttr-group	See Table 69	dece: VerificationAttr-group	
Value			xs:string	
ConfirmationEndpoint			xs:anyURI	0..1
VerificationToken			xs:string	0..1

Table 67: ConfirmedCommunicationEndpoint Definition

14.2.5 AlternateEmail Definition

Element	Attribute	Definition	Value	Card.
AlternateEmail			Extends dece:ConfirmedCommunicationEndpoint-type	
	Notify	Flag for alternate email address to indicate it is to be used for notification purposes (Notify=True).	xs:Boolean (default: false)	

Table 68: AlternateEmail Definition

14.2.6 VerificationAttr-group Definition

Element	Attribute	Definition	Value	Card.
VerificationAttr-group			dece:VerificationAttr-group	
	ID		xs:anyURI	0..1
	verified	Indication if the communication endpoint has been confirmed. A Node may set this value to true, if it has completed the verification of this communication endpoint for this User in accordance with 14.1.2.	xs:boolean	0..1
	VerificationStatus	Indication of the verification status, if the verification is to be performed by the Coordinator. Nodes SHALL set this value to urn:dece:type:status:success if and only if it has indicated positive verification in the verified attribute above. Valid values are described below.	dece:VerificationStatus-type Restricts dece:EntityID-type	0..1

Element	Attribute	Definition	Value	Card.
	VerificationDateTime	The DateTime the communication endpoint was confirmed by the Coordinator or Node.	`xs:dateTime`	0..1
	VerificationEntity	The NodeID of the node that performed the confirmation	`xs:anyURI`	0..1

Table 69: VerificationAttr-group Definition

14.2.6.1 VerificationStatus-type Definition

When the Coordinator is in the process of performing validation of a communication endpoint (for example, the PrimaryEmail), the VerificationStatus attribute will indicate the current state of the process. Possible values (dece:VerificationStatus-type) are:

- `urn:dece:type:status:pending` – the verification processes in underway, but has not been completed yet
- `urn:dece:type:status:success` – the verification processes has been successfully completed
- `urn:dece:type:status:failed` – the verification processes failed. This may mean that the endpoint responded with an undeliverable error response or other delivery-related failure
- `urn:dece:type:status:expired` – the verification process reached its maximum attempt threshold. For example, the DCOORD_E-MAIL_CONFIRM_TOKEN_MAXLIFE limit was reached

Nodes may make use of this information to assist Users in completing the verification process.

14.2.7 PasswordRecovery Definition

Element	Attribute	Definition	Value	Card.
PasswordRecovery			`dece:PasswordRecovery-type`	
RecoveryItem			`dece:PasswordRecoveryItem-type`	1…n

Table 70: PasswordRecovery Definition

14.2.8 PasswordRecoveryItem Definition

Element	Attribute	Definition	Value	Card.
PasswordRecovery Item			dece:PasswordRecoveryItem-type	
QuestionID			xs:positiveInteger	
Question			xs:string	0..1
QuestionResponse			xs:string	

Table 71: PasswordRecoveryItem Definition

14.2.8.1 Visibility of User Attributes

The following table indicates the ability of User Access Levels to read and write the values of a User resource property. An *R* indicates that the User may read the value of the property, and a *W* indicates that the User may write the value.

User Property	Self*	Basic-Access	Standard-Access	Full-Access	Notes
UserClass	R	R	RW[1]	RW	
UserID	R	R	R	R	The UserID is typically not displayed, but may appear in the URL.
Name	RW	R	RW[1]	RW	
DisplayImage	RW	R	RW[1]	RW	
ContactInfo	RW	R	RW[1]	RW	ContactInfo/Address/Country is only writable under conditions described in [DGeo].
Languages	RW	R	RW[1]	RW	
DateOfBirth	RW	R	R	RW	Since standard-access Users may not set parental controls, they should not be able to write to this property.
Policies:Consent	RW	R	R	RW	
Policies:ParentalControl	R	R	R	RW	
Credentials/Username	RW	R	RW[1]	RW	
Credentials/Password	W	N/A	W[1]	W	
UserRecoveryTokens	RW	N/A	RW[1]	RW	
ResourceStatus/Current	R	R	R	RW	The current status of the User can be read (and written to, in the case of the full-access User). Prior status is not available to any User.

Table 72: User Attributes Visibility

*The pseudo-role Self applies to any user's access to properties of his or her own User. The policy evaluation determines access based on the union of the Self column with the user classification column.

[1] The standard-access User has write access to the basic-access and standard-access Users.

In addition to the constraints listed in Table 72, access to User resource properties using a Node other than the Web Portal requires the ManageUserConsent policy to be TRUE for the User (and EnableManageUserConsent to be TRUE for the Account). See Section 5 for additional details.

The customer support Roles may, in addition to always having read access to the UserRecoveryTokens, have write-only access to the Credentials/Password property in order to reset a user's password, provided that the ManageUserConsent policy is TRUE for the User (and EnableManageUserConsent is TRUE for the Account). The `portal:customersupport` and `dece:customersupport` Roles shall always have write access to the Credential/Password and read access to UserRecoveryTokens properties, regardless of the ManageUserConsent policy setting for the User.

14.2.8.2 ResourceStatus-type

A User's status may undergo change, from one status to another (for example, from `urn:dece:type:status:active` to `urn:dece:type:status:deleted`). The Status element (in the ResourceStatus element) may have the following values.

User Status	Description
`urn:dece:type:status:active`	User is active (the normal condition for a User)
`urn:dece:type:status:archived`	The User has been removed from the Coordinator. Only the Coordinator can set a User to this status.
`urn:dece:type:status:blocked`	Indicates that the User experienced multiple login failures, and requires reactivation either through password recovery or update by a full access User in the same Account. While this status is no longer in use, Users created prior to this version of the specification may be in this status.
`urn:dece:type:status:blocked:clg`	Indicates that a User under the DGEO_CHILDUSER_AGE has been suspended as a result of a status change of the User identified in the LegalGuardian element of the User.
`urn:dece:type:status:blocked:tou`	User has been blocked because the User has not accepted the current, in force Terms Of Use (TOU). The User can authenticate to the Web Portal or other Node, but cannot have any actions performed on their behalf via Web Portal or other Node until the DECE terms have been accepted via the Web Portal or other Node and status is returned to `active`.
`urn:dece:type:status:deleted`	User has been deleted from the Account (but not removed from the Coordinator). This status can be set by a full-access User or customer support Role. Only the customer support Roles can view Users in this state.
`urn:dece:type:status:forcedeleted`	An administrative delete was performed on the User.
`urn:dece:type:status:other`	User is in a non-active, but undefined state
`urn:dece:type:status:pending`	Indicates that the User resource has been created, but has not been activated.
`urn:dece:type:status:mergedeleted`	Indicates that the resource should be (in context of merge test) or is (after merge) force deleted as part of a merge process
`urn:dece:type:status:suspended`	User has been suspended for some reason. Only the Coordinator or the customer support Role can set this status value.

Table 73: User Status Enumeration

StatusHistory values SHALL be available using the API for historical resources for no longer than the number of days determined by the defined Ecosystem parameter DCOORD_DELETION_RETENTION.

14.2.9 UserCredentials Definition

User credentials are authentication tokens used when the Coordinator is directly authenticating a User, or when a Node is employing the Login API.

Element	Attribute	Definition	Value	Card.
UserCredentials			dece:UserCredentials-type	
Username		User's user name	xs:string	
Password		Password associated with user name. This element SHALL NOT be included in UserCreate if the intention is to have the Coorddinator generate the password.	dece:Password-type	0..1

Table 74: UserCredentials Definition

14.2.10 Password-type Definition

Element	Attribute	Definition	Value	Card.
dece:Password-type		Password. SHALL be empty if IsRandom is 'true'	`Extends xs:string`	
	IsRandom	Indication if the stored password was randomly assigned by the Coordinator or not. SHALL NOT be included if 'false'. Nodes SHALL NOT include this attribute during User creation.	`xs:boolean`	0..1

14.2.11 UserContactInfo Definition

UserContactInfo describes the methods by which a User may be reached. The uniqueness of e-mail addresses SHALL NOT be required: Users may share primary or alternate e-mail addresses within or

across Accounts. The PrimaryE-mail and AlternateE-mail elements SHALL be limited to DCOORD_EMAIL_ADDRESS_MAXLENGTH.

Element	Attribute	Definition	Value	Card.
UserContactInfo			`dece:UserContactInfo-type`	
PrimaryE-mail		Primary e-mail address for User.	`dece:ConfirmedCommunicationEndpoint-type`	
AlternateE-mail		Alternate e-mail addresses, if any	`dece:Confirmed CommunicationEndpoint-type`	0..n
Address		Mailing address	`dece:Confirmed PostalAddress-type`	0..1
TelephoneNumber		Phone number (uses international format, that is, +1).	`dece:Confirmed CommunicationEndpoint-type`	0..1
Mobile TelephoneNumber		Phone number (uses international format, that is, +1).	`dece:Confirmed CommunicationEndpoint-type`	0..1

Table 75: UserContactInfo Definition

14.2.12ConfirmedCommunicationEndpoint Definition

E-mail addresses SHOULD be confirmed by the Coordinator or other entity. The Coordinator SHALL reflect the status of the confirmation after confirmation is obtained (using appropriate mechanisms).

An e-mail address is considered confirmed if either

- The Coordinator has received a response to a verification email within DCOORD_CONFIRMATION_AGE of current time
- A Node has attested that email verification was performed by a third party by setting the `verificationEntity` attribute to a URL representing the third party. Note that `verificationEntity` is included in the `VerificationAttribute-group`.

Element	Attribute	Definition	Value	Card.
Confirmed Communication Endpoint			`dece:Confirmed CommunicationEndpoint-type`	

Element	Attribute	Definition	Value	Card.
	VerificationAttr-group		dece:VerificationAttr-Group	0..1
Value		The string value of the User attribute.	xs:string	
ConfirmationEndpoint		When confirmation actions occur, this value indicates the URI endpoint used to perform the confirmation (may be a mailto:URI, an https:URI, a tel:URI or other scheme).	xs:anyURI	0..1
VerificationToken		This value is only known only to the Coordinator and cannot be set or retrieved via any API invocation. This element SHOULD NOT be used.	xs:string	0..1

Table 76: ConfirmedCommunicationEndpoint Definition

14.2.13Languages Definition

The Languages element specifies which language or languages the User prefers to use when communicating. The language should be considered preferred if the Primary attribute is TRUE. A primary language should be preferred over any language whose Primary attribute is missing or FALSE. Language preferences SHALL be used by the Coordinator to determine user-interface language, and MAY be used for other user interfaces. At least one language must be specified.

HTTP-specified language preferences as defined in [RFC2616] SHOULD be used when rendering user interfaces to the Coordinator. For API-based interactions, the Coordinator SHOULD use the language preference stored by the User resource when returning system messages such as error messages. (The User is derived from the associated Delegation Security Token presented to the API endpoint.)
Languages extends the xs:language type with the following elements.

Element	Attribute	Definition	Value	Card.
Languages			dece:Languages-type extends xs:language	

Element	Attribute	Definition	Value	Card.
	primary	If TRUE, language is the preferred language for the User.	`xs:boolean`	0..1

Table 77: Languages Definition

14.2.14UserList Definition

This construct provides a list of Users either by reference or value. The list of Users by value is only available to the `urn:dece:role:dece:customersupport` Role.

Element		Attribute	Definition	Value	Card.
UserList-type					
choice	UserReference		The unique identifier of the User	`dece:EntityID-type`	0..n
	User		The User element	`dece:User-type`	0..n

Table 78: UserList Definition

14.3 User Status and APIs Availability

As the User status evolves per the diagrams in section 5.8, certain Coordinator APIs will become available to Nodes (assuming they have a delegation token targeted to that particular User). The table in Appendix H details the availability of each API based on the User status. Note that the table accounts for the differences between Nodes and their Customer Support roles, but does not distinguished between Node Roles (see appendix A for a complete list of API availability per Node Role).

14.4 User Transition from Youth to Adult

When a User transitions through age categories as defined by [DGeo], the Coordinator will automatically adjust the applicable User and Policy resources as described in [DGeo]. The Coordinator SHALL complete these actions within 24 hours of the transition day. If the date of birth of the User contains only year and month, the Coordinator SHALL perform those actions within 24 hours of the first day of that month.

14.5 User Status Transitions

The possible Status values are: `active`, `pending`, `deleted`, `forcedeleted`, `blocked`, `blocked:clg`, `blocked:tou`, `suspended`, `mergedeleted` and `deidentified`.

15 Node Management

A Node is an instantiation of a Role. Nodes are known to the Coordinator and must be authenticated to perform Role functions. Each Node is represented by a corresponding Node resource in the Coordinator. Node resources are only created as an administrative function of the Coordinator and must be consistent with business and legal agreements.

Nodes covered by these APIs are listed in the table below. API definitions make reference to one or more Roles, as defined in the table below, to determine access policies. Each Role identified in this table includes a customersupport specialization, which usually has greater capabilities than the primary Role. Each specialization shall be identified by adding the suffix `:customersupport` to the primary Role. In addition, there is a specific Role identified for DECE customer support.

Role Name	Role URN
Retailer	`urn:dece:role:retailer[:customersupport]`
Linked LASP	`urn:dece:role:lasp:linked[:customersupport]`
Dynamic LASP	`urn:dece:role:lasp:dynamic[:customersupport]`
DSP	`urn:dece:role:dsp[:customersupport]`
DECE Customer Support	`urn:dece:role:dece:customersupport`
Web Portal	`urn:dece:role:portal[:customersupport]`
Content Provider	`urn:dece:role:contentprovider[:customersupport]`
Access Portal	`urn:dece:role:accessportal[:customersupport]`
Coordinator Customer Support	`urn:dece:role:coordinator:customersupport`

Table 79: Roles

15.1 Nodes

Node resources are created through administrative functions of the Coordinator. These resources are thus exclusively internal to the Coordinator.

The Node resources supply the Coordinator with information about the Node implementations. Once a Node is implemented and provisioned with its credentials, it may access the Coordinator in accordance with the access privileges associated with its Role.

15.1.1 Customer Support Considerations

For the purposes of authenticating the customer support Role specializations of parent Roles, the NodeID SHALL be unique. Customer Support Nodes SHALL be authenticated by a unique x509 certificate.

The Coordinator SHALL associate the two distinct Roles. Delegation Security Token profiles specified in [DSecMech] which support multi-party tokens SHOULD identify the customer support specialization as part of the authorized bearers of the Security Token.

For example, using the [SAML] delegation token profile, the AudienceRestriction for a SAML token issued to a retailer should include both the NodeID for the `urn:dece:role:retailer` Role and the NodeID for the `urn:dece:role:retailer:customersupport` Role.

In addition, should a resource have policies which provide the creating Node privileged entitlements, the customer support specialization of that Role SHALL have the same entitlements. This shall be determined by each Nodes association to the same organization. This affiliation is determined by inspecting the OrgID values for each of the Nodes in question.

15.1.2 Basic API Usage by the DECE Customer Care Role

The following is an overview of a customer care applications use of these APIs.

- **Finding a User:** DECE Customer Support performs a query using the ResourcePropertyQuery defined in [DCoord] section 17.3.
- **Obtaining a Delegation Security Token:** DECE Customer Support uses the Delegation Security Token Service defined in [DSecMech] section 8.
- **Obtaining a Resource within an Account** (e.g. User, Right, Policy, etc...): DECE Customer Support performs the UserGet API defined in [DCoord] section 14, using the Delegation Security Token obtained above.

15.1.3 Determining Customer Support Scope of Access to Resources

Most resources of the Coordinator are defined with processing rules on the availability of such resources based on their status. For example, Users that have a status of `urn:dece:type:status:deleted` are not visible to Nodes. This restriction SHALL be relaxed for customer support specializations of the Role (of the same organization, as discussed above). That is, Customer Support Nodes will see resources with status such as `urn:dece:type:status:deleted` and `urn:dece:type:status:mergedeleted`.

15.2 Node and Organization Functions

15.2.1 NodeGet()

NodeGet() retrieves descriptive information about a Node.

15.2.1.1 API Description

This is the means to obtain Node information from the Coordinator.

15.2.1.2 API Details

Path:

```
[BaseURL]/Node/{NodeID}
```

Method: GET

Authorized role:

```
urn:dece:role:accessportal[:customersupport]
urn:dece:role:coordinator:customersupport
urn:dece:role:dece[:customersupport]
urn:dece:role:retailer[:customersupport]
urn:dece:role:lasp:dynamic[:customersupport]
urn:dece:role:lasp:linked[:customersupport]
urn:dece:role:portal[:customersupport]
```

Request Parameters:

`NodeID` – the unique identifier for a Node

Request Body: None

Response Body: Node

15.2.1.3 Behavior

The identified Node is returned.

If the requestor is the requested Node or it is a member of the Organization of the requested Node, the complete Node resource is returned. Otherwise the Coordinator SHALL omit any of the following XML elements from its response:

- //Node/KeyDescriptor
- //Node/DECEProtocolVersion
- //Node/OrgAddress
- //Node/Contacts
- //Node/MediaDownloadLocationBase

15.2.2 NodeList()

NodeList returns a set of Nodes.

15.2.2.1 API Description

This is the means to obtain Node(s) information from the Coordinator.

15.2.2.2 API Details

Path:

```
[BaseURL]/Node/List[?response={responseType}]
```

Method: GET

Authorized role:

```
urn:dece:role:accessportal[:customersupport]
urn:dece:role:coordinator:customersupport
urn:dece:role:dece[:customersupport]
urn:dece:role:retailer[:customersupport]
urn:dece:role:lasp:dynamic[:customersupport]
urn:dece:role:lasp:linked[:customersupport]
urn:dece:role:portal[:customersupport]
```

Request Parameters: None

`response` – optional. By default, that is if no request parameter is provided, the operation returns a list of Nodes. When present, the `response` parameter can be set to one of the 2 following values:

- **node** – return the actual Nodes (default setting)
- **reference** – return references to the Nodes (`NodeReference`)

For example, `[BaseURL]/Node/List?response=node` will instruct the Coordinator to return a list of Nodes.

Request Body: None

Response Body: `NodeList`

15.2.2.3 Behavior

A collection containing all of the Nodes in the system is returned.

If the requestor is a member of the same Organization or if its one of urn:dece:role:dece:[customersupport] or urn:dece:role:coordinator:[customersupport] roles, the complete NodeList is returned. Otherwise the Coordinator SHALL omit any of the following XML elements from its response:

- //Node/KeyDescriptor
- //Node/DECEProtocolVersion
- //Node/OrgAddress
- //Node/Contacts
- //Node/MediaDownloadLocationBase

15.2.3 NodeCreate(), NodeUpdate()

Nodes are managed by the Coordinator in order to ensure licensing, conformance, and compliance certifications have occurred.

15.2.3.1 API Details

Path:

```
[BaseURL]/Node

[BaseURL]/Node/{EntityID}
```

Method: POST | PUT

Authorized role: `urn:dece:role:coordinator:customersupport`

Request Parameters:

Request Body:

Element	Attribute	Definition	Value	Card.
Node			`dece:NodeInfo-type`	

Response Body: None

15.2.3.2 Behavior

With a POST, Node resource is created. Nodes become `active` when the Coordinator has approved the Node for activation.

With a PUT, an existing Node resource identified by the EntityID in the resource request is replaced by the new information. The Coordinator keeps a complete audit of behavior.

15.2.4 NodeDelete()

Node resources cannot simply be deleted, as in many cases, User experience may be affected and portions of the ecosystem may not operate correctly.

15.2.4.1 API Description

The Node's status is set to *deleted*.

15.2.4.2 API Details

Path:

```
[BaseURL]/Node/{EntityID}
```

Method: DELETE

Authorized role: `urn:dece:role:coordinator:customersupport`

Request Parameters: `EntityID` is the unique identifier for a Node

Request Body: None

Response Body: None

15.2.4.3 Behavior

The Node status is set to "deleted". API access to the Node is terminated.

15.2.5 OrganizationGet()

OrganizationGet() retrieves descriptive information about an Organization.

15.2.5.1 API Description

This is the means to obtain Organization information from the Coordinator.

15.2.5.2 API Details

Path:

```
BaseURL]/Org/{OrganizationID}
```

Method: GET

Authorized role:

```
urn:dece:role:accessportal[:customersupport]
urn:dece:role:coordinator:[customersupport]
urn:dece:role:dece[:customersupport]
urn:dece:role:retailer[:customersupport]
urn:dece:role:lasp:dynamic[:customersupport]
urn:dece:role:lasp:linked[:customersupport]
urn:dece:role:portal[:customersupport]
```

Request Parameters:

`OrganizationID` – the unique identifier for an Organization

Request Body: None

Response Body: Organization

15.2.5.3 Behavior

If the requestor is a member of the requested Organization or if its one of urn:dece:role:dece:[customersupport] or urn:dece:role:coordinator:[customersupport] roles, the complete Organization resource is returned. Otherwise the Coordinator SHALL omit the following XML elements from the response:

- //Organization/OrgAddress
- //Organization/Contacts
- //Organization/MediaDownloadLocationBase

15.3 Node Types

15.3.1 NodeList Definition

The NodeList element is a list of Nodes either by value or reference.

Element		Attribute	Definition	Value	Card.
NodeList				dece:NodeList-type	
choice	NodeReference			dece:EntityID-type	0..n
	Node			dece:NodeInfo-type	0..n
		ViewFilterAttr	Response filtering information, see section 17.5	dece:ViewFilterAttr-type	

Table 80: NodeList Definition

15.3.2 NodeInfo Definition

The NodeInfo element contains a Node's information. The `NodeInfo-type` extends the `OrgInfo-type` with the following elements.

Element	Attribute	Definition	Value	Card.
NodeInfo			dece:NodeInfo-type extends dece:OrgInfo-type	
	NodeID	Unique identifier of the Node	dece:EntityID-type	0..1
Role		Role of the Node (a URN of the form urn:dece:role:<Role name>	xs:anyURI	
DeviceManagement URL		Indicates the URL for a user interface which provides legacy device management functionality. This value must only be present for the retailer Role.	xs:anyURI	0..1
DECEProtocol Version		The DECE Protocol version or versions supported by this Node. Valid values are specified in **22**	xs:anyURI	0…n
KeyDescriptor		See Section **17.6**	dece:KeyDescriptor-type	0…n
ResourceStatus		Status of the resource. See section **17.2**	dece:ElementStatus-type	0..1

Table 81: NodeInfo Definition

These types are in the NodeAccess element in the Account-type data element, which is defined in **Table 58**.

15.3.3 OrgInfo-type Definition

Element	Attribute	Definition	Value	Card.
OrgInfo			dece:OrgInfo-type	
	organizationID	Unique identifier for organization defined by DECE.	md:EntityID-type	
DisplayName		Localized User-friendly display name for the organization.	dece:localized StringAbstractType	1.n
SortName		Name suitable for performing alphanumeric sorts	dece:localized StringAbstractType	0..n
OrgAddress		Primary addresses for contact	dece:Confirmed PostalAddress-type	0..1
Contacts			dece:ContactGroup-type	0..1
Website		Link to organization's top-level page.	dece:LocalizedURI Abstract-type	
MediaDownload LocationBase		Location for media download, if organization holds a Retailer Role	xs:anyURI	0..1
LogoResource		Reference to logo image. height and width attributes convey image dimensions suitable for various display requirements	dece:AbstractImage Resource-type	0..n

Table 82: OrgInfo Definition

15.4 Node and Org Images

Node and Org images are intended for display by the Web Portal and by Account Management interfaces at other Nodes. For example, the Web Portal uses these images in the Locker view to identify original Retailers.

During the onboarding process, Node and Org images SHALL be provisioned by the Coordinator for Retailer, LASP, and Access Portal Roles. The Coordinator MAY provision Node and Org images for other Roles.

The following refers to images provided by Nodes as referenced by LogoResource. Note that these are Node requirements, not Coordinator requirements.

- Images SHALL be compliant with [DMeta], Section 3.2. Note that image formats in Section 3.2.2 do not apply.
- Images SHOULD be designed to display against a dark background
- Images SHOULD provide transparency (PNG with Alpha channel) that is suitable for display against a black or dark background.
- Images SHALL be provided in the following sizes (in pixels):
    - For the User LinkedServices and AccountSettings pages: 120 x 80
    - For Media List and Media Details pages: 60 x 40

The following Coordinator processing rules and requirements are applied:

- The images will be fetched from the provided URL and hosted at the Coordinator
- The images will be scanned for viruses, and quarantined as necessary
- The image assets will be published at Coordinator-controlled URLs

The following applies to Nodes displaying images referenced by LogoResource.

- Nodes SHOULD display images over a black or dark background. Note that images are designed to display against a dark background and could have transparent pixels (i.e., alpha channel) that will display background pixels. Node UI designers need to provide a suitable background, at least directly underneath images.

15.5 Node Status Transitions

The possible Status values are: `active`, `deleted`, `pending` and `suspended`.

16 Discrete Media

Discrete Media is the ability for a User to receive a version of the Content on physical media in an approved format, such as a CSS-protected DVD or a CPRM-protected SD Card. DECE Content may be sold by a Retailer with or without a Discrete Media Right.

Fulfilling Discrete Media is the process of creating or otherwise providing to a User a physical instantiation of a right located in an Account's Rights Locker. The specification is designed with some generality to support additional media formats as they become available and approved for use. [DDiscreteMedia] provides an overview of the actual Fulfillment processes.

The Coordinator maintains a record of the availability of fulfillment as one or more Discrete Media Tokens. Each Discrete Media Token serves as a record of the Discrete Media Right, which identifies availableand completed fulfillment of the right.

The processe commences when a Retailer creates a Discrete Media Right at the Coordinator (typically, immediately following the creation of the associated Rights Token). When a Retailer or DSP chooses to fulfill a Discrete Media Right referenced in a Rights Token, the process begins with directly consuming the Discrete Media Right.

A User is said to possess a suitable Discrete Media Right should one be indicated in the Rights Token. This right must be present in the Rights Token in order to obtain a physical media copy of a right recorded in the locker. These entitlements are identified in the Rights Token as DiscreteMediaRightsRemaining. It conveys the list of Discrete Media copies that may be made by the Account. The Coordinator provides a set of APIs, specified here, which enable authorized Roles to create, update, or fulfill the DiscreteMediaRights present in the Rights Token.

Due to a change of policy, prior versions of these APIs restricted the number of Discrete Media Tokens to one per Rights Token. As a result, Nodes that support only 1.x versions of this specification will be limited to fulfilling one Discrete Media Token.

16.1 Discrete Media Functions

Nodes that fulfill Discrete Media SHALL implement the APIs of this section.

The Discrete Media APIs SHALL adhere to the access policies of the Rights Token with which the Discrete Media resource is associated with respect to User policies, including parental controls.

The Coordinator enforces the maximum number of Discrete Media Rights associated with a given Rights Token as defined by DISCRETE_MEDIA_LIMIT in [Dsystem].

In order to supply a Discrete Media Right, a Retailer will be required to create a Discrete Media Right, and the Coordinator will update the DiscreteMediaRightsRemaining in the Rights Token accordingly.

Any Retailer or DSP may fulfill a Discrete Media Right identified as available in a Rights Token. The following APIs provide mechanisms for the fulfillment process of Discrete Media:

- DiscreteMediaRightConsume
- DiscreteMediaRightFulfill

In addition to the ResourceStatus, Discrete Media Rights have a 'state', which indicates the consumption disposition of the right. These states include: `available`, `fulfilled` and `leased`.

16.1.1 DiscreteMediaRightCreate()

16.1.1.1 API Description

When a Retailer offers a Discrete Media Right with a Rights Token, or at any time chooses to add Discrete Media capabilities to an existing Rights Token, the Retailer uses this API to register that right with the Coordinator, subject to the DISCRETE_MEDIA_LIMIT. Any Retailer may ammend an existing Rights Token with a Discrete media Right, provided the Retailer has access to the Rights Token via the RightsTokenGet API after all policy evaluations are applied (including consent and parental control policies).

See also DiscreteMediaRightFulfill (section 16.1.11) which creates and fulfills a Right in a single API call.

16.1.1.2 API Details

Path:

```
[BaseURL]/Account/{AccountID}/RightsToken/{RightsTokenID}/DiscreteMediaRight
```

Method: POST

Authorized Roles:

`urn:dece:role:retailer[:customersupport]`

Request Parameters:

`AccountID` – The Account into which to register the Discrete Media Right

`RightsTokenID` – The Rights Token to which the Discrete Media Right applies

Security Token Subject Scope: `urn:dece:role:user`

Opt-in Policy Requirements: `urn:dece:type:policy:LockerViewAllConsent if Retailer is not the issuing Retailer.`

Request Body: `DiscreteMediaToken`

Element	Attribute	Definition	Value	Card.
DiscreteMediaToken		See Table 83	`dece:DiscreteMediaToken-type`	

Response Body: None.

16.1.1.3 Request Behavior

The Retailer creates a Discrete Media Token which SHALL only include:

- The `MediaProfile` element, indicating which Media Profile can be used for fulfillment.
- The `AuthorizedFulfillmentMethods`, which indicates which DiscreteMediaFulfillment methods can be used for the indicated Rights Token and Media Profile.
- The `RightsTokenID` element.

The Coordinator then:

- Assigns the DiscreteMediaTokenID,
- Sets the State to `available`,
- Sets the RightsTokenID form the value supplied in the invocation URI,
- Increments the DiscreteMediaRightsRemaining and populates FulfillmentMethod of the associated Rights Token

When a DiscreteMedia Right is created, the Coordinator does not enforce any constraints expressed in the `AssetRestriction` element of the corresponding Logical Asset. Enforcement, if any, is performed by Nodes.

16.1.1.4 Response Behaviour

Successful creation will respond with the Location of the newly created resource, or an error (see section 21.1) .

16.1.2 DiscreteMediaRightUpdate()

16.1.2.1 API Description

This API allows a Retailer to update a previously created Discrete Media Right. Only the Node or any other Retailer Affiliated Node that created the Discrete Media Right can update it. The full Discrete Media Token shall be submitted, however, only the MediaProfile and AuthorizedFulfillmentMethod values may be updated.

16.1.2.2 API Details

Path:

```
[BaseURL]/Account/{AccountID}/RightsToken/{RightsTokenID}/DiscreteMediaRight/{DiscreteMediaRightID}
```

Method: PUT

Authorized Roles:

`urn:dece:role:retailer[:customersupport]`

Request Parameters:

`AccountID` is the unique identifier for an Account
`RightsTokenID` is the unique identifier for a right
`DiscreteMediaRightID` is the unique identifier for a Discrete Media Right

Security Token Subject Scope: `urn:dece:role:user`

Opt-in Policy Requirements: none

Request Body: DiscreteMediaToken

Element	Attribute	Definition	Value	Card.
DiscreteMediaToken		See Table 83	`dece:DiscreteMediaToken-type`	

Response Body: none

16.1.2.3 Request Behavior

The Retailer updates a Discrete Media Token which must only alter:

- The `MediaProfile` element
- The `AuthorizedFulfillmentMethods`

The Coordinator validates the updated Discrete Media Right in an identical fashion to those defined above to DiscreteMediaRightCreate().

16.1.2.4 Response Behaviour

If successful, a 200 OK response is given, otherwise, for 400-class errors, the errors are provided in the body.

16.1.3 DiscreteMediaRightDelete()

16.1.3.1 API Description

This API allows the Retailer or Affiliated Node who created the Discrete media Right can delete the Discrete Media Right. Only a Discrete Media Right in the `available` state may be deleted.

16.1.3.2 API Details

Path:

```
[BaseURL]/Account/{AccountID}/RightsToken/{RightsTokenID}/DiscreteMediaRight/{DiscreteMediaRightID}
```

Method: DELETE

Authorized Roles:

`urn:dece:role:retailer[:customersupport]`

Request Parameters:

`AccountID` is the unique identifier for an Account
`RightsTokenID` is the unique identifier for a right
`DiscreteMediaRightID` is the unique identifier for a Discrete Media Right

Security Token Subject Scope: `urn:dece:role:user`

Opt-in Policy Requirements: none

Request Body: none

Response Body: none

16.1.3.3 Request Behavior

The Retailer may delete a Discrete Media Right if its state is `available`, and the requesting Node is an Affiliated Node.

The Coordinator shall follow the deletion by adjusting the associated Rights Token's DiscreteMediaRightsRemaining value appropriately, and may be required to adjust the Rights Token's FulfillmentMethod.

16.1.3.4 Response Behaviour

If successful, a 200 OK response is given, otherwise, for 400-class errors, the errors are provided in the body.

16.1.4 DiscreteMediaRightGet()

16.1.4.1 API Description

Allows an API Client to obtain the details of a Discrete Media Token.

16.1.4.2 API Details

Path:

```
[BaseURL]/Account/{AccountID}/RightsToken/{RightsTokenID}/DiscreteMediaRight/{DiscreteMediaRightID}
```

Method: GET

Authorized Roles:

```
urn:dece:role:accessportal[:customersupport]
urn:dece:role:coordinator:customersupport
urn:dece:role:dece[:customersupport]
urn:dece:role:dsp[:customersupport]
urn:dece:role:lasp[:customersupport]
urn:dece:role:portal[:customersupport]
urn:dece:role:retailer[:customersupport]
```

Request Parameters:

`AccountID` is the unique identifier for an Account

`DiscreteMediaRightID` is the unique identifier for a Discrete Media Token

`RightsTokenID` is the unique identifier for a rights token

Security Token Subject Scope: `urn:dece:role:user`

Opt-in Policy Requirements: Access is restricted to only those API Client that can view the associated Rights Token.

Request Body: None

Response Body:

Element	Attribute	Definition	Value	Card.
DiscreteMediaToken		Describes the Discrete Media Right for a Rights Token	`DiscreteMediaToken-type`	

16.1.4.3 Behavior

Since basic Discrete Media Rights are visible within the Rights Token, only those roles associated with fulfillment can utilize this API, which simplifies policy controls on Account Resources.

16.1.5 DiscreteMediaRightList()

16.1.5.1 API Description

Allows a API Client to obtain a list of DiscreteMediaTokens issued against a particular rights token.

16.1.5.2 API Details

Path:

```
[BaseURL]/Account/{AccountID}/RightsToken/{RightsTokenID}/DiscreteMediaRight/List
```

Method: GET

Authorized Roles:

```
urn:dece:role:accessportal[:customersupport]
urn:dece:role:coordinator:customersupport
urn:dece:role:dece[:customersupport]
urn:dece:role:dsp[:customersupport]
urn:dece:role:lasp[:customersupport]
urn:dece:role:portal[:customersupport]
urn:dece:role:retailer[:customersupport]
```

Request Parameters:

`AccountID` is the unique identifier for an Account

`RightsTokenID` is the unique identifier for a Rights Token

Security Token Subject Scope: `urn:dece:role:user`

Opt-in Policy Requirements: Access is restricted to only those API Client that can view the associated Rights Token.

Request Body: None

Response Body:

Element	Attribute	Definition	Value	Card.
`DiscreteMediaTokenList`		A collection of DiscreteMediaToken resources	`DiscreteMediaTokenList-type`	

16.1.5.3 Behavior

Resource visibility must follow the same policies as a single Discrete Media resource request, thus DiscreteMediaTokens which cannot be accessed SHALL NOT be included in the list.

Only tokens for which the state is:

`urn:dece:type:state:discretemediaright:available`,
`urn:dece:type:state:discretemediaright:leased`, or
`urn:dece:type:state:discretemediaright:fulfilled`

shall be returned. All tokens meeting the state requirements above shall be returned.

For Customer Support-originated requests, tokens of all states shall be returned.

The sort order of the response is arbitrary.

16.1.6 DiscreteMediaRightLeaseCreate()

Note: This feature is no longer supported. It is retained here for historical purposes.

This API is used to reserve a Discrete Media Right. It is used by a DSP or a Retailer to reserve the Discrete Media Right. Once a lease has been created, the Coordinator considers the associated Discrete Media right fulfilled, until either the expiration date and time of the DiscreteMediaToken resource has been

reached, or the Node indicates to the Coordinator to either remove the lease explicitly (in the case of failure), or when a Discrete Media lease is converted to a fulfilled Discrete Media resource.

If a DiscreteMediaToken lease expires, its State attribute shall revert to `available` by the Coordinator.

16.1.6.1 API Details

Path:

```
[BaseURL]/Account/{AccountID}/RightsToken/{RightsTokenID}/{MediaProfile}/
DiscreteMediaRight/{DiscreteMediaTokenID}/{DiscreteMediaFulfillmentMethod}/Lease
```

Method: POST

Authorized Roles:

```
urn:dece:role:dsp
urn:dece:role:retailer
```

Any Retailer or DSP may request a lease, provided they have access to the associated Rights Token.

Request Parameters:

`AccountID` is the unique identifier for an Account
`RightsTokenID` is the unique identifier for a rights token
`MediaProfile` is the identifier of the PurchaseProfile's MediaProfile being fulfilled
`DiscreteMediaTokenID` is the unique identifier for a discrete media rights token
`DiscreteMediaFulfillmentMethod` is the DiscreteMediaFulfillmentMethod identifier for which fulfillment has commenced.

Security Token Subject Scope: `urn:dece:role:user`

Opt-in Policy Requirements: `urn:dece:type:policy:LockerViewAllConsent`

Request Body: Null

Response Body: DiscreteMediaRight Resource

16.1.6.2 Requester Behavior

To obtain a lease on a Discrete Media right (thus reserving a Discrete Media right from being fulfilled by another entity), the Node POSTs a request to the resource (with no body). The requestor SHALL NOT use DiscreteMediaLeaseCreate() unless it is in the process of preparing to Fulfill Discrete Media.

A lease SHALL be followed within the expiration time specified in the DiscreteMediaToken with DiscreteMediaRightLeaseRelease, DiscreteMediaRightLeaseConsume or DiscreteMediaRightLeaseRenew.

If a requestor needs to extend the time, DiscreteMediaRightLeaseRenew() SHOULD be invoked, but only before the lease expiration date and time is reached.

16.1.6.3 Responder Behavior

If no error conditions occur, the Coordinator SHALL respond with an HTTP 200 status code and a DiscreteMediaRight body.

The Coordinator SHALL monitor the frequency leases are allowed to expire by a Node without releasing, renewing, or fulfilling them. Nodes which reach the expiration limit determined by the defined Ecosystem parameter DCOORD_DISCRETEMEDIA_LEASE_EXPIRE_LIMIT may be prevented from creating new leases until the use of the APIs is corrected.

Leases SHALL NOT exceed the duration determined by the defined Ecosystem parameter DCOORD_DISCRETEMEDIA_LEASE_DURATION.

Lease renewals SHALL NOT exceed the amount of time determined by the defined Ecosystem parameter DCOORD_DISCRETEMEDIA_LEASE_MAXTIME.

The Coordinator shall record the requested DiscreteMediaFulfillmentMethod in the Discrete Media Right's FulfillmentMethod element.

The Coordinator shall record the requested MediaProfile in the Discrete Media Right's MediaProfile element.

The Coordinator shall record the UserID in the Discrete Media Right's UserID element from the corresponding value in the provided Security Token.

16.1.7 DiscreteMediaRightLeaseConsume()

Note: This feature is no longer supported. It is retained here for historical purposes.

16.1.7.1 API Description

When a Discrete Media Lease results in the successful fulfillment of physical media, the Node that holds the lease converts the Discrete Media State from `leased` to `fulfilled`.

16.1.7.2 API Details

Path:

```
[BaseURL]/Account/{AccountID}/RightsToken/DiscreteMediaRight/{DiscreteMediaRightID}/Lease/Consume
```

Method: POST

Authorized Roles:

```
urn:dece:role:dsp[:customersupport]
urn:dece:role:retailer[:customersupport]
urn:dece:role:dece:customersupport
```

Request Parameters:

`AccountID` is the unique identifier for an Account

`DiscreteMediaRightID` is the unique identifier for a Discrete Media Right

Security Token Subject Scope: `urn:dece:role:user`

Opt-in Policy Requirements: Access is restricted to only those Nodes that can view the associated Rights Token.

Request Body: None

Response Body:

The Discrete Media Right resource `dece:DiscreteMediaToken-type` is returned in the response, incorporating the updated `State` attribute to `fulfilled`.

Element	Attribute	Definition	Value	Card.
DiscreteMediaToken		The DiscreteMediaToken resource (after updating the type from leased to fulfilled)	`DiscreteMediaToken-type`	1

16.1.7.3 Behavior

The Node that holds the Discrete Media lease (identified by the Discrete Media identifier), SHALL consume a Discrete Media lease. Nodes that do not properly manage their leases may be administratively blocked from performing Discrete Media resource operations until the error is corrected.

Only the Node who is holding the lease, the retailer who issued the Rights Token, its affiliated DSP role, and any of their associated customer support specializations may consume a lease.

Upon successful consumption of the lease, the Coordinator shall update the Discrete Media Right's state to `fulfilled`, and update the Discrete Media Right with the UserID identified in the provided Security Token and the RightsTokenID of the corresponding Rights Token. The Discrete Media Right's LeaseExpiration date time element will be removed.

16.1.8 DiscreteMediaRightLeaseRelease()

Note: This feature is no longer supported. It is retained here for historical purposes.

16.1.8.1 API Description

Nodes that obtained a lease from the Coordinator may release the lease if the Discrete Media operation has failed.

16.1.8.2 API Details

Path:

```
[BaseURL]/Account/{AccountID}/RightsToken/DiscreteMediaRight/
{DiscreteMediaRightID}/Lease/Release
```

Method: POST

Authorized Roles:

`urn:dece:role:dece:customersupport`
`urn:dece:role:coordinator:customersupport`
`urn:dece:role:dsp[:dsp:customersupport]`
`urn:dece:role:retailer[:customersupport]`

Request Parameters:

`AccountID` is the unique identifier for an Account

`DiscreteMediaRightID` is the unique identifier for a Discrete Media Right

Security Token Subject Scope: `urn:dece:role:user`

Opt-in Policy Requirements: None

Request Body: None

Response Body: DiscreteMediaRight Resource

16.1.8.3 Behavior

Only the Node that holds the lease (and its associated customer support specialization) may release the lease.

The Coordinator shall remove the Discrete Media Right's FulfillmentMethod and MediaProfile element values, and update the state to `available`.

16.1.9 DiscreteMediaRightConsume()

16.1.9.1 API Description

Some circumstances may allow a Discrete Media right to be immediately converted from a Discrete Media Right, to a fulfilled Discrete Media Right Resource (with a state of `urn:dece:type:state:discretemediaright:fulfilled`).

With the recent change of policy to relax the the number of Discrete Media Tokens per Rights Token, Coordinator will allow multiple DiscreteMediaTokens to be created under a Rights token. Therefore, the URL path for this API has been modified to take the DiscreteMediaRightID (and remove MediaProfile and RightsTokenID) to uniquely identify the Discrete Media Right to be fulfilled.

16.1.9.2 API Details

Path:

```
[BaseURL]/Account/{AccountID}/RightsToken/
DiscreteMediaRight/{DiscreteMediaRightID}/{DiscreteMediaFulfillmentMethod}/Consume
```

Method: POST

Authorized Role:

`urn:dece:role:retailer[:customersupport]`

Only the Retailer who created the Rights Token and its customer support specialization may invoke this API.

Request Parameters:

`AccountID` is the unique identifier for an Account
`DiscreteMediaRightID` is the unique identifier for a DiscreteMediaRight
`DiscreteMediaFulfillmentMethod` is the identifier for a defined Discrete Media Profile

Security Token Subject Scope: `urn:dece:role:user`

Opt-in Policy Requirements: `urn:dece:type:policy:LockerViewAllConsent`

Request Body:

Response Body: DiscreteMediaRight Resource

16.1.9.3 Behavior

Upon successful consumption of the Discrete Media Right, the Coordinator shall update the Discrete Media Right's state to `fulfilled`, and update the Discrete Media Right with the UserID identified in the provided Security Token and the RightsTokenID of the corresponding Rights Token. The Discrete Media Right's FulfillmentMethod element will be populated with the DiscreteMediaFulfillmentMethod provided in the request. Its MediaProfile element will be populated with the MediaProfile provided in the request (from the corresponding Rights Token).

16.1.10DiscreteMediaRightLeaseRenew()

Note: This feature is no longer supported. It is retained here for historical purposes.

This operation can be used when there is a need to extend the lease of a Discrete Media Right.

16.1.10.1 API Description

The DSP (or retailer) uses this message to inform the Coordinator that the expiration of a Discrete Media Right lease needs to be extended.

16.1.10.2 API Details

Path:

```
[BaseURL]/Account/{AccountID}/RightsToken/DiscreteMediaRight/
{DiscreteMediaRightID}/Lease/Renew
```

Method: PUT

Authorized Roles:

```
urn:dece:role:retailer[:customersupport}
urn:dece:role:dsp[:customersupport]
```

Request Parameters:

`AccountID` is the unique identifier for an Account
`DiscreteMediaRightID` is the unique identifier for a Discrete Media Right

Request Body: None

Response Body:

The Discrete Media Right resource `dece:DiscreteMediaToken-type` is returned in the response, incorporating the updated ExpirationDateTime.

Element	Attribute	Definition	Value	Card.
DiscreteMediaToken			`dece:DiscreteMediaToken-type`	

16.1.10.3 Behavior

Only the Node that holds the lease (and its associated customer support specialization) may renew the lease.

The Coordinator may add a period of time up to the length of time determined by the defined Ecosystem parameter DCOORD_DISCRETEMEDIA_LEASE_DURATION to the identified Discrete Media Right lease. Leases may only be renewed up to the maximum length of time determined by the defined Ecosystem parameter DCOORD_DISCRETEMEDIA_LEASE_MAXTIME.

A new lease must be requested once a lease has exceeded the maximum time allowed.

The Coordinator SHALL NOT issue a lease renewal that exceeds the expiration time of the Security Token provided to this API. In this case the Coordinator SHALL set the lease expiration to match the Security Token expiration.

16.1.11DiscreteMediaRightFulfill()

The DiscreteMediaRightFulfill API is very similar to the DiscreteMediaRightCreate API defined in section 16.1.1, however the outcome of the API invocation is a fulfilled Right (as if a Node called both DiscreteMediaRightCreate and DiscreteMediaRightConsume), providing a more efficient API for Retailers.

16.1.11.1 API Details

Path:

```
[BaseURL]/Account/{AccountID}/RightsToken/{RightsTokenID}/DiscreteMediaRight/Fulfill
```

Method: POST

Authorized Roles:

`urn:dece:role:retailer[:customersupport]`

Request Parameters:

`AccountID` – The Account into which to register the Discrete Media Right

`RightsTokenID` – The Rights Token to which the Discrete Media Right applies

Security Token Subject Scope: `urn:dece:role:user`

Opt-in Policy Requirements: `urn:dece:type:policy:LockerViewAllConsent` if Retailer is not the issuing Retailer.

Request Body: `DiscreteMediaToken`

Element	Attribute	Definition	Value	Card.
DiscreteMediaToken		See Table 83	`dece:DiscreteMediaToken-type`	

Response Body: None.

16.1.11.2 Request Behavior

The Retailer creates a fulfilled Discrete Media Token which SHALL only include:

- The `MediaProfile` element, indicating which Media Profile was used for fulfillment.
- The `DiscreteMediaFulfillment` method used to fulfill the DiscreteMediaRight.
- The `RightsTokenID` element.
- The `State` attribute SHALL be set to `urn:dece:type:state:discretemediaright:fulfilled`

The LeaseExpiration element SHALL NOT be included

16.1.11.3 Response Behavior

Upon successful creation of the fulfilled Discrete Media Token,, the Coordinator shall update the Discrete Media Right with the UserID identified in the provided Delegation Security Token.

16.2 Discrete Media Data Model

16.2.1 DiscreteMediaToken

When created in a RightsToken, the DiscreteMediaToken will carry the ResourceStatus/Current value only. The Coordinator generates all other values.

Element	Attribute	Definition	Value	Card.
DiscreteMediaToken		Describes the lease on a DiscreteMedia right	DiscreteMediaToken-type	
	DiscreteMediaTokenID	A unique, Coordinator-defined identifier for the token.	`xs:anyURI`	0..1
	State	The state of the right. See Table 85 for defined values. This value is set by the Coordinator, except when the DiscreteMediaRightFulfill API is used.	`xs:anyURI`	0..1
RequestingUserID		When a DiscreteMediaRight is leased or fulfilled, indicates the UserID associated with the change.	`dece:EntityID-type`	0..1
RightsTokenID		Indicates the associated Rights Token. Set by the Coordinator, except when the DiscreteMediaRightFulfill API is used.	`xs:anyURI`	
DiscreteMediaFulfillmentMethod		When the Discrete Media Right is fulfilled, the Node sets this value indicating fulfillment method used.	`xs:anyURI`	
AuthorizedFulfillmentMethod		One or more Fulfillment methods authorized for the indicated Rights Token and Media Profile. Valid values are defined in [DDiscrete]. Once the DiscreteMediaRight is consumed, these values may be removed.	`xs:anyURI`	0..n
MediaProfile		This value is derived by the Coordinator from the Rights Token, and is provided here for convenience.	`dece:AssetProfile-type`	0..1
LeaseExpiration		If the DiscreteMediaRight is leased, this indicates when the lease expires.	`xs:dateTime`	0...1
ResourceStatus		The status of the lease. Since the RightsTokenCreate API sets this value, it is mandatory.	`dece:ElementStatus-type`	0..1

Table 83:DiscreteMediaToken Definition

16.2.2 DiscreteMediaTokenList Definition

Element	Attribute	Definition	Value	Card.
DiscreteMediaTokenList		An enumeration of established Discrete Media Rights Tokens	dece:Discrete MediaTokenList-type	
DiscreteMediaToken			dece:Discrete MediaToken-type	0…n

Table 84:DiscreteMediaTokenList Definition

16.2.3 Discrete Media States

State	Definition
urn:dece:type:state:discretemediaright:available	Indicates that a Discrete Media Right may be fulfilled
urn:dece:type:state:discretemediaright:fulfilled	Indicates that a Discrete Media Right has been fulfilled

Table 85: Discrete Media States

16.2.4 Discrete Media Resource Status

Discrete Media Resource Statuses can only be affected by the Coordinator and Coordinator Customer Support roles.

Status	Definition
urn:dece:type:status:active	Indicates that the Discrete Media Right is available for Discrete Media API access (this should not be confused with the state of the Discrete Media Right, defined in table 78).
urn:dece:type:status:deleted	Indicates that a Discrete Media Right has been deleted, and no longer available for lease or fulfillment. This is generally due to an administrative action.
urn:dece:type:status:other	Indicates that a Discrete Media Right is in an indeterminate state, and is no longer available for lease or fulfillment. This is generally due to an administrative action.

Table 86: Discrete Media Resource Status values

16.2.5 DiscreteFulfillmentMethod

The following Fulfillment Methods are defined for use in the FulfillmentMethod in the Discrete Media Right. These methods are derived from Annex A.1 of [DDiscreteMedia].

Fulfillment Method	Definition
urn:dece:type:discretemediaformat:dvd:packaged	The Packaged DVD form of the Approved Discrete Media Fulfillment Method.
urn:dece:type:discretemediaformat:bluray:packaged	The Packaged Blu-ray form of the Approved Discrete Media Fulfillment Method as a packaged fulfillment.
urn:dece:type:discretemediaformat:dvd:cssrecordable	The CSS Recordable DVD form of the Approved Discrete Media Fulfillment Method.
urn:dece:type:discretemediaformat:securedigital	The 3.Recordable SD Card with CPRM to protect standard definition video form of the Approved Discrete Media Fulfillment Method.
urn:dece:type:discretemediaformat:scsa	The SCSA Handle form of the Approved Discrete Media Fulfillment Method.
urn:dece:type:discretemediaformat:nsm	The NSM Media form of the Approved Discrete Media Fulfillment Method.

Table 87: DiscreteMediaFulfillmentMethod

16.3 Discrete Media State Transitions

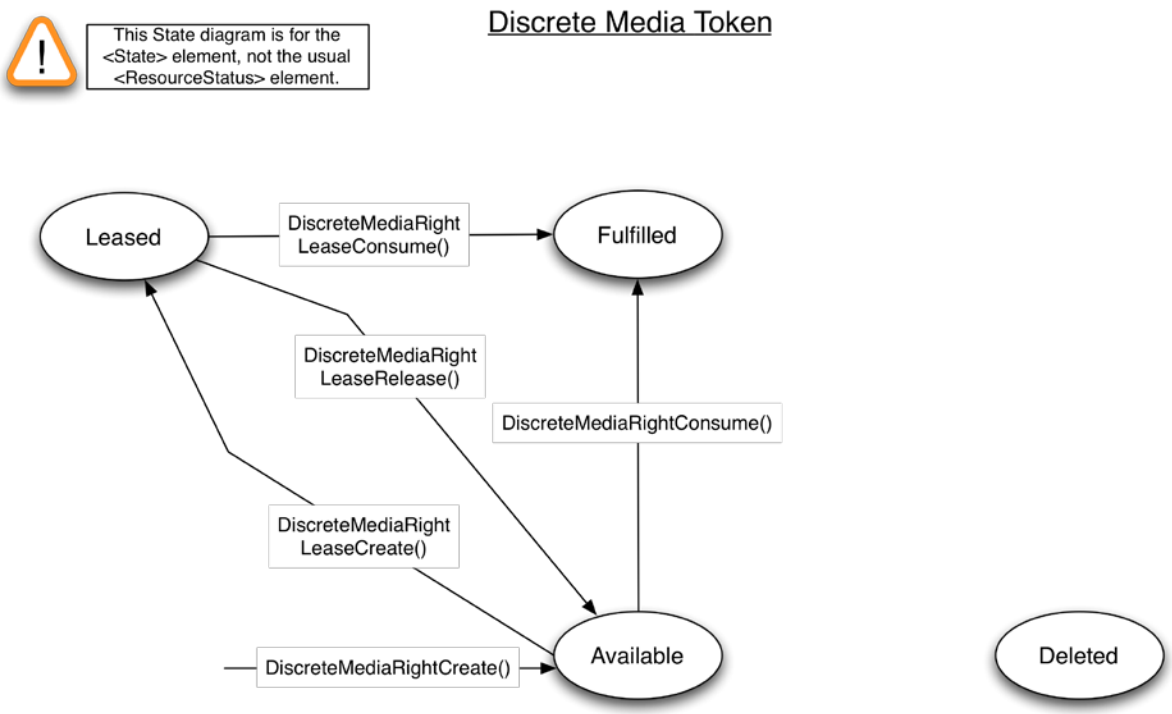


Figure 15: Discrete Media Right State Transitions

17 Other

17.1 Resource Status APIs

17.1.1 StatusUpdate()

17.1.1.1 API Description

This API allows a Resource's status to be updated. Only the Current element of the resource is updated. The prior value of Current will be demoted to the History structure.

17.1.1.2 API Details

Path:

```
{ResourceID}/ResourceStatus/Current/Update
```

Method: PUT

Authorized Role(s):

```
urn:dece:role:dece[:customersupport]
urn:dece:role:coordinator:customersupport
urn:dece:role:portal:customersupport
urn:dece:role:retailer:customersupport
urn:dece:role:accessportal:customersupport
urn:dece:role:lasp:linked:customersupport
urn:dece:role:lasp:dynamic:customersupport
urn:dece:role:contentprovider:customersupport
```

Status of a resource can only be updated by the Customer Support specialization of Nodes authorized to update that resource.

Request Parameters: ResourceID is the absolute path of a Resource

Security Token Subject Scope:

```
urn:dece:user:self
urn:dece:role:user:fullaccess (with further constraints within a given
Geography Policy)
```

Applicable Policy Classes: The applicable Policy Classes depend on the Resource

Request Body: `ResourceStatus`

Response Body: None

17.1.1.3 Behavior

Within the Current structure, the AdminGroup element cannot be updated. The AdminGroup element SHALL NOT be included in the structure sent in the request. All of the other elements of the Current structure SHALL be present. After the Resource's status is updated, the 303 (*See Other*) status code will be returned, and the requester will be provided the URL of the resource whose status was updated via the Location HTTP header.

The StatusUpdate API is the exclusive mechanism for transition of a Resource's Status beyond `pending`, `active` and `deleted`, and generally performed by administrative activities of customer support functions. Each Resource definition section provides a state transition diagram which depicts valid status changes.

Security Token Subject Scope may be further restricted by Geography Policies, but at a minimum, Role restrictions are identical to those specified in the Role Matrix defined in [DSystem] for updating a resource.

Inclusion of ResourceStatus element is optional for Create and Update requests. If included, it shall be ignored by Coordinator, except when otherwise indicated.

The table below indicates the Resources which may be updated using StatusUpdate():

Resource	Transitions		Authorized Roles
	From	To	
Account	pending	Active	coordinator:customersupport dece:customersupport portal:customersupport access:customersupport lasp:*:customersupport retailer:customersupport
	active	blocked	coordinator:customersupport dece:customersupport portal:customersupport
	active	suspended	
	blocked	active	
	suspended	active	
User	active	suspended	coordinator:customersupport dece:customersupport portal:customersupport access:customersupport lasp:*:customersupport retailer:customersupport
	pending	active	
	deleted	blocked:tou	
	blocked:clg	Any except deleted	
	suspended	active	
	blocked:tou	Active	
	deleted	forcedeleted/mergedeleted	coordinator:customersupport dece:customersupport portal:customersupport
	forcedeleted/mergedeleted	blocked:tou	
RightsToken	active	pending	retailer:customersupport
	pending	active	
	active	deleted	
	deleted	active	
	pending	deleted	
	deleted	pending	
Basic, Bundle Assets	pending	active	contentprovider:customersupport
	active	pending	
	active	other	
	other	active	
	active	suspended	
	suspended	active	
	17.1.2 suspended	17.1.3 deleted	
	17.1.4 deleted	17.1.5 active	
Digital Assets	17.1.6 pending	17.1.7 active	contentprovider:customersupport
	17.1.8 active	17.1.9 pending	
Logical Assets	17.1.10active	17.1.11suspended	contentprovider:customersupport
	17.1.12suspended	17.1.13active	
	17.1.14suspended	17.1.15deleted	
	17.1.16deleted	17.1.17active	
	17.1.18	17.1.19	

17.2 ResourceStatus Definition

The ResourceStatus element is used to capture the status of a resource. When an API invocation for a Resource does not include values for relevant status fields (relevance is resource- and context-dependent) the Coordinator SHALL insert the appropriate values.

Element	Attribute	Definition	Value	Card.
ResourceStatus			dece:ElementStatus-type	
Current		Current status of the resource (see Table 89)	dece:Status-type	
History		Prior status values	dece:StatusHistory-type	0..1

Table 88: ElementStatus

17.2.1 Status Definition

Element	Attribute	Definition	Value	Card.
Status			dece:Status-type	
Value		A URI for resource status (defined as a restriction to xs:anyURI). Possible values: urn:dece:type:status:active urn:dece:type:status:archived urn:dece:type:status:blocked urn:dece:type:status:blocked:clg urn:dece:type:status:blocked:tou urn:dece:type:status:deleted urn:dece:type:status:forcedeleted urn:dece:type:status:other urn:dece:type:status:pending urn:dece:type:status:suspended urn:dece:type:status:mergedeleted urn:dece:type:status:deidentified	dece:StatusValue-type	
Description		A free-form description for any additional details about resource status.	xs:String	0..1
	AdminGroup	See Table 96	dece:AdminGroup	0..1
	Modification Group	See Table 97	Dece:ModificationGroup	0..1

Table 89: Status Definition

17.2.2 StatusHistory Definition

Element	Attribute	Definition	Value	Card.
ElementStatus			dece:StatusHistory-type	
Prior		Prior status value	dece:PriorStatus-type	1…n

Table 90: StatusHistory Definition

17.2.3 PriorStatus Definition

Element	Attribute	Definition	Value	Card.
ElementStatus			dece:PriorStatus-type	
	ModificationGroup	See Table 97	dece:ModificationGroup	
Value		Status value	dece:StatusValue-type	
Description			xs:string	

Table 91: PriorStatus Definition

17.3 ResourcePropertyQuery()

17.3.1 API Description

This method offers a general mechanism to retrieve information about resource properties.

A Node can use this method to test the existence of a specific resource property at the Coordinator.

For example,

- A Node can test the availability of a Username, or the existence of an email address within the Coordinator.
- DECE and Coordinator Customer Support Roles can search for Users using various search criteria.

The request is represented by an XPath expression as defined in [XPATH] and further constrained in the sections below. Expressions also include XPath Functions and Operators as defined in [XPATHFN].

Note that this API uses a very narrow subset of XPath. This could be expanded in the future.

17.3.2 API Details

Path:

```
[BaseURL]/Info/
```

Method: POST
Authorized Roles:

```
urn:dece:role:accessportal[:customersupport]
urn:dece:role:dece[:customersupport]
urn:dece:role:coordinator:customersupport
urn:dece:role:lasp:dynamic[:customersupport]
urn:dece:role:lasp:linked[:customersupport]
urn:dece:role:portal[:customersupport]
urn:dece:role:retailer[:customersupport]
```

Request Parameters: None

Security Token Subject Scope: none (no Security Token is required for this API); if provided, it is ignored.

Opt-in Policy Requirements: None

Request Body: XPath expression

Response Body: `UserList-Type` or None

17.3.3 Behavior

A Node indicates the targeted Resource type and the search criteria within the XPath expression. Per [XPATH], the general format can be summarized as follows:

```
//Targeted_Resource_Type[Search_Criteria]
```

17.3.3.1 Targeted Resource Type

Requesting Nodes may target different resource types based on their Role. The table below provides details on Resource accessibility based on the requester's Role.

Targeted Resource Type	XPath Path Expression	Authorized Requester Roles	Response body	Comment
User-type	//User	DECE & Coordinator Customer Support	`UserList`	List of Users by value
		Any other	None	Can check existence, but does not get data

Table 92 Resource Accessibility

17.3.3.2 Search Criteria: XPath Expression

A Search Criteria is an XPath Predicate Expression.

The Coordinator only supports a subset of the XPath expression language. The supported XPath functions and operators are described in the two tables below.

Allowed XPath Expression Component (non Customer Support Role)		Comment
String functions	`fn:matches(`*`$input, $pattern`*`)`	Only alphanumeric strings are supported for $pattern. That is, regular expressions or special characters (^, $) are not supported.
Operators	= predicate operators ([]) path operators (/, //)	
XPath axes	child::	Implicit (need not be included)

Table 93: Supported XPath Expression Components for non Customer Support Role

Allowed XPath Expression Component (Customer Support Role)		Comment
String functions	`fn:matches(`*`$input, $pattern`*`)`	Only alphanumeric strings are supported for $pattern. That is, regular expressions or special characters (^, $) are not supported.
Other functions	`fn:not(`*`arg`*`)`	
Operators	= != and (Boolean operator) predicate operators ([]) path operators (/, //)	

dateTime comparison operators	`op:dateTime-equal()`
	`op:dateTime-less-than()`
	`op:dateTime-greater-than()`
		Noted '>', '<' and '=' in expressions.

XPath axes	child::	Implicit (need not be included)
	attribute::	Abbreviated as '@'
	parent::node()	Abbreviated as '..'

Table 94: Supported XPath Expression Components for Customer Support Role

Requestors SHALL NOT include any other XPath expression language component, as they will not be supported. In particular, XPath axes (other than the ones mentioned in the above tables), node-test (other than the default node() which is implicit) and local path expressions are not supported.

The following XPath Path Expressions MAY be used in the search Expression. The form given in the table is consistent with an implicit 'child::' XPath Axes.

Path Expression	Search Criteria	Substring	Account-scoped
//User	Credentials/Username	Y	N
	ContactInfo/PrimaryEmail/Value	Y	N
	@UserID	N	Y

Table 95: Supported Path Expressions

The table above describes the search criteria (aka. Node selections) that can be used to construct a supported XPath expression. The table's columns provide the following information:

- **Substring**: If "N", only string operators that constitute exact string matches (i.e., = and `!=`)are allowed. When "Y", the XPath [XPATHFN] `fn:matches()` string operator is allowed. Note that the XPath `fn:matches()` string operator returns 'true' when substring matches
- **Account-scoped**: If "Y", the result of this search is limited to a particular Account. If "N" (No), the search criteria is applied to the all resources. For Account-scoped requests, the AccountID is either implicit in the provided criteria (e.g. `AccountRightsLockerID` corresponds to a unique Account) or is explicitly provided within the XPath expression (e.g. `//Account[@AccountID='urn:dece:accountid:org:dece:CB1234']`)

Additional constraints on search criteria are as follows:

- No more than 2 search criteria can be combined together (using XPath's and operator).
- Search values for the //User/Credentials/Username SHALL be at least DCOORD_USERNAME_SEARCH_MIN_LENGTH characters long.
- Search values for the //User/ContactInfo/PrimaryEmail/Value SHALL be at least DCOORD_EMAIL_SEARCH_MIN_LENGTH characters long.
- A maximum of DCOORD_USERLIST_SEARCH_MAX_SIZE matches will be returned for UserList responses
-

Unlike other API calls that return collections, ResourcePropertyQuery() does not support response pagination.Criteria that are not scoped to a specific Account may lead to thousands or more matches. It is **strongly** recommended that search critera be combined using the XPath operator 'and' to reduce the number of matches.

17.3.3.3 Examples

The following are examples of XPath expressions leveraging different search criteria. Examples 1 and 2 can be submitted by either a Customer Support Role or a non-Customer Support Role. Other examples are only for Customer Support Roles.

Example 1 : to search for a list of Users whose primary email address is my_email@example.org.

```
//User[ContactInfo/PrimaryEmail[Value='my_email@example.org']]
```

Example 2: to search for a list of Users whose username is 'Craig'.

```
//User[Credentials[Username='Craig']]
```

Example 3: to search for a list of Users whose username contains 'Hub':

```
//User[Credentials/Username[matches(.,'Hub')]]
```

Example 4: to search for a list of Users whose Username contains 'uBE' but is not 'hubert':

```
//User[Credentials/Username[matches(., 'uBE') and (.!='hubert')]]
```

Responses to the DECE and Coordinator Customer Support Role

If the querying Node dons the `urn:dece:role:dece:customersupport` or `urn:dece:role:coordinator:customersupport` Role, responses may, as appropriate, include a body with a list of element of the targeted resource type.

As with any DECE identifiers (such as UserID) returned by the Coordinator, DECE identifiers are Node-specific to the `urn:dece:role:dece:customersupport` or `urn:dece:role:coordinator:customersupport` Node performing the query. These Node-specific identifiers are to be used by the Node to compose additional queries to the Coordinator. Such responses will be made with the HTTP `200 OK` response status, when successful.

If an error occurs during the validation of the request parameters (other than a 404 `Not Found` error), an HTTP status of 400 will be returned, and an <`ErrorList`> body will be included in the response.

If the Node is not allowed to perform this request, a `403 Forbidden` HTTP response is returned.

If the search does not yield any matches, a `404 Not Found` HTTP response is returned.

Responses to non-DECE and non-Coordinator Customer Support Roles

If an error occurs during the validation of the request parameters (other than a 404 Not Found error), an HTTP status of 400 will be returned, however no <`ErrorList`> body will be included in the response.

Otherwise, the result of the request will be an HTTP response code, as follows:

`300 Multiple Choices` – the search matched more than one resource. No disambiguation information will be provided. This will only be returned for queries targeting `PrimaryEmail`.

`302 Found` - the search matched an existing entry for the targeted resource type.

`400 Bad Request` - the XPath expression is not valid, or the request cannot otherwise be fulfilled.

`403 Forbidden` - the Node is not allowed to perform this request.

`404 Not Found` – the search did not yield any match.

In addition, temporary or permanent redirects may be indicated in the response, as discussed in section 3.

Nodes other than dece and Coordinator Customer Support SHALL NOT use this API for any purpose other than 1) to determine ahead of presenting an option to a user that the intended operation would fail or 2) to provide guidance to a user during Account/User creation. This function is specifically intended to support Account/User creation or assist Customer Support although there may be other uses in the future.

Nodes SHOULD use this API during the Account creation process to determine if a supplied username is already in use and if it is in use.

It is anticipated that Nodes will expose to users input mechanisms that will perform existence queries to the Coordinator using this API. For example, during account create process, assistive techniques to determine if a user already has an Account, or is trying to select an available Username value. This could facilitate attacks such as existence proof attacks and account hijacking attempts. To reduce the risk of automated attacks on this API, Nodes SHALL, in accordance with [DSecMech] 3.4.3, employ a reverse Turing test when the Node detects repeated attempts to obtain information via this API. The Node may implement its own policy, however, at a minimum 3 attempts from the same web page or HTTP session within 5 minutes should be considered repeated attempts.

17.4 Other Data Elements

17.4.1 AdminGroup Definition

The AdminGroup provides a flexible structure to store information about the creation and deletion date (as well as the unique identifier of the entity that performed the operation) of an associated resource. For privacy and security reasons, the information about the author of any creation or deletion (that is, the values of the `Createdby` and `DeletedBy` attributes) must only be present when:

- The requester is the owner of the associated resource.
- The requester is associated to the resource's creator.

Element	Attribute	Definition	Value	Card.
AdminGroup			`dece:AdminGroup`	
	Creation Date		`xs:dateTime`	0..1
	CreatedBy		`dece:EntityID-type`	0..1
	Deletion Date		`xs:dateTime`	0..1
	DeletedBy		`dece:EntityID-type`	0..1

Table 96: AdminGroup Definition

17.4.2 ModificationGroup Definition

The ModificationGroup provides the modification date and identifier for an associated resource. For privacy and security reasons, the information about the author of any creation or deletion (that is, the values of the `Createdby` and `DeletedBy` attributes) must only be present when:

- The requester is the owner of the associated resource.
- The requester is associated to the resource's creator.

Element	Attribute	Definition	Value	Card.
ModificationGroup			`dece:ModificationGroup`	
	Modification Date		`xs:dateTime`	0..1
	ModifiedBy		`dece:EntityID-type`	0..1

Table 97: ModificationGroup Definition

17.5 ViewFilterAttr Definition

The ViewFilter attribute defines a set of attributes used when an offset request has been made. The attributes are defined in section 3.15.

Element	Attribute	Definition	Value	Card.
ViewFilter Attr			dece:ViewFilterAttr-type	
	FilterClass	Any of the filters defined in section **3.15** If more than one filter is used, the filters SHALL be concatenated using '+' as a separator (e.g. urn:dece:type:viewfilter:lastmodifiedd ate+urn:dece:type:viewfilter:userbuyer)	xs:NMTOKENS	0..1
	FilterOffset		xs:NonNegativeInteger	0..1
	FilterCount		xs:int	0..1
	FilterMoreA vailable		xs:boolean	0..1
	FilterDRM		xs:string	0..1

Table 98: ViewFilterAttr Definition

17.6 LocalizedStringAbstract Definition

Element	Attribute	Definition	Value	Card.
Localized String Abstract			dece:LocalizedString Abstract-type extends xs:string	
	Language		xs:language	

Table 99: LocalizedStringAbstract Definition

17.7 KeyDescriptor Definition

The KeyDescriptor element describes the cryptographic keys used to protect communication between the Coordinator and a provisioned Node.

Element	Attribute	Definition	Value	Card.
KeyDescriptor			dece:KeyDescriptor-type	
	use		dece:KeyTypes	0..1
KeyInfo		See [DSecMech] section 5.7	ds:KeyInfo	
EncrytpionMethod		See [XMLENC]	xenc:EncryptionMethod type	

Table 100: KeyDescriptor Definition

17.8 SubDividedGeolocation-type Definition

SubDivided geolocations is a general mechanism which provides varying granularity of a physical location which may be used for windowing, auditing or other purposes. Population of this element should be considered best-effort unless otherwise indicated for a specific purpose.

Element	Attribute	Definition	Value	Card.
SubDividedGeolocation-type			`Extends xs:string` See 0 for potential values.	
	Confidence	An optional indication of the subjective quality of the geolocation value.	`xs:positiveInteger` Value range is 1 to 100, where 1 indicates a very low confidence, and 100 indicates absolute certainty. CalculationMethod will likely inform possible upper bounds of confidence.	0..1
	Calculation Method	A URN indicating the methodology employed to calculate the geolocation string value.	`xs:anyURI` See 17.8.2 for defined values.	

	ViaProxy	A indication on whether or not the submitted believes geography data may have been derived from a network proxy, rather than from the client directly.	`urn:dece:type:true`
			`urn:dece:type:false`
			`urn:dece:type:unknown`
			
			The default value is:
			`urn:dece:type:unknown`
				0..1

Table 101: SubDividedGelocation-type Definition

17.8.1 SubDividedGeolocation Values

The SubDividedGeolocation element, when present, SHALL be populated as follows and in accordance with [ISO3166-1] and [ISO3166-2], using the most precise value available to the Node:

1. ISO 3166-1-alpha-2 code (if no finer detail)
   Examples: Canada = “CA”; United States = “US”; China = “CN”

2. ISO 3166-1-alpha-2 code + space + [postal code]
   Examples: Acadia Valley, Alberta, Canada = “CA T0J 0A0”; Abbeville, Alabama, US = “US 36310”; Shanghai, China (entire municipality) = “CN 200000”; Pudong New District, Shanghai, China = “CN 200120”

3. ISO 3166-2 code (ISO 3166-1-alpha-2 code + "-" + ISO 3166-2 subdivision code [2-3 characters])
   Examples: Alberta, Canada = “CA-AB”; Northwest Territories, Canada = “CA-NT”; Alabama, US = “US-AL”; District of Columbia, US = “US-DC”

Where [postal code] meets local postal code syntax requirements. If the calculation method does not provide a precise postal code (for example it indicates only a province or state but not a city or post office) it is acceptable to omit part of the code for multipart codes (e.g., 98333 instead of 98333-9667 in the U.S. or V5K instead of V5K 1B8 in Canada) or use zeroes (e.g., 200000 or 200100 instead of 200120 in China or 97000 instead of 97604 in the U.S.).

17.8.2 CalculationMethod Values

The calculation method indicates what methodology was employed to determine the supplied SubDividedGeolocation value. The following values are defined:

1. urn:dece:type:geoloc:networkaddress – the calculation method employed a network address to geolocation algorithm (either commercial or proprietary). For example, calculated from a public IP address.
2. urn:dece:type:geoloc:networkderived - the calculation method employed another network-based mechanism. For example, mobile network triangulation.
3. urn:dece:type:geoloc:gps - the calculation method employed an available Global Positioning System – based coordinate.
4. urn:dece:type:geoloc:usersupplied - the calculation method employed a location which was supplied by a user manually
5. urn:dece:type:geoloc:confirmedpostaladdress – the calculation method employed a location which was determined from on a street address known to be valid by the Node. For example, an established street address based on a billing system record.
6. urn:dece:type:geoloc:other – the calculation method employed a location which was determined through another, unspecified means.

17.9 Transaction and TransactionList Definitions

The Transaction element is used to log information about an event. A Node can then retrieve that record in order to support activities like Customer Support.

A Transaction Resource is defined as a `Transaction-type` as follows:

Element	Attribute	Definition	Value	Card.
Transaction			`dece:Transaction-type`	
	transactionDate	Date transaction occurred	`xs:dateTime`	0..1
	transactionID	Unique ID for transaction as defined in Section 3.13.	`xs:string`	0..1
InvokingUserID		Unique identifier of the User on whose behalf the event occurred.	`dece:EntityID-type`	0..1
InvokingNodeID		Unique identifier of the Node that requested the action recorded in this transaction.	`dece:EntityID-type`	
ResourceType		A user-friendly name of the resource type that was accessed during this event.	`xs:string`	
ResourceID		The unique identifier of the resource that was accessed during this event.	`dece:EntityID-type`	

Element	Attribute	Definition	Value	Card.
APIMethod		A user-friendly name of the API method invoked during this event.	xs:string	
RequestURL		The invocation URL as used during this event.	xs:anyURI	
HTTPStatusCode		The HTTP status code returned by the Coordinator.	xs:positiveInteger	
PrimaryErrorCode		If an error occurred, this is the primary error code.	dece:EntityID-type	0..1
PrimaryErrorMessage		If an error occurred, this is the message that accompanies the primary error code.	xs:string	0..1
Description		A human-friendly description of the transaction. This will not necessarily be populated in the near-term.	xs:string	0..1

Table 102: Transaction Definition

A TransactionList is a list of Transactions.

Element	Attribute	Definition	Value	Card.
TransactionList			dece:TransactionList-type	
	AccountID		dece:EntityID-type	0..1
Transaction		A transaction record.	dece:Transaction-type	0..n

Table 103: TransactionList Definition

18 Push Notification

The existing mechanism for Nodes (Retailers, LASPs, and Access Portals) to use ETag-based polling to retrieve data from the Coordinator and manage updates is supplemented with a publish/subscribe framework. Changes to a Rights Locker or UserLinkConsent Policies will be the first events using this service. Additional notifications may be added at a future date.

Each event type will have its own Topic and Nodes will have their own set of Topics that they can subscribe to. The Coordinator acts as the event publisher and publishes Notification events, via one or more established Pub/Sub systems. Appendix I documents the details of each supported system.

The Coordinator publishes one topic per event class per Node. Nodes can request to subscribe to push either during onboarding or subsequently by contacting the Coordinator.

For each supported event type listed below, the Coordinator publishes a message to each Node, and includes UserIDs for each User having the UserLinkConsent policy in place. For example, if an account has two users and three linked nodes, for any change to RightsToken(s) in the account, there will be three event messages generated, one event generated for each of the three linked nodes.

Nodes can call the RightsLockerDataGet and RightsTokenGet APIs to update their caches.

18.1.1 Supported Event Classes

Presently, 2 event classes are provided: Rights Locker changes and the removal of UserLinkConsent policies. Other event classes may be supported in future.

18.1.1.1 Rights Locker Notifications

Notifications will be initiated as a result of any change to the Locker collection, including the addition, deletion or update of a Right in an account.

All Nodes that have indicated support for notifications, and that have an active UserLinkConsent Policy in place for the associated Account will receive Rights Locker Notifications, including the Node that caused the Notification event to occur.

18.1.1.2 PolicyDelete Notifications

This Notification is the result of the deletion of the UserLinkConsent policy for a User for one or more Nodes.

All Nodes that have indicated support for notifications, have an active UserLinkConsent Policy in place for the associated Account, and belong to the same Organization will receive notifications.

18.1.2 Eligibility for Subscriptions

The following Nodes are eligible for establishing subscriptions for notifications:

- `urn:dece:role:retailer[:customersupport]`
- `urn:dece:role:lasp[:customersupport]`
- `urn:dece:role:accessportal[:customersupport]`
- `urn:dece:role:portal[:customersupport]`
- `urn:dece:role:dece:customersupport`

18.1.3 Event Data Structures

The notification message is compact and includes information both at the message level (one for each collection of events) and event level (repeats for each event in the collection).

This section describes the Event resource model of the `EventList-type` complex type.

18.1.3.1 EventList-type Definition

The Event list collection captures all class of events for which Coordinator triggers notifications. It is conveyed in the Events element, which holds a list of individual Event elements (as defined in section 18.1.3.2).

Element	Attribute	Definition	Value	Card.
Events			`dece:EventList-type`	
	EventType	Identifies the class of event for which the notification is triggered.	`xs:string`	
	SubscriptionID	This attribute is reserved for future use.	`xs:string`	0..1
	SubscriptionContext	Opaque data provided by Node during the subscription process.	`xs:string`	0…1
	MessageID	Unique ID generated by Coordinator or by individual Notification systems	`xs:ID`	0…1
Event		See below.	`dece:EventData-type`	0..n

Table 133: EventList-type Definition

18.1.3.2 Event Type Definition

The following table describes the EventData-type complex type

Element	Attribute	Definition	Value	Card.
Event		An individual event description, that conveys sufficient detail for a Node to update local caches based on what has changed.	dece:EventData-type	
	ChangeDateTime	The date and time when the REST request generating the event was completed by the Coordinator	xs:dateTime	
EventParam		A URI reference to the associated change. This may be a URN identifier or a relative URI to the resource.	xs:anyURI	1..n
	name	See defined values below.	xs:string	

Table 134: Event-type Definition

18.1.3.3 Name Attribute Values

The EventParam's name attribute has 4 possible values:

- AccountID – the unique identifier for the Account
- UserID – the unique identifier for the User in associated Account
- RightsTokenID – the unique identifier for the RightsToken in associated Account
- ResourcePath – A relative path to the resource to be dereferenced. This ResourcePath does not include the endpoint version information, allowing Nodes to prepend that based on which release of the Coordinator the Node presently supports. The ResourcePath value also does not include any optional query parameters

18.1.4. Notification Payload Example

The following example includes two UserID values, both linked the target Node.

```
<Events EventType=”RightsLockerUpdate” MessageID="ABC123" SubscriptionContext="Optional
node supplied data">
   <Event ChangeDateTime="2015-03-17T08:22:59Z">
   <EventParam name="AccountID">
          urn:dece:accountid:org:dece:BBF31BCEEF60890AE0405B0A09344071
      </EventParam>
   <EventParam name="UserID">
          urn:dece:userid:org:dece:0000000000000000E0405B0A09344058
    </EventParam>
```

```
   <EventParam name="UserID">
          urn:dece:userid:org:dece:0000000000000000E0405B0A0934ABCD
    </EventParam>
<EventParam name="RightsTokenID">
          urn:dece:rightstokenid:org:dece:150A7D369BEA1497E0530F345A0A77D7
    </EventParam>
    <EventParam name="ResourcePath">
  /Account/urn:dece:accountid:org:dece:BBF31BCEEF60890AE0405B0A09344071/RightsToken/List
      </EventParam>
   </Event>
</Events>
```

19 Error Management

This section defines the error responses to Coordinator API requests.

19.1 ResponseError Definition

The `ResponseError-type` is used as part of each response element to describe error conditions. This appears as an Error element. ErrorID is an integer assigned to an error that uniquely identifies the error condition. Reason is a text description of the error in English. In the absence of more descriptive information, this should be the title of the error, as defined in section 3.14. OriginalRequest is a string containing information from the request.

Element	Attribute	Definition	Value	Card.
ResponseError			`dece:ResponseError-type`	
	ErrorID	HTTP error status code	`xs:anyURI`	
Reason		Human-readable explanation of reason. English being the only language used for error reporting, the <Language> attribute SHALL be set accordingly.	`dece:LocalizedString Abstract-type`	
OriginalRequest		The request that generated the error. This includes the URL but not information provided in the original HTTP request.	`xs:string`	
ErrorLink		URL for a detailed explanation of the error with possible self-help instructions.	`xs:anyURI`	0..1

Table 104: ResponseError Definition

20 Appendix A: API Invocation by Role

The following table lists all the APIs in the system, divided into sections and alphabetized within each section. The Roles that may invoke the APIs are listed across the top. The markings indicate that the Node may invoke the API, and the annotations provide additional information about the Node's invocation of the API.

		DECE	DECE Customer Support+	Coordinator	Coordinator Customer+	Web Portal	Web Portal Customer+	Retailer	Retailer Customer+	Access Portal	Access Portal Customer+	Linked LASP	Linked LASP Customer+	Dynamic LASP	Dynamic LASP Customer+	DSP	DSP Customer Support+	Content Provider	Content Provider+	Basic-Access User*	Standard-Access User*	Full-Access User*
	AccountUserCreate		●		●	●	●	●	●			●	●	●	●					n/a	n/a	n/a
	AccountDelete		●		●	●	●	●	●[3]			●	●[3]	●	●[3]							●
	AccountGet	●	●	●	●	●	●	●	●	●	●	●	●	●	●					●	●	●
	AccountUpdate		●	●	●	●	●	●[3]	●[3]			●[3]	●[3]	●[3]	●[3]							●
	AccountMergeTest		●		●	●	●	●	●	●	●	●	●	●	●							●
	AccountMerge		●		●	●	●	●	●	●	●	●	●	●	●							●
	AccountMergeUndo		●		●		●		●		●		●		●							●
Discrete Media	DiscreteMediaRightConsume							●	●											●	●	●
	DiscreteMediaRightCreate							●	●													
	DiscreteMediaRightDelete							●[1]	●[1]													
	DiscreteMediaRightFulfill							●	●													
	DiscreteMediaRightGet[10]	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●			●	●	●
	DiscreteMediaRightList[10]	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●			●	●	●
	DiscreteMediaRightUpdate							●[1]	●[1]													
Le ga	LegacyDeviceCreate							●[1]	●[1]												●	●

	DECE	DECE Customer Support+	Coordinator	Coordinator Customer+	Web Portal	Web Portal Customer+	Retailer	Retailer Customer+	Access Portal	Access Portal Customer+	Linked LASP	Linked LASP Customer+	Dynamic LASP	Dynamic LASP Customer+	DSP	DSP Customer Support+	Content Provider	Content Provider+	Basic-Access User*	Standard-Access User*	Full-Access User*
LegacyDeviceDelete		●		●			●[1]	●[1]												●	●
LegacyDeviceGet	●	●	●	●	●	●	●[1]	●[1]											●	●	●
LegacyDeviceUpdate							●[1]	●[1]												●	●

		DECE	DECE Customer Support+	Coordinator	Coordinator Customer+	Web Portal	Web Portal Customer+	Retailer	Retailer Customer+	Access Portal	Access Portal Customer+	Linked LASP	Linked LASP Customer+	Dynamic LASP	Dynamic LASP Customer+	DSP	DSP Customer Support+	Content Provider	Content Provider+	Basic-Access User*	Standard-Access User*	Full-Access User*
Metadata	MetadataBasicList	●	●		●	●	●	●	●	●	●	●	●	●	●			●	●	n/a	n/a	n/a
	MetadataDigitalList	●	●		●	●	●	●	●	●	●	●	●	●	●			●	●	n/a	n/a	n/a
	LogicalAssetList	●	●		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	n/a	n/a	n/a
	LogicalAssetDelete																	●[1]	●[1]	n/a	n/a	n/a
	AssetMapALIDtoAPIDGet	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●[4]	●[4]	●[4]
	AssetMapAPIDtoALIDGet	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●[4]	●[4]	●[4]
	MapALIDtoAPIDCreate																	●	●	n/a	n/a	n/a
	MapALIDtoAPIDUpdate																	●[1]	●[1]	n/a	n/a	n/a
	BundleCreate							●	●									●	●	n/a	n/a	n/a
	BundleDelete							●[1]	●[1]									●[1]	●[1]	n/a	n/a	n/a
	BundleGet	●	●	●	●	●	●	●	●	●	●	●	●	●	●			●	●	●[4]	●[4]	●[4]
	BundleUpdate							●[1]	●[1]									●[1]	●[1]	n/a	n/a	n/a
	MetadataBasicCreate																	●	●	n/a	n/a	n/a
	MetadataBasicDelete																	●[1]	●[1]	n/a	n/a	n/a
	MetadataBasicGet	●	●	●	●	●	●	●	●	●	●	●	●	●	●			●	●	●[4]	●[4]	●[4]
	MetadataBasicUpdate																	●[1]	●[1]	n/a	n/a	n/a

		DECE	DECE Customer Support+	Coordinator	Coordinator Customer+	Web Portal	Web Portal Customer+	Retailer	Retailer Customer+	Access Portal	Access Portal Customer+	Linked LASP	Linked LASP Customer+	Dynamic LASP	Dynamic LASP Customer+	DSP	DSP Customer Support+	Content Provider	Content Provider+	Basic-Access User*	Standard-Access User*	Full-Access User*
	MetadataDigitalCreate																	●	●	n/a	n/a	n/a
	MetadataDigitalDelete																	●[1]	●[1]	n/a	n/a	n/a
	MetadataDigitalGet	●	●	●	●	●	●	●	●	●	●	●	●	●	●			●	●	●[4]	●[4]	●[4]
	MetadataDigitalUpdate																	●[1]	●[1]	n/a	n/a	n/a
Nodes	NodeCreate				●																	
	NodeGet	●	●		●	●	●	●	●	●	●	●	●	●	●					n/a	n/a	n/a
	NodeList	●	●		●	●	●	●	●	●	●	●	●	●	●							
	NodeUpdate				●																	
	NodeDelete				●																	
	OrganizationGet	●	●	●	●	●	●	●	●	●	●	●	●	●	●					n/a	n/a	n/a
Policies	PolicyGet	●	●	●	●	●	●	●	●	●	●	●	●	●	●					●	●	●
	PolicyCreate	●	●	●	●	●	●	●	●	●	●	●	●	●	●					●	●	●
	PolicyUpdate	●	●	●	●	●	●	●	●	●	●	●	●	●	●					●	●	●
	PolicyDelete	●	●	●	●	●	●	●	●	●	●	●	●	●	●					●	●	●
Rights Tokens	RightsLockerDataGet	●	●	●	●	●	●	●[1]	●[1]	●[1]	●[1]	●[1]	●[1]	●[1]	●[1]					●[1]	●[1]	●[1]
	RightsTokenDataGet	●	●	●	●	●	●	●[1]	●[1]	●[1]	●[1]	●[1]	●[1]	●[1]	●[1]	●[11]	●[11]			●[1]	●[1]	●[1]
	RightsTokenCreate							●	●											●	●	●

		DECE	DECE Customer Support+	Coordinator	Coordinator Customer+	Web Portal	Web Portal Customer+	Retailer	Retailer Customer+	Access Portal	Access Portal Customer+	Linked LASP	Linked LASP Customer+	Dynamic LASP	Dynamic LASP Customer+	DSP	DSP Customer Support+	Content Provider	Content Provider+	Basic-Access User*	Standard-Access User*	Full-Access User*
	RightsTokenListCreate							●	●											●	●	●
	RightsTokenDelete				●	●	●	●[1]	●[1]											●[1]	●[1]	●[1]
	RightsTokenGet	●	●	●	●	●	●	●[1]	●[1]	●	●	●	●	●	●					●[1]	●[1]	●[1]
	RightsTokenUpdate		●		●			●[1]	●[1]											●	●	●
	DownloadPlaybackLicenseReporting							●	●							●	●					
Resources	StatusUpdate		●		●		●[10]		●[10]		●[10]		●[10]		●[10]				●[10]			
Security Tokens	STS Service (UserPassword profile)		●					●		●		●		●	●					●	●	●
	STS Service (SAML2 profile)	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●			●	●	●
Streams	StreamCreate											●	●	●	●						●	●
	StreamDelete											●[1]	●[1]	●[1]	●[1]						●	●
	StreamListView	●	●	●	●	●	●	●	●	●	●	●[1]	●[1]	●[1]	●[1]					●[1]	●[1]	●[1]
	StreamRenew											●[1]	●[1]	●[1]	●[1]						●	●
	StreamView	●	●	●	●	●	●	●	●	●	●	●[1]	●[1]	●[1]	●[1]					●[1]	●[1]	●[1]
	ResourcePropertyQuery	●	●		●	●	●	●	●	●	●	●	●	●	●					●	●	●
Users	UserCreate		●		●	●	●	●[3]	●[3]	●[3]	●[3]	●[3]	●[3]	●[3]	●[3]						●	●
	UserDelete	●	●	●	●	●	●	●[3]	●[3]	●[3]	●[3]	●[3]	●[3]	●[3]	●[3]						●	●

	DECE	DECE Customer Support†	Coordinator	Coordinator Customer†	Web Portal	Web Portal Customer†	Retailer	Retailer Customer†	Access Portal	Access Portal Customer†	Linked LASP	Linked LASP Customer†	Dynamic LASP	Dynamic LASP Customer†	DSP	DSP Customer Support†	Content Provider	Content Provider†	Basic-Access User*	Standard-Access User*	Full-Access User*
UserGet	●	●	●	●	●	●	●[3]	●[3]	●[3]	●[3]	●[3]	●[3]	●[3]	●[3]					●	●	●
UserList	●	●	●	●	●	●	●[3]	●[3]	●[3]	●[3]	●[3]	●[3]	●[3]	●[3]					●	●	●
UserUpdate	●	●	●	●	●	●	●[3]	●[3]	●[3]	●[3]	●[3]	●[3]	●[3]	●[3]					●[9]	●	●
UserValidationTokenCreate	●	●	●	●	●	●	●	●	●	●	●	●	●	●					●	●	●

Notes on the API Invocation by Role Table

† The customer support role always interprets the security context at the account level.

* When composed with a Role, the entries indicate the user classification that is necessary to initiate the API request using the Node.

[1] The Node may perform operations (using the API) only on objects created by the Node and by its associated customer support role (and vice versa).

[2] In the absence of policies altering the API's behavior, the response will be limited to objects created by the Node. The API's response will vary according to the Role.

[3] A successful API invocation requires explicit consent (at the user level, at the account level, or both).

[4] The API's response varies according to the Role.

[5] The API's response depends on which Policies (if any) have been applied to the User, the object, or both.

[7] Nodes may manipulate the listed policy on behalf of full-access Users only. Requires the application of the Account-level EnableManageUserConsent Policy as well as the User-level ManageUserConsent Policy.

[8] Limited to the `urn:dece:role:user:self` and `urn:dece:role:user:parent` pseudo-classes

[9] Limited the `urn:dece:role:user:class:self` pseudo-class

[10] Limited to the Customer Support specialization of the Roles authorized to update that resource type. This also requires that the appropriate consent policies are in place.

[11] Refer to the API definition for specific limitations for the identified role(s).

21 Appendix B: Error Codes

All of the Coordinator's error codes are prefixed with `urn:dece:errorid:org:dece:`

21.1 Coordinator API Error Messages

API	Error ID	Reason	Status
AccountDelete	AccountDeleted	The account has already been removed.	404
AccountDelete	AccountNotActive	The account is not active.	403
AccountDelete	NodeUnauthorizedToActOnAccount	The request is not authorized.	401
AccountDelete	RequestorNotActive	The requestor is not active.	403
AccountDelete	RequestorNotFound	The requestor was not found.	404
AccountDelete	RequestorPrivilegeInsufficient	You do not have permission to perform this action. Ask a full access member of your account for help.	403
AccountDelete	SecurityTokenDeleteFailed	The security tokens associated with the licensed application was not removed.	500
AccountDelete	UserSAMLTokenDeleteFailed	Deletion of the member's security token failed.	500
AccountGet	NodeUnauthorizedToActOnAccount	The request is not authorized.	401
AccountGet	RequestorNotActive	The requestor is not active.	403
AccountGet	RequestorNotFound	The requestor was not found.	404
AccountMerge	AccountActiveUserCountReachedMaxLimit	The maximum number of active members allowed has been reached.	400
AccountMerge	AccountIDNotValid	The account ID is not valid.	400
AccountMerge	AtleastOneOfTheRequestorsMustBeRetained	At least one of the signed-in members must be retained.	403
AccountMerge	BadRequest	The request is not valid.	400
AccountMerge	CountriesOfMergingAccountsDoNotMatch	The accounts being merged must be from the same country.	403
AccountMerge	DeviceLimitExceeded	The merging of these accounts would result in the maximum number of allowed devices being exceeded.	403
AccountMerge	MergedAccountRequiresAtleastOneActiveFAU	The account resulting from the merge must have at least one active full-access member.	400
AccountMerge	RequestorNotActive	The requestor is not active.	403

API	Error ID	Reason	Status
AccountMerge	RequestorPrivilegeInsufficient	You do not have permission to perform this action. Ask a full access member of your account for help.	403
AccountMerge	SurvivingAccountCannotBeSameAsRetiringAccount	The accounts being merged cannot be the same.	403
AccountMerge	UserListEmpty	The user list is empty.	400
AccountMerge	UserListHasDuplicatedUserID	The user list contains duplicate user IDs.	400
AccountMerge	UserNotFound	The user ID was not found.	404
AccountMerge	UsersMissingInRequest	The user list does not identify all users in the accounts being merged.	400
AccountMergeTest	AccountActiveUserCountReachedMaxLimit	The maximum number of active members allowed has been reached.	400
AccountMergeTest	AccountIDNotValid	The account ID is not valid.	400
AccountMergeTest	AtleastOneOfTheRequestorsMustBeRetained	At least one of the signed-in members must be retained.	403
AccountMergeTest	CountriesOfMergingAccountsDoNotMatch	The accounts being merged must be from the same country.	403
AccountMergeTest	DeviceLimitExceeded	The merging of these accounts would result in the maximum number of allowed devices being exceeded.	403
AccountMergeTest	MergedAccountRequiresAtleastOneActiveFAU	The account resulting from the merge must have at least one active full-access member.	400
AccountMergeTest	RequestorNotActive	The requestor is not active.	403
AccountMergeTest	RequestorPrivilegeInsufficient	You do not have permission to perform this action. Ask a full access member of your account for help.	403
AccountMergeTest	SurvivingAccountCannotBeSameAsRetiringAccount	The accounts being merged cannot be the same.	403
AccountMergeTest	UserListEmpty	The user list is empty.	400
AccountMergeTest	UserListHasDuplicatedUserID	The user list contains duplicate user IDs.	400
AccountMergeTest	UserNotFound	The user ID was not found.	404
AccountMergeTest	UsersMissingInRequest	The user list does not identify all users in the accounts being merged.	400
AccountMergeUndo	AccountIDNotValid	The account ID is not valid.	400
AccountMergeUndo	AccountMergeAlreadyUndone	The account merge has already been undone, and cannot be performed again.	403
AccountMergeUndo	AccountNotPreviouslyMerged	The account merge cannot be undone because the identified account has not been merged with another account.	403

API	Error ID	Reason	Status
AccountMergeUndo	MergeUndoPeriodExceeded	The merge undo period has been exceeded.	403
AccountMergeUndo	MergeUndoPoliciesNotMet	Policies that allow a merge to be undone are not met.	403
AccountMergeUndo	RequestorNotActive	The requestor is not active.	403
AccountMergeUndo	RequestorPrivilegeInsufficient	You do not have permission to perform this action. Ask a full access member of your account for help.	403
AccountUpdate	AccountCannotBeNull	The account name is required.	400
AccountUpdate	AccountCountryCodeCannotBeNull	The country code is required.	400
AccountUpdate	AccountCountryCodeNotValid	The country code is not valid.	400
AccountUpdate	AccountDisplayNameNotValid	The display name is not valid.	400
AccountUpdate	AccountIDNotValid	The account ID is not valid.	400
AccountUpdate	AccountStatusNotActive	The account is not active.	403
AccountUpdate	CountryCannotBeChangedOnceSet	The country cannot be changed.	400
AccountUpdate	NodeUnauthorizedToActOnAccount	The request is not authorized.	401
AccountUpdate	RequestorNotActive	The requestor is not active.	403
AccountUpdate	RequestorNotFound	The requestor was not found.	404
AccountUpdate	RequestorPrivilegeInsufficient	You do not have permission to perform this action. Ask a full access member of your account for help.	403
AccountUserCreate	AccountCountryCodeCannotBeNull	The country code is required.	400
AccountUserCreate	AccountCountryCodeNotValid	The country code is not valid.	400
AccountUserCreate	AccountDisplayNameNotValid	The display name is not valid.	400
AccountUserCreate	UserInformationRequired	The User Information is Required.	400
AccountUserCreate	UserListCannotHaveMoreThanOneUser	UserList can not have more than one user set in the request.	403
AccountUserCreate	AccountUserAddressNotValid	The address is not valid.	400
AccountUserCreate	AccountUserAlternateEmailNotValid	The alternate email address is not valid.	400
AccountUserCreate	AccountUserCountryNotValid	The country is not valid.	400
AccountUserCreate	AccountUserEmailAddressDuplicated	The email address is a duplicate.	400
AccountUserCreate	AccountUserGivenNameNotValid	The given name is not valid.	400
AccountUserCreate	AccountUserLanguageDuplicated	The language is a duplicate.	400
AccountUserCreate	AccountUserLanguageNotValid	The language is not valid.	400
AccountUserCreate	AccountUserMobilePhoneNumberNotValid	The mobile telephone number is not valid.	400
AccountUserCreate	AccountUsernameRegistered	The sign-in name already exists.	400
AccountUserCreate	AccountUserPasswordNotValid	The password is not valid.	400
AccountUserCreate	AccountUserPrimaryEmailNotValid	The primary email address is not valid.	400
AccountUserCreate	AccountUserPrimaryLanguageNotValid	The primary language is not valid.	400

API	Error ID	Reason	Status
AccountUserCreate	AccountUserSecurityAnswerNotValid	The answer to the security question is not valid.	400
AccountUserCreate	AccountUserSecurityQuestionDuplicated	The security question is a duplicate.	400
AccountUserCreate	AccountUserSecurityQuestionIDNotValid	The security question is not valid.	400
AccountUserCreate	AccountUserSurnameNotValid	The surname is not valid.	400
AccountUserCreate	AccountUserTelephoneNumberNotValid	The telephone number is not valid.	400
AccountUserCreate	FirstUserMustBeCreatedWithFullAccessPrivilege	The first member must be a full-access member.	403
AccountUserCreate	PrimaryEmailConfirmationEndpointRequired	A confirmation endpoint is required for the member primary email address.	400
AccountUserCreate	PrimaryEmailVerifiedAttributeMustBeTrue	If the member's primary email address has been verified by the node, the setting of the PrimaryEmailVerified attribute must be set to TRUE.	400
AccountUserCreate	UserPrimaryEmailVerificationDateNotValid	The verification date for the member's primary email address is not valid.	400
AccountUserCreate	UserPrimaryEmailVerificationEntityNotValid	The node that verified the member's primary email address is not valid.	400
AccountUserCreate	UserPrimaryEmailVerificationEntityRequired	The node that verified the member's primary email address must be identified.	400
AccountUserCreate	UserPrimaryEmailVerificationStatusRequired	The verification status is required.	400
AccountUserCreate	ValidPrimaryEmailVerificationDateRequired	The verification date for the user primary email address is required.	400
AccountUserCreate	VerificationStatusNotConsistentWithVerifiedAttributeDeclaration	The verification status is not consistent with the declaration of a verified attribute.	400
AssetMapALIDtoAPIDCreate	ActiveApidDoesNotExist	The physical asset (APID) was not found.	404
AssetMapALIDtoAPIDCreate	ActiveApidInvalid	The physical asset (APID) is not valid.	400
AssetMapALIDToAPIDCreate	AlidNotMatchingWiththeXMLAlid	The logical asset (ALID) does not match.	403
AssetMapALIDtoAPIDCreate	AssetLogicalIDNotFound	The logical asset (ALID) was not found.	404
AssetMapALIDtoAPIDCreate	AssetProfileDoesNotExist	The asset profile was not found.	404
AssetMapALIDtoAPIDCreate	AssetProfileInvalid	The asset profile is not valid.	400
AssetMapALIDtoAPIDCreate	DuplicateAPIDNotAllowed	The APIDs are duplicates.	400
AssetMapALIDToAPIDCreate	LogicalAssetAlreadyExist	The logical asset already exists.	409
AssetMapALIDToAPIDCreate	MdNodeIdDiffrentFromCreateRequest	The node did not create the resource.	400

API	Error ID	Reason	Status
AssetMapALIDToAPIDCreate	MediaProfileNotMatchingWiththeXMLMediaProfile	The media profile does not match.	403
AssetMapALIDtoAPIDCreate	RecalledAPIDDoesNotExist	The recalled physical asset (APID) was not found.	404
AssetMapALIDtoAPIDCreate	RecalledAPIDInvalid	The replaced physical asset (APID) is not valid.	400
AssetMapALIDtoAPIDCreate	ReplacedAPIDDoesNotExist	The replaced physical asset (APID) was not found.	404
AssetMapALIDtoAPIDCreate	ReplacedAPIDInvalid	The replaced physical asset (APID)is not valid	400
AssetMapALIDToAPIDCreate	RestrictionTypeDoesNotExist	The supplied restriction type was not found.	404
AssetMapALIDtoAPIDCreate	RestrictionTypeInvalid	The identified restriction type is invalid	400
AssetMapALIDtoAPIDGet	AssetIdentifierNotValid	The physical asset (APID) or the logical asset (ALID) is not valid.	400
AssetMapALIDtoAPIDGet	AssetLogicalIDNotFound	The logical asset (ALID) was not found.	404
AssetMapALIDtoAPIDGet	AssetPhysicalIDNotFound	The physical asset (APID) was not found.	404
AssetMapALIDtoAPIDGet	AssetProfileInvalid	The asset profile is not valid.	400
BundleCreate	BundleIDNotValid	The bundle is not valid.	400
BundleDelete	BundleIDNotValid	The bundle is not valid.	400
BundleGet	BundleIDNotValid	The bundle is not valid.	400
BundleResourceStatusUpdate	BundleIDNotValid	The bundle is not valid.	400
Common	AccountIdUnmatched	The account ID does not match.	403
Common	AccountNotFound	The account ID was not found.	404
Common	AccountNotActive	The identified account is not active.	400
Common	AccountUsernameNotValid	The sign-in name is not valid.	400
Common	AdminAccessDenied	Administrative access has been denied.	401
Common	AdultContentNotAllowed	The member does not have permission to access this content because of its rating.	403
Common	APIDInvalid	The physical asset (APID) is not valid.	400
Common	AssetLogicalIDNotValid	The logical asset (ALID) is not valid.	400
Common	AuthnRequestNotValid	The authentication request not valid.	400
Common	ContactIdInvalid	The contact ID is not valid.	400
Common	ContentIDNotActive	The content is not active.	403
Common	ContentIDNotFound	The content ID was not found.	404
Common	ContentIDNotValid	The content is not valid.	400
Common	DiscreteMediaFulfillmentMethodInvalid	The discrete media fulfillment method is not valid.	400
Common	EnableUserDataUsageConsentRequired	The setting of the EnableUserDataUsageConsent policy prevents the requested action from being completed.	403

API	Error ID	Reason	Status
Common	Forbidden	The requesting node is not allowed to perform this request.	403
Common	InternalServerError	An internal server error occurred.	500
Common	InternalServerErrorRetry	Please submit the request again.	500
Common	InvalidBaseLocationDelegationName	The base location delegation name is invalid	400
Common	InvalidBaseLocationDelegationNameServer	The base location delegation name server is invalid	400
Common	InvalidLogoResourceWidthOrHeight	The logo's resource width or height is invalid	400
Common	InvalidScheme	The scheme is not valid.	400
Common	InvalidSSID	The schema-specific identifier is not valid.	400
Common	InvocationPathHasNonEncodedParameters	The parameters in the invocation path must be escape-encoded.	400
Common	InvocationTargetException	The method parameter types are not valid.	400
Common	LockerViewAllConsentRequired	The setting of the LockerViewAllConsent policy prevents the requested action from being completed.	403
Common	ManageAccountConsentRequired	The setting of the ManageAccountConsent policy prevents the requested action from being completed.	403
Common	ManageUserConsentRequired	The setting of the ManageUserConsent policy prevents the requested action from being completed.	403
Common	MandatoryFieldCannotBeNullOrEmpty	This field cannot be empty or null.	400
Common	MethodNotSupported	The requested method is not supported.	405
Common	NodeIdInvalid	The node ID is not valid.	400
Common	NodeIdUnmatched	The node ID does not match.	400
Common	NodeNotActive	The node is not active.	403
Common	NodeNotFound	The node ID was not found.	404
Common	NotFound	The requested resource was not found.	404
Common	RatingPolicyExists	The member does not have permission to access this content because of its rating.	403
Common	RightsTokenIDNotValid	The rights token ID is not valid.	400
Common	RoleInvalid	The API call is not authorized.	403
Common	SaxParserException	DECE parser exception.	400
Common	Unauthorized	The request is not authorized.	401
Common	UnexpectedXmlForbidden	The URL does not match.	403
Common	UnratedContentBlocked	The member does not have permission to access this content because it is unrated.	403

API	Error ID	Reason	Status
Common	UserDataUsageConsentRequired	The setting of the UserDataUsageConsent policy prevents the requested action from being completed.	403
Common	UserIdInvalid	The user ID is not valid.	400
Common	UserIdUnmatched	The user ID does not match.	403
Common	UserLinkConsentRequired	The setting of the UserLinkConsent policy prevents the requested action from being completed.	403
Common	UserNotActive	The member is not active.	403
Common	UserNotFound	The user ID was not found.	404
ContactCreate	ConfirmationEndPointNotValid	The confirmation end point is not valid.	400
ContactCreate	ContactAlternateEmailInvalid	The contact's alternate email is not valid.	400
ContactCreate	ContactGivenNameInvalid	The contact's given name is not valid.	400
ContactCreate	ContactMobilephoneNumberInvalid	The contact's mobile phone number is not valid.	400
ContactCreate	ContactPrimaryEmailInvalid	The contact's primary email is not valid.	400
ContactCreate	ContactSurnameInvalid	The contact surname is not valid.	400
ContactCreate	ContactTelephoneNumberInvalid	The contact's telephone number is not valid.	400
ContactCreate	LocalityNotValid	The locality is not valid.	400
ContactCreate	PostalAddressNotValid	The postal address is not valid.	400
ContactCreate	PostalCodeNotValid	The postal code is not valid.	400
ContactCreate	ResourceStatusTransitionRequestedNotAllowed	The requested status transition is not allowed for the resource.	403
ContactCreate	StateOrProvinceNotValid	The state or province is not valid.	400
ContactDelete	ContactDeleteConflict	The last remaining contact for a node or organization cannot be removed.	401
ContactDelete	ContactDoesNotExist	The contact was not found.	404
ContactGet	ContactNotFound	The contact was not found.	404
ContactResourceStatusUpdate	BadGateWay	The request cannot be fulfilled because of a server error..	502
ContactResourceStatusUpdate	ContactNotFound	The contact was not found.	404
ContactResourceStatusUpdate	ResourceAlreadyinSameStatus	The resource is already in the requested status.	409
ContactResourceStatusUpdate	ResourceCurrentStatusValueRequired	The resource's current status is required.	400
ContactResourceStatusUpdate	ResourceStatusTransitionRequestedNotAllowed	The requested status transition is not allowed for the resource.	403
ContactUpdate	ConfirmationEndPointNotValid	The confirmation end point is not valid.	400
ContactUpdate	ContactAlreadyExists	The contact already exists.	409
ContactUpdate	ContactAlternateEmailInvalid	The contact's alternate email is not valid.	400
ContactUpdate	ContactDoesNotExist	The contact was not found.	404
ContactUpdate	ContactGivenNameInvalid	The contact's given name is not valid.	400

API	Error ID	Reason	Status
ContactUpdate	ContactMobilephoneNumberInvalid	The contact's mobile phone number is not valid.	400
ContactUpdate	ContactNotActive	The contact is not active.	404
ContactUpdate	ContactPrimaryEmailInvalid	The contact's primary email is not valid.	400
ContactUpdate	ContactSurnameInvalid	The contact surname is not valid.	400
ContactUpdate	ContactTelephoneNumberInvalid	The contact's telephone number is not valid.	400
ContactUpdate	LocalityNotValid	The locality is not valid.	400
ContactUpdate	PostalAddressNotValid	The postal address is not valid.	400
ContactUpdate	PostalCodeNotValid	The postal code is not valid.	400
ContactUpdate	ResourceStatusTransitionRequestedNotAllowed	The requested status transition is not allowed for the resource.	403
ContactUpdate	StateOrProvinceNotValid	The state or province is not valid.	400
CreateAttestation	ResourceStatusTransitionRequestedNotAllowed	The requested status transition is not allowed for the resource.	403
DeleteDeviceAuthTokenDeviceString	DeviceAuthHandleIDNotValid	The device authorization token ID is not valid.	400
DigitalAssetCreate	ApidNotMatchingWiththeXMLApid	The physical asset (APID) does not match.	403
DigitalAssetCreate	BitrateMaxValueIsRequired	The maximum value for the bitrate is required.	400
DigitalAssetCreate	CodecTypeIsRequired	The codec type is required.	400
DigitalAssetCreate	InvalidLanguage	The language is not valid.	400
DigitalAssetCreate	MdDigitalMetadataAlreadyExist	The digital metadata already exists.	409
DigitalAssetCreate	MdDigitalRecordDoesNotExist	The digital metadata was not found.	404
DigitalAssetCreate	MdNodeIdDiffrentFromCreateRequest	The node did not create the resource.	400
DigitalAssetCreate	ResourceStatusElementNotAllowed	The resource status element is not allowed.	403
DigitalAssetCreate	UpdateNumIsInvalid	The version number is not valid.	400
DigitalAssetCreate	UpdateNumIsRequired	The version number is required.	400
DigitalAssetDelete	ApidRefenceToAssetMapLpIsActive	The physical asset (APID) is referred to by an active logical asset (ALID).	409
DigitalAssetDelete	MdDigitalRecordDoesNotExist	The digital metadata was not found.	404
DigitalAssetDelete	MdNodeIdDiffrentFromCreateRequest	The node did not create the resource.	400
DigitalAssetDelete	ResourceStatusTransitionRequestedNotAllowed	The requested status transition is not allowed for the resource.	403
DigitalAssetGet	MdDigitalRecordDoesNotExist	The digital metadata was not found.	404
DigitalAssetResourceStatusUpdate	BadRequest	The request is not valid.	400
DigitalAssetResourceStatusUpdate	MdDigitalRecordDoesNotExist	The digital metadata was not found.	404
DigitalAssetResourceStatusUpdate	ResourceAlreadyinSameStatus	The resource is already in the requested status.	409

API	Error ID	Reason	Status
DigitalAssetResourceStatusUpdate	ResourceStatusTransitionRequestedNotAllowed	The requested status transition is not allowed for the resource.	403
DigitalAssetUpdate	CodecTypeIsRequired	The codec type is required.	400
DigitalAssetUpdate	InvalidLanguage	The language is not valid.	400
DigitalAssetUpdate	UpdateNumIsInvalid	The version number is not valid.	400
DigitalAssetUpdate	UpdateNumIsRequired	The version number is required.	400
DiscreteMediaRightConsume	DiscreteMediaFulfillmentMethodDoesNotExist	The discrete media fulfillment method was not found.	404
DiscreteMediaRightConsume	DiscreteMediaFulfillmentMethodNotValid	The discrete media fulfillment method is not valid.	400
DiscreteMediaRightConsume	DiscreteMediaFulfillmentMethodNotValidForRightsToken	The discrete media fulfillment method is not valid for the rights token.	409
DiscreteMediaRightConsume	DiscreteMediaRightExpireLimitReached	The discrete media right has expired.	403
DiscreteMediaRightConsume	DiscreteMediaRightRemainingCountRestriction	Insufficient discrete media rights remain.	409
DiscreteMediaRightConsume	MediaProfileNotValid	The media profile is not valid.	400
DiscreteMediaRightConsume	MediaProfileNotValidForRightsToken	The media profile is not valid for the rights token.	409
DiscreteMediaRightConsume	PurchaseProfileNotFound	The purchase profile was not found.	404
DiscreteMediaRightConsume	RightsTokenNotActive	The rights token is not active.	403
DiscreteMediaRightConsume	RightsTokenNotFound	The rights token was not found.	404
DiscreteMediaRightCreate	AuthorizedFulfillmentMethodNotValid	The authorized fulfillment method is not valid.	400
DiscreteMediaRightCreate	DiscreteMediaLimitExceeded	The maximum number of discrete media rights allowed has been exceeded.	400
DiscreteMediaRightCreate	DuplicateAuthorizedFulfillmentMethodsNotAllowed	The authorized fulfillment methods are not allowed.	400
DiscreteMediaRightCreate	MediaProfileNotValid	The media profile is not valid.	400
DiscreteMediaRightCreate	MediaProfileNotValidForRightsToken	The media profile is not valid for the rights token.	409
DiscreteMediaRightCreate	PurchaseProfileNotFound	The purchase profile was not found.	404
DiscreteMediaRightCreate	ResourceStatusElementNotAllowed	The resource status element is not allowed.	403
DiscreteMediaRightCreate	RightsTokenNotActive	The rights token is not active.	403
DiscreteMediaRightCreate	RightsTokenNotFound	The rights token was not found.	404
DiscreteMediaRightCreate	UserIdUnmatched	The user ID does not match.	403
DiscreteMediaRightDelete	DiscreteMediaRightAlreadyConsumedOrLeased	The discrete media right has been consumed or leased.	400
DiscreteMediaRightDelete	DiscreteMediaRightIDNotValid	The discrete media right ID is not valid.	400

API	Error ID	Reason	Status
DiscreteMediaRightDelete	DiscreteMediaRightNotFound	The discrete media right ID was not found.	404
DiscreteMediaRightDelete	DiscreteMediaRightOwnerMismatch	The discrete media right's owner does not match.	403
DiscreteMediaRightDelete	RightsTokenNotActive	The rights token is not active.	403
DiscreteMediaRightDelete	RightsTokenNotFound	The rights token was not found.	404
DiscreteMediaRightGet	DiscreteMediaRightIDNotValid	The discrete media right ID is not valid.	400
DiscreteMediaRightGet	DiscreteMediaRightNotActive	The discrete media right is not active.	403
DiscreteMediaRightGet	DiscreteMediaRightNotFound	The discrete media right ID was not found.	404
DiscreteMediaRightGet	RightsTokenNotActive	The rights token is not active.	403
DiscreteMediaRightGet	RightsTokenNotFound	The rights token was not found.	404
DiscreteMediaRightListGet	RightsTokenNotActive	The rights token is not active.	403
DiscreteMediaRightListGet	RightsTokenNotFound	The rights token was not found.	404
DiscreteMediaRightUpdate	AuthorizedFulfillmentMethodNotValid	The authorized fulfillment method is not valid.	400
DiscreteMediaRightUpdate	DiscreteMediaRightAlreadyConsumedOrLeased	The discrete media right has been consumed or leased.	400
DiscreteMediaRightUpdate	DiscreteMediaRightIDNotValid	The discrete media right ID is not valid.	400
DiscreteMediaRightUpdate	DiscreteMediaRightNotFound	The discrete media right ID was not found.	404
DiscreteMediaRightUpdate	DiscreteMediaRightOwnerMismatch	The discrete media right's owner does not match.	403
DiscreteMediaRightUpdate	DiscreteMediaStateNotValid	The status of the discrete media right is not valid.	400
DiscreteMediaRightUpdate	DiscreteMediaStateShouldBeAvailable	The discrete media right is not available.	400
DiscreteMediaRightUpdate	DuplicateAuthorizedFulfillmentMethodsNotAllowed	The authorized fulfillment methods are not allowed.	400
DiscreteMediaRightUpdate	MediaProfileNotValid	The media profile is not valid.	400
DiscreteMediaRightUpdate	MediaProfileNotValidForRightsToken	The media profile is not valid for the rights token.	409
DiscreteMediaRightUpdate	PurchaseProfileNotFound	The purchase profile was not found.	404
DiscreteMediaRightUpdate	RightsTokenNotActive	The rights token is not active.	403
DiscreteMediaRightUpdate	RightsTokenNotFound	The rights token was not found.	404
DiscreteMediaRightUpdate	UserIdUnmatched	The user ID does not match.	403
LegacyDeviceCreate	AccountDeviceCountExceedMaxLimit	The maximum number of devices allowed has been reached.	400
LegacyDeviceCreate	AccountIDNotValid	The account ID is not valid.	400
LegacyDeviceCreate	DeviceAlreadyExist	The device already exists.	409
LegacyDeviceCreate	DeviceCountExceedMaxLimit	The maximum number of devices allowed has been exceeded.	401
LegacyDeviceCreate	DeviceIdNotMatchingWiththeXMLDeviceID	The device ID does not match.	403

API	Error ID	Reason	Status
LegacyDeviceCreate	DeviceNodeIdDiffrentFromCreateRequest	The node ID identifies a node that is different from the node that created the device.	403
LegacyDeviceCreate	DeviceRecordDoesNotExist	The device was not found.	404
LegacyDeviceCreate	InvalidDeviceId	The device ID is not valid.	404
LegacyDeviceCreate	InvalidLegacyDeviceImageUrl	The URL for the legacy device's image is not valid.	400
LegacyDeviceCreate	NonLegacyDeviceNotSupported	The non-legacy device is not supported.	409
LegacyDeviceCreate	ReachedMaxRegisteredLegacyDevice	The maximum number of devices allowed has been reached.	409
LegacyDeviceCreate	ResourceStatusElementNotAllowed	The resource status element is not allowed.	403
LegacyDeviceDelete	DeviceNodeIdDiffrentFromCreateRequest	The node ID identifies a node that is different from the node that created the device.	403
LegacyDeviceDelete	DeviceRecordDoesNotExist	The device was not found.	404
LegacyDeviceDelete	InvalidDeviceId	The device ID is not valid.	404
LegacyDeviceUpdate	NonLegacyDeviceNotSupported	The non-legacy device is not supported.	409
LegacyDeviceUpdate	ResourceStatusElementNotAllowed	The resource status element is not allowed.	403
LogicalAssetDelete	LogicalAssetRightsReferenceActive	The Logical ID is referred to by an active rights token.	409
LogicalAssetDelete	LogicalAssetBundleReferenceActive	The Logical ID is referred to by an active Bundle	409
LogicalAssetDelete	AssetLogicalIDNotValid	The logical asset is not valid.	400
LogicalAssetDelete	AssetLogicalIDNotFound	The logical asset is not found	404
LogicalAssetDelete	Request:MdNodeIdDiffrentFromCreateRequest	The node did not create the resource.	400
LogicalAssetDelete	MediaProfileNotValid	The media profile is not valid.	400
MDBasicCreate	AccountCountryCodeNotValid	The country code is not valid.	400
MDBasicCreate	ArtReferenceImageUrlCannotBeNull	A URL for the art reference is required.	400
MDBasicCreate	ArtReferenceRequired	An art reference is required	400
MDBasicCreate	ContentIdNotMatchingWiththeXMLContentId	The content ID does not match.	403
MDBasicCreate	DuplicateContentRating	The content rating is a duplicate.	400
MDBasicCreate	DuplicateLanguageForDisplayName	The language of the display name is a duplicate.	400
MDBasicCreate	DuplicateLanguageForLocalizedInfo	The language of the localized information is a duplicate.	400
MDBasicCreate	DuplicateLanguageForSortName	The language of the sort name is a duplicate.	400
MDBasicCreate	DuplicateParent	The content parent ID is a duplicate.	400
MDBasicCreate	InvalidArtReferenceImageFormat	The format of the image is not valid.	400
MDBasicCreate	InvalidArtReferenceImageUrl	The image's URL is not valid.	400
MDBasicCreate	InvalidContentParentID	The content parent ID is not valid.	400
MDBasicCreate	InvalidDisplayIndicator	The display indicator is not valid.	400

API	Error ID	Reason	Status
MDBasicCreate	InvalidGenre	One or more of the genres is not valid.	400
MDBasicCreate	InvalidKeyword	One or more of the keywords is not valid.	400
MDBasicCreate	InvalidLanguage	The language is not valid.	400
MDBasicCreate	InvalidParentID	The parent ID is not valid.	400
MDBasicCreate	InvalidPeopleLocalNameIdentifier	The people local namespace/identifier combination is not valid.	400
MDBasicCreate	InvalidPeopleNameIdentifier	The people namespace/identifier combination is not valid.	400
MDBasicCreate	InvalidReleaseHistory	The release history is a duplicate.	400
MDBasicCreate	InvalidResolution	The resolution is not valid.	400
MDBasicCreate	InvalidResolutionWidthHeight	The resolution width and height is not valid.	400
MDBasicCreate	InvalidURIResolution	The URI is not valid.	400
MDBasicCreate	InvalidWorkType	The work type is not valid.	400
MDBasicCreate	MdBasicMetadataAlreadyExist	The basic metadata already exists.	409
MDBasicCreate	MdNodeIdDiffrentFromCreateRequest	The node did not create the resource.	400
MDBasicCreate	MultipleDefaultLanguageForLocalizedInfo	Only one default language is allowed for localized info.	400
MDBasicCreate	ReleaseHistoryDateCannotBeNull	The release history date is required.	400
MDBasicCreate	ReleaseYearCannotBeNull	The release year is required.	400
MDBasicCreate	ResolutionCannotBeNull	The resolution is required.	400
MDBasicCreate	SequenceInfoAndParentInfoRequired	The sequence information and parent information elements are required.	400
MDBasicCreate	UpdateNumIsInvalid	The version number is not valid.	400
MDBasicCreate	UpdateNumIsRequired	The version number is required.	400
MDBasicDelete	MdBasicAssetMapReferenceActive	The content ID is referred to by an active asset map.	409
MDBasicDelete	MdBasicBundleReferenceActive	The content ID is referred to by an active bundle.	409
MDBasicDelete	MdBasicDigitalReferenceActive	The content ID is referred to by an active digital asset.	409
MDBasicDelete	MdBasicRightsTokenReferenceActive	The content ID is referred to by an active rights token.	409
MDBasicDelete	MdNodeIdDiffrentFromCreateRequest	The node did not create the resource.	400
MDBasicDelete	ResourceStatusTransitionRequestedNotAllowed	The requested status transition is not allowed for the resource.	403
MDBasicGet	PostProcessingFailed	Post-processing of the image failed.	409

API	Error ID	Reason	Status
MDBasicGet	PostProcessingNotCompleted	Post-processing of the image was not completed.	404
MDBasicResourceStatusUpdate	BadRequest	The request is not valid.	400
MDBasicResourceStatusUpdate	ResourceAlreadyinSameStatus	The resource is already in the requested status.	409
MDBasicResourceStatusUpdate	ResourceStatusTransitionRequestedNotAllowed	The requested status transition is not allowed for the resource.	403
MDBasicUpdate	AccountCountryCodeNotValid	The country code is not valid.	400
MDBasicUpdate	ArtReferenceImageUrlCannotBeNull	A URL for the art reference is required.	400
MDBasicUpdate	ArtReferenceRequired	An art reference is required	400
MDBasicUpdate	DuplicateContentRating	The content rating is a duplicate.	400
MDBasicUpdate	DuplicateLanguageForDisplayName	The language of the display name is a duplicate.	400
MDBasicUpdate	DuplicateLanguageForLocalizedInfo	The language of the localized information is a duplicate.	400
MDBasicUpdate	DuplicateLanguageForSortName	The language of the sort name is a duplicate.	400
MDBasicUpdate	DuplicateParent	The content parent ID is a duplicate.	400
MDBasicUpdate	InvalidArtReferenceImageFormat	The format of the image is not valid.	400
MDBasicUpdate	InvalidArtReferenceImageUrl	The image's URL is not valid.	400
MDBasicUpdate	InvalidContentParentID	The content parent ID is not valid.	400
MDBasicUpdate	InvalidContentRating	The content rating is not valid.	400
MDBasicUpdate	InvalidGenre	One or more of the genres is not valid.	400
MDBasicUpdate	InvalidKeyword	One or more of the keywords is not valid.	400
MDBasicUpdate	InvalidLanguage	The language is not valid.	400
MDBasicUpdate	InvalidParentID	The parent ID is not valid.	400
MDBasicUpdate	InvalidPeopleLocalNameIdentifier	The people local namespace/identifier combination is not valid.	400
MDBasicUpdate	InvalidPeopleNameIdentifier	The people namespace/identifier combination is not valid.	400
MDBasicUpdate	InvalidReleaseHistory	The release history is a duplicate.	400
MDBasicUpdate	InvalidResolution	The resolution is not valid.	400
MDBasicUpdate	InvalidResolutionWidthHeight	The resolution width and height is not valid.	400
MDBasicUpdate	InvalidURIResolution	The URI is not valid.	400
MDBasicUpdate	InvalidWorkType	The work type is not valid.	400
MDBasicUpdate	MdBasicMetadataAlreadyExist	The basic metadata already exists.	409
MDBasicUpdate	MultipleDefaultLanguageForLocalizedInfo	Only one default language is allowed for localized info.	400
MDBasicUpdate	ReleaseHistoryDateCannotBeNull	The release history date is required.	400
MDBasicUpdate	ReleaseYearCannotBeNull	The release year is required.	400
MDBasicUpdate	ResolutionCannotBeNull	The resolution is required.	400

API	Error ID	Reason	Status
MDBasicUpdate	SequenceInfoAndParentInfoRequired	The sequence information and parent information elements are required.	400
MDBasicUpdate	UpdateNumIsInvalid	The version number is not valid.	400
MDBasicUpdate	UpdateNumIsRequired	The version number is required.	400
MDBundleCreate	AssetLogicalIDNotFound	The logical asset (ALID) was not found.	404
MDBundleCreate	BundleAlreadyExist	The bundle already exists.	409
MDBundleCreate	BundleIDNotFound	The bundle ID was not found.	404
MDBundleCreate	BundleIdNotMatchingWiththeXMLBundleId	The bundle ID does not match.	403
MDBundleCreate	DuplicateContentId	The content ID is a duplicate.	400
MDBundleCreate	InvalidArtReferenceImageFormat	The format of the image is not valid.	400
MDBundleCreate	InvalidArtReferenceImageUrl	The image's URL is not valid.	400
MDBundleCreate	InvalidContentRating	The content rating is not valid.	400
MDBundleCreate	InvalidDisplayIndicator	The display indicator is not valid.	400
MDBundleCreate	InvalidLanguage	The language is not valid.	400
MDBundleCreate	InvalidPeopleLocalNameIdentifier	The people local namespace/identifier combination is not valid.	400
MDBundleCreate	InvalidReleaseHistory	The release history is a duplicate.	400
MDBundleCreate	InvalidResolution	The resolution is not valid.	400
MDBundleCreate	InvalidURIResolution	The URI is not valid.	400
MDBundleCreate	InvalidWorkType	The work type is not valid.	400
MDBundleCreate	MdNodeIdDiffrentFromCreateRequest	The node did not create the resource.	400
MDBundleCreate	MultipleDefaultLanguageForLocalizedInfo	Only one default language is allowed for localized info.	400
MDBundleDelete	BundleIDNotFound	The bundle ID was not found.	404
MDBundleDelete	BundleLinkedWithRightsTokenCannotBeDeleted	The bundle cannot be removed.	409
MDBundleDelete	ResourceStatusTransitionRequestedNotAllowed	The requested status transition is not allowed for the resource.	403
MDBundleGet	BundleIDNotFound	The bundle ID was not found.	404
MDBundleGet	PostProcessingFailed	Post-processing of the image failed.	409
MDBundleGet	PostProcessingNotCompleted	Post-processing of the image was not completed.	404
MdBundleResourceStatusUpdate	BadRequest	The request is not valid.	400
MdBundleResourceStatusUpdate	BundleIDNotFound	The bundle ID was not found.	404
MdBundleResourceStatusUpdate	ResourceAlreadyinSameStatus	The resource is already in the requested status.	409

API	Error ID	Reason	Status
MdBundleResourceStatusUpdate	ResourceStatusTransitionRequestedNotAllowed	The requested status transition is not allowed for the resource.	403
MDBundleUpdate	AssetLogicalIDNotFound	The logical asset (ALID) was not found.	404
MDBundleUpdate	BundleIDNotFound	The bundle ID was not found.	404
MDBundleUpdate	DuplicateContentId	The content ID is a duplicate.	400
MDBundleUpdate	InvalidLanguage	The language is not valid.	400
NodeCreate	AddressDoesNotExist	The address was not found.	404
NodeCreate	ContactDoesNotExist	The contact was not found.	404
NodeCreate	DeceProtocolVersionNotProper	The DECE protocol version is not valid.	400
NodeCreate	DisplayNameRequired	The display name is required.	400
NodeCreate	InvalidLogoResourceUrl	The URL for the logo is not valid.	400
NodeCreate	InvalidMediaDownloadLocBase	The base media download location is invalid.	400
NodeCreate	LocalityNotValid	The locality is not valid.	400
NodeCreate	NodeAlreadyExists	The node already exists.	409
NodeCreate	NodeDeviceManagementURLNotValid	The device management URL is not valid.	400
NodeCreate	NodeProxyOrgIdDoesNotExist	The node's proxy organization does not exist.	404
NodeCreate	NodeRoleInvalid	The node/role is not valid.	401
NodeCreate	OrgIdInvalid	The organization ID is not valid.	400
NodeCreate	OrgIdRequired	An organization ID is required.	400
NodeCreate	OrgIdUnmatched	The organization ID does not match.	400
NodeCreate	OrgNotActive	The organization is not active.	404
NodeCreate	OrgNotFound	The organization was not found.	404
NodeCreate	PostalAddressNotValid	The postal address is not valid.	400
NodeCreate	PostalCodeNotValid	The postal code is not valid.	400
NodeCreate	ResourceStatusTransitionRequestedNotAllowed	The requested status transition is not allowed for the resource.	403
NodeCreate	StateOrProvinceNotValid	The state or province is not valid.	400
NodeCreate	StatusInvalid	The status is not valid.	400
NodeDelete	NodeIDRequired	The node ID is required.	400
NodeDelete	OrgIdInvalid	The organization ID is not valid.	400
NodeDelete	OrgIdRequired	An organization ID is required.	400
NodeGet	NodeIDRequired	The node ID is required.	400
NodeGet	OrgIdInvalid	The organization ID is not valid.	400
NodeGet	OrgNotFound	The organization was not found.	404
NodeGet	OrgNotFound	The organization was not found.	404
NodeList	OrgIdInvalid	The organization ID is not valid.	400
NodeResourceStatusUpdate	AccountStatusNotValid	The account status is not valid.	400

API	Error ID	Reason	Status
NodeResourceStatusUpdate	NodeUnauthorizedToActOnAccount	The request is not authorized.	401
NodeResourceStatusUpdate	OrgIdUnmatched	The organization ID does not match.	400
NodeResourceStatusUpdate	OrgNotFound	The organization was not found.	404
NodeResourceStatusUpdate	RequestorPrivilegeInsufficient	You do not have permission to perform this action. Ask a full access member of your account for help.	403
NodeResourceStatusUpdate	ResourceAlreadyInRequestedStatus	The resource is already in the requested status.	400
NodeResourceStatusUpdate	ResourceStatusTransitionRequestedNotAllowed	The requested status transition is not allowed for the resource.	403
NodeResourceStatusUpdate	StatusInvalid	The status is not valid.	400
NodeUpdate	AddressDoesNotExist	The address was not found.	404
NodeUpdate	ContactDoesNotExist	The contact was not found.	404
NodeUpdate	DeceProtocolVersionNotProper	The DECE protocol version is not valid.	400
NodeUpdate	InvalidLogoResourceUrl	The URL for the logo is not valid.	400
NodeUpdate	InvalidMediaDownloadLocBase	The base media download location is invalid.	400
NodeUpdate	LocalityNotValid	The locality is not valid.	400
NodeUpdate	NodeAlreadyExists	The node already exists.	409
NodeUpdate	NodeDeviceManagementURLNotValid	The device management URL is not valid.	400
NodeUpdate	NodeDoesNotBelongsToOrg	The node does not belong to the organization.	400
NodeUpdate	NodeIDRequired	The node ID is required.	400
NodeUpdate	NodeProxyOrgIdDoesNotExist	The node's proxy organization does not exist.	404
NodeUpdate	NodeRoleInvalid	The node/role is not valid.	401
NodeUpdate	OrgIdInvalid	The organization ID is not valid.	400
NodeUpdate	OrgIdRequired	An organization ID is required.	400
NodeUpdate	OrgIdUnmatched	The organization ID does not match.	400
NodeUpdate	PostalAddressNotValid	The postal address is not valid.	400
NodeUpdate	PostalCodeNotValid	The postal code is not valid.	400
NodeUpdate	ResourceStatusTransitionRequestedNotAllowed	The requested status transition is not allowed for the resource.	403
NodeUpdate	StateOrProvinceNotValid	The state or province is not valid.	400
NodeUpdate	StatusInvalid	The status is not valid.	400
OrgCreate	AddressDoesNotExist	The address was not found.	404
OrgCreate	AppAuthTokenDataOrValueInvalid	The authorization token contains invalid information.	400
OrgCreate	AppAuthTokenIdInvalid	The authorization token is not valid.	400
OrgCreate	ContactDoesNotExist	The contact was not found.	404
OrgCreate	ContactPrimaryEmailInvalid	The contact's primary email is not valid.	400
OrgCreate	ContactSurnameInvalid	The contact surname is not valid.	400
OrgCreate	ContactTelephoneNumberInvalid	The contact's telephone number is not valid.	400

API	Error ID	Reason	Status
OrgCreate	DisplayNameLanguageNotValid	The language of the display name is not valid.	400
OrgCreate	FieldExceedsMaxLength	The number of characters in the field exceeds the maximum number allowed.	400
OrgCreate	OrgAlreadyExists	The organization already exists.	409
OrgCreate	OrganizationSortNameInvalid	The organization's sort name is not valid.	400
OrgCreate	OrganizationWebsiteInvalid	The organization's web site is not valid.	400
OrgCreate	OrgIdInvalid	The organization ID is not valid.	400
OrgCreate	OrgNotActive	The organization is not active.	404
OrgCreate	ResourceStatusTransitionRequestedNotAllowed	The requested status transition is not allowed for the resource.	403
OrgDelete	OrgHasActiveNodes	The organization has associated active nodes.	401
OrgDelete	OrgIdInvalid	The organization ID is not valid.	400
OrgDelete	OrgIdRequired	An organization ID is required.	400
OrganizationGet	OrgIdRequired	An organization ID is required.	400
OrganizationGet	OrgNotFound	The organization was not found.	404
OrgResourceStatusUpdate	OrgHasActiveNodes	The organization has associated active nodes.	401
OrgResourceStatusUpdate	ResourceAlreadyinSameStatus	The resource is already in the requested status.	409
OrgResoureStatusUpdate	OrgNotFound	The organization was not found.	404
OrgUpdate	AddressDoesNotExist	The address was not found.	404
OrgUpdate	AppAuthTokenDataOrValueInvalid	The authorization token contains invalid information.	400
OrgUpdate	AppAuthTokenIdInvalid	The authorization token is not valid.	400
OrgUpdate	ContactDoesNotExist	The contact was not found.	404
OrgUpdate	ContactPrimaryEmailInvalid	The contact's primary email is not valid.	400
OrgUpdate	ContactSurnameInvalid	The contact surname is not valid.	400
OrgUpdate	ContactTelephoneNumberInvalid	The contact's telephone number is not valid.	400
OrgUpdate	DisplayNameLanguageNotValid	The language of the display name is not valid.	400
OrgUpdate	FieldExceedsMaxLength	The number of characters in the field exceeds the maximum number allowed.	400
OrgUpdate	OrgAlreadyExists	The organization already exists.	409
OrgUpdate	OrganizationSortNameInvalid	The organization's sort name is not valid.	400
OrgUpdate	OrganizationWebsiteInvalid	The organization's web site is not valid.	400
OrgUpdate	OrgIdInvalid	The organization ID is not valid.	400
OrgUpdate	OrgIdRequired	An organization ID is required.	400
OrgUpdate	OrgNotActive	The organization is not active.	404
OrgUpdate	OrgNotFound	The organization was not found.	404
OrgUpdate	ResourceStatusTransitionRequestedNotAllowed	The requested status transition is not allowed for the resource.	403

API	Error ID	Reason	Status
PolicyCreate	AccountIDNotValid	The account ID is not valid.	400
PolicyCreate	AccountStatusNotValid	The account status is not valid.	400
PolicyCreate	CLGNotAttested	The underage member does not have a connected legal guardian (CLG).	403
PolicyCreate	DuplicatePolicyCannotBeAdded	The requested policy already exists.	403
PolicyCreate	EnableManageUserConsentRequired	The setting of the EnableManageUserConsent policy prevents the requested action from being completed.	403
PolicyCreate	IncomingPoliciesOrExistingPoliciesAreInvalid	The requested policies or those already applied are not valid.	400
PolicyCreate	LatestTOUNotAccepted	The latest version of the Terms of Use has not been accepted.	403
PolicyCreate	PolicyActorInvalid	The policy actor is not valid.	400
PolicyCreate	PolicyClassInvalid	The policy class is not valid.	400
PolicyCreate	PolicyClassNotValid	The policy class is not valid	400
PolicyCreate	PolicyCreatorInvalid	The policy creator is not valid.	400
PolicyCreate	PolicyCreatorNotFound	The policy creator was not found.	404
PolicyCreate	PolicyIdNotValid	The policy ID is not valid.	400
PolicyCreate	PolicyListInvalid	The policy list is not valid.	400
PolicyCreate	PolicyRequestingEntityInvalid	The policy requesting entity is not valid.	400
PolicyCreate	PolicyRequestingEntityInvalidForPolicyClass	The policy requesting entity is not valid for the policy class.	400
PolicyCreate	PolicyRequestingEntityNotFound	The policy requesting entity is not valid.	404
PolicyCreate	PolicyResourceInvalid	The policy resource is not valid.	400
PolicyCreate	PolicyResourceInvalidForPolicyClass	The policy resource is not valid for the policy class.	400
PolicyCreate	PolicyResourceNotFound	The policy resource was not found.	404
PolicyCreate	PolicyResourceStatusRequired	A policy resource status is required.	400
PolicyCreate	PolicyStatusNotValid	The policy's status is not valid.	400
PolicyCreate	TOUNotAccepted	The Terms of Use policy was not accepted.	403
PolicyCreate	UserStatusNotValid	The member's status is not valid.	400
PolicyDelete	AccountIDNotValid	The account ID is not valid.	400
PolicyDelete	EnableManageUserConsentCannotBeDeleted	The EnableManageUserConsent policy cannot be removed.	400
PolicyDelete	EnableManageUserConsentRequired	The setting of the EnableManageUserConsent policy prevents the requested action from being completed.	403
PolicyDelete	EnableUserDataUsageConsentCannotBeDeleted	The EnableUserDataUsageConsent policy cannot be removed.	400

API	Error ID	Reason	Status
PolicyDelete	PolicyIdNotValid	The policy ID is not valid.	400
PolicyDelete	PolicyInfoInURLNotValid	The policy information in the URL is not valid.	400
PolicyDelete	PolicyNotFound	The policy was not found.	404
PolicyDelete	TOUCannotBeDeleted	The Terms of Use policy cannot be removed.	403
PolicyDelete	TOUNotAccepted	The Terms of Use policy was not accepted.	403
PolicyDelete	UserAccessToPolicyNotAuthorized	The member does not have permission to access the policy.	403
PolicyGet	AccountIDNotValid	The account ID is not valid.	400
PolicyGet	AccountStatusNotValid	The account status is not valid.	400
PolicyGet	NodeUserIdFailure	The node/member does not exist for the node.	500
PolicyGet	PolicyClassNotValid	The policy class is not valid	400
PolicyGet	PolicyIdNotValid	The policy ID is not valid.	400
PolicyGet	PolicyListIdNotValid	The policy list ID is not valid.	400
PolicyGet	PolicyNotFound	The policy was not found.	404
PolicyGet	UserStatusNotValid	The member's status is not valid.	400
PolicyUpdate	AccountIDNotValid	The account ID is not valid.	400
PolicyUpdate	AccountStatusNotValid	The account status is not valid.	400
PolicyUpdate	DuplicatePolicyCannotBeAdded	The requested policy already exists.	403
PolicyUpdate	EnableManageUserConsentRequired	The setting of the EnableManageUserConsent policy prevents the requested action from being completed.	403
PolicyUpdate	EnableUserDataUsageConsentCannotBeDeleted	The EnableUserDataUsageConsent policy cannot be removed.	400
PolicyUpdate	IncomingPoliciesOrExistingPoliciesAreInvalid	The requested policies or those already applied are not valid.	400
PolicyUpdate	LockerViewAllConsentCannotBeDeleted	The LockerViewAllConsent policy cannot be removed.	403
PolicyUpdate	PolicyActorInvalid	The policy actor is not valid.	400
PolicyUpdate	PolicyClassInvalid	The policy class is not valid.	400
PolicyUpdate	PolicyClassNotValid	The policy class is not valid	400
PolicyUpdate	PolicyCreatorInvalid	The policy creator is not valid.	400
PolicyUpdate	PolicyCreatorNotFound	The policy creator was not found.	404
PolicyUpdate	PolicyIdNotValid	The policy ID is not valid.	400
PolicyUpdate	PolicyInfoInURLNotValid	The policy information in the URL is not valid.	400
PolicyUpdate	PolicyListInvalid	The policy list is not valid.	400
PolicyUpdate	PolicyNotFound	The policy was not found.	404
PolicyUpdate	PolicyRequestingEntityInvalid	The policy requesting entity is not valid.	400

API	Error ID	Reason	Status
PolicyUpdate	PolicyRequestingEntityInvalidForPolicyClass	The policy requesting entity is not valid for the policy class.	400
PolicyUpdate	PolicyRequestingEntityNotFound	The policy requesting entity is not valid.	404
PolicyUpdate	PolicyResourceInvalid	The policy resource is not valid.	400
PolicyUpdate	PolicyResourceInvalidForPolicyClass	The policy resource is not valid for the policy class.	400
PolicyUpdate	PolicyResourceNotFound	The policy resource was not found.	404
PolicyUpdate	PolicyResourceStatusRequired	A policy resource status is required.	400
PolicyUpdate	PolicyStatusNotValid	The policy's status is not valid.	400
PolicyUpdate	PolicyUpdatorInvalid	The requesting member of the policy update is not valid.	400
PolicyUpdate	PolicyUpdatorNotFound	The requestor of the policy update was not found.	404
PolicyUpdate	ResourceStatusRequired	The resource status is required.	400
PolicyUpdate	TOUAcceptanceNotAllowedViaPolicyUpdate	Terms of Use acceptance cannot be performed using this method.	405
PolicyUpdate	TOUNotAccepted	The Terms of Use policy was not accepted.	403
PolicyUpdate	UserStatusNotValid	The member's status is not valid.	400
ResourcePropertyQuery	PrimaryEmailAddressMinLengthNotMet	The primary email address is too short.	400
ResourcePropertyQuery	UserNameMinimumLengthNotMet	The sign-in name is too short.	400
ResourcePropertyQuery	XPathExpressionisInvalid	The XPath expression is not valid.	400
RightsLockerDataGet	FilterClassNotValid	The filter class is not valid.	400
RightsLockerDataGet	FilterCountNotValid	The filter count is not valid.	400
RightsLockerDataGet	FilterOffsetNotValid	The filter offset is not valid.	400
RightsLockerDataGet	ResponseQueryParameterNotValid	The response query parameter is not valid (must be token, reference, download, or metadata).	400
RightsTokenCreate	AlidCidMappingNotFound	The mapping between the logical asset (ALID) and the content ID was not found.	404
RightsTokenCreate	ALIDInBundleNotFound	The logical asset (ALID) was not found in the bundle.	404
RightsTokenCreate	AssetLogicalIDNotActive	The logical asset (ALID) is not active.	403
RightsTokenCreate	AssetLogicalIDNotFound	The logical asset (ALID) was not found.	404
RightsTokenCreate	BundleIDNotActive	The bundle is not active.	403
RightsTokenCreate	BundleIDNotFound	The bundle ID was not found.	404
RightsTokenCreate	DiscreteMediaRightsRemainingNotAllowed	The number of discrete rights remaining cannot be set during rights token creation.	400
RightsTokenCreate	DisplayNameLanguageNotValid	The language of the display name is not valid.	400
RightsTokenCreate	DisplayNameNotValid	The display name is not valid.	400
RightsTokenCreate	FulfillmentLocNotValid	The fulfillment location is not valid.	400

API	Error ID	Reason	Status
RightsTokenCreate	FulfillmentWebLocMediaProfileRequired	The fulfillment location is required.	400
RightsTokenCreate	HDContentProfileForLogicalAssetNotAllowed	The HD content profile is not allowed for the logical asset (ALID).	403
RightsTokenCreate	MediaProfileNotValid	The media profile is not valid.	400
RightsTokenCreate	MediaProfileRequired	A media profile is required.	400
RightsTokenCreate	PurchaseAccountNotValid	The purchase account ID is not valid.	400
RightsTokenCreate	PurchaseNodeIDNotValid	The purchase node ID is not valid.	400
RightsTokenCreate	PurchaseUserNotValid	The purchasing member's user ID is not valid.	400
RightsTokenCreate	RightsLockerNotFound	The rights locker was not found.	404
RightsTokenCreate	SDContentProfileForLogicalAssetNotAllowed	The standard-definition content profile is not allowed for the logical asset (ALID).	403
RightsTokenCreate	StandardDefinitionMissing	The standard-definition media profile is missing.	400
RightsTokenCreate	TransactionTypeIDNotValid	The transaction type is not valid.	400
RightsTokenCreate	UHDContentProfileForLogicalAssetNotAllowed	The UHD content profile is not allowed for the logical asset (ALID).	403
RightsTokenListCreate	RightsTokenListWithEmptyData	RightsTokenListCreate does not have RightsToken	400
RightsTokenListCreate	ExceededTheMaximumNoOfRightsToken	RightsTokenListCreate exceeds the maximum number of RightsToken	400
RightsTokenListCreate	AccountIdUnmatched	The account ID does not match.	403
RightsTokenListCreate	RightsLockerIDInRequestDoNotMatchAccountRightsLockerID	The rights locker ID does not match.	400
RightsTokenListCreate	ProfileParametersNotAllowedForMediaProfileSD	Profile parameters are not allowed for media profile SD	400
RightsTokenListCreate	AssetLogicalIDNotValid	The logical asset (ALID) is not valid.	400
RightsTokenListCreate	ContentIDNotValid	The content is not valid.	400
RightsTokenListCreate	BundleIDNotValid	The bundle is not valid.	400
RightsTokenListCreate	DisplayNameNotValid	The display name is not valid.	400
RightsTokenListCreate	DisplayNameLanguageNotValid	Language of the display name is not valid.	403
RightsTokenListCreate	PurchaseAccountNotValid	The purchase account ID is not valid.	400
RightsTokenListCreate	PurchaseUserNotValid	The purchasing member's user ID is not valid.	400
RightsTokenListCreate	PurchaseNodeIDNotValid	The purchase node ID is not valid.	400
RightsTokenListCreate	FulfillmentLocNotValid	The fulfilment location is not valid.	400
RightsTokenListCreate	FulfillmentWebLocMediaProfileRequired	The fulfilment location is required.	400
RightsTokenListCreate	StreamWebLocRequired	StreamWebLoc is required	400

API	Error ID	Reason	Status
RightsTokenListCreate	DiscreteMediaRightsRemainingNotAllowed	The number of discrete rights remaining cannot be set during rights token creation.	400
RightsTokenListCreate	StandardDefinitionMissing	The standard-definition media profile is missing.	400
RightsTokenListCreate	AlidCidMappingNotFound	The mapping between the logical asset (ALID) and the content ID was not found.	404
RightsTokenListCreate	ALIDInBundleNotFound	The logical asset (ALID) was not found in the bundle.	404
RightsTokenListCreate	AssetLogicalIDNotActive	The logical asset (ALID) is not active.	403
RightsTokenListCreate	AssetLogicalIDNotFound	The logical asset (ALID) was not found.	404
RightsTokenListCreate	BundleIDNotActive	The bundle is not active.	403
RightsTokenListCreate	BundleIDNotFound	The bundle ID was not found.	404
RightsTokenListCreate	HDContentProfileForLogicalAssetNotAllowed	The HD content profile is not allowed for the logical asset (ALID).	403
RightsTokenListCreate	SDContentProfileForLogicalAssetNotAllowed	The standard-definition content profile is not allowed for the logical asset (ALID).	403
RightsTokenListCreate	UHDContentProfileForLogicalAssetNotAllowed	The UHD content profile is not allowed for the logical asset (ALID).	403
RightsTokenListCreate	PurchaseUserNotValid	The purchasing member's user ID is not valid.	400
RightsTokenListCreate	PurchaseNodeIDNotValid	The purchase node ID is not valid.	400
RightsTokenListCreate	FulfillmentLocNotValid	The fulfilment location is not valid.	400
RightsTokenListCreate	FulfillmentWebLocMediaProfileRequired	The fulfilment location is required.	400
RightsTokenListCreate	RightsLockerNotFound	The rights locker was not found.	404
RightsTokenListCreate	TransactionTypeIDNotValid	The transaction type is not valid.	400
RightsTokenDataGet	AssetLogicalIDNotActive	The logical asset (ALID) is not active.	403
RightsTokenDataGet	NativeDRMClientIDNotFound	The native DRM client ID was not found.	404
RightsTokenDelete	AccountIDNotValid	The account ID is not valid.	400
RightsTokenDelete	RightsTokenAlreadyDeleted	The rights token has already been removed.	403
RightsTokenDelete	RightsTokenNodeNotIssuer	The requesting node did not issue the rights token, and therefore cannot delete it.	403
RightsTokenDelete	RightsTokenNotFound	The rights token was not found.	404
RightsTokenGet	AccountDoesNotHaveRightsTokenInURL	The rights token was not found in the account.	400
RightsTokenGet	RightsTokenNotAvailable	The rights token is not available.	403
RightsTokenGet	RightsTokenNotFound	The rights token was not found.	404
RightsTokenGetAlid	AssetIdentifierNotValid	The physical asset (APID) or the logical asset (ALID) is not valid.	400
RightsTokenGetAlid	AssetLogicalIDNotActive	The logical asset (ALID) is not active.	403

API	Error ID	Reason	Status
RightsTokenGetAlid	AssetLogicalIDNotFound	The logical asset (ALID) was not found.	404
RightsTokenResourceStatusUpdate	AccountDoesNotHaveRightsTokenInURL	The rights token was not found in the account.	400
RightsTokenResourceStatusUpdate	NodeIdOrgIdUnmatched	The node does not belong to the organization.	400
RightsTokenResourceStatusUpdate	ResourceAlreadyinSameStatus	The resource is already in the requested status.	409
RightsTokenResourceStatusUpdate	ResourceStatusTransitionRequestedNotAllowed	The requested status transition is not allowed for the resource.	403
RightsTokenResourceStatusUpdate	RightsTokenNotFound	The rights token was not found.	404
RightsTokenResourceStatusUpdate	StatusInvalid	The status is not valid.	400
RightsTokenUpdate	AccountDoesNotHaveRightsTokenInURL	The rights token was not found in the account.	400
RightsTokenUpdate	AssetLogicalIDNotActive	The logical asset (ALID) is not active.	403
RightsTokenUpdate	AssetLogicalIDNotFound	The logical asset (ALID) was not found.	404
RightsTokenUpdate	DisplayNameLanguageNotValid	The language of the display name is not valid.	400
RightsTokenUpdate	FulfillmentLocNotValid	The fulfillment location is not valid.	400
RightsTokenUpdate	FulfillmentWebLocMediaProfileRequired	The fulfillment location is required.	400
RightsTokenUpdate	HDContentProfileForLogicalAssetNotAllowed	The HD content profile is not allowed for the logical asset (ALID).	403
RightsTokenUpdate	MediaProfileNotValid	The media profile is not valid.	400
RightsTokenUpdate	MediaProfileRequired	A media profile is required.	400
RightsTokenUpdate	PurchaseAccountNotFound	The purchase account was not found.	404
RightsTokenUpdate	PurchaseAccountNotValid	The purchase account ID is not valid.	400
RightsTokenUpdate	PurchaseNodeIDNotValid	The purchase node ID is not valid.	400
RightsTokenUpdate	PurchaseProfileHasDMRAlreadyCreated	The purchase profile already has a discrete media right.	400
RightsTokenUpdate	PurchaseTimeNotValid	The purchase time is not valid.	400
RightsTokenUpdate	PurchaseUserDoesNotBelongToPurchaseAccount	The purchasing member does not belong to the purchase account.	400
RightsTokenUpdate	PurchaseUserNotFound	The purchasing member was not found.	404
RightsTokenUpdate	PurchaseUserNotValid	The purchasing member's user ID is not valid.	400
RightsTokenUpdate	RightsLockerIDInRequestDoNotMatchAccountRightsLockerID	The rights locker ID does not match.	400

API	Error ID	Reason	Status
RightsTokenUpdate	RightsLockerNotFound	The rights locker was not found.	404
RightsTokenUpdate	RightsTokenNodeNotIssuer	The requesting node did not issue the rights token, and therefore cannot delete it.	403
RightsTokenUpdate	RightsTokenNotFound	The rights token was not found.	404
RightsTokenUpdate	RightsTokenNotPurchasedThroughRetailer	The rights token being updated was not purchased through the retailer.	403
RightsTokenUpdate	SDContentProfileForLogicalAssetNotAllowed	The standard-definition content profile is not allowed for the logical asset (ALID).	403
RightsTokenUpdate	StandardDefinitionMissing	The standard-definition media profile is missing.	400
RightsTokenUpdate	TransactionTypeIDNotValid	The transaction type is not valid.	400
StreamCreate	AccountStreamCountExceedMaxLimit	The maximum number of streams allowed in an account has been reached.	409
StreamCreate	CalculationMethodNotValid	The calculation method is not valid.	400
StreamCreate	ConfidenceOutOfRange	Confidence must be between 1 and 100.	400
StreamCreate	GeoLocationValueFormatNotValid	The format of the country name, postal code, or subdivision is not valid.	400
StreamCreate	RightsTokenNotActive	The rights token is not active.	403
StreamCreate	RightsTokenNotFound	The rights token was not found.	404
StreamCreate	StreamClientNicknameTooLong	The stream client nickname is too long.	400
StreamCreate	StreamRightsNotGranted	The logical asset (ALID) cannot be streamed.	403
StreamCreate	StreamTransactionIdInvalid	The stream transaction ID is not valid.	400
StreamCreate	UserIdUnmatched	The user ID does not match.	403
StreamCreate	UserNotSpecified	A user ID is required.	400
StreamCreate	UserPrivilegeAccessRestricted	The user does not have permission to access this content.	403
StreamCreate	ViaProxyNotValid	The via proxy element is not valid.	400
StreamDelete	StreamHandleIDNotValid	The stream handle ID is not valid.	400
StreamDelete	StreamHandleIDRequired	A stream handle ID is required.	400
StreamDelete	StreamNotFound	The stream was not found.	404
StreamDelete	StreamOwnerMismatch	The stream's owner does not match.	403
StreamDelete	UserNotSpecified	A user ID is required.	400
StreamDelete	UserPrivilegeAccessRestricted	The user does not have permission to access this content.	403
StreamListView	UserNotSpecified	A user ID is required.	400
StreamRenew	RightsTokenNotActive	The rights token is not active.	403
StreamRenew	RightsTokenNotFound	The rights token was not found.	404
StreamRenew	StreamHandleIDNotValid	The stream handle ID is not valid.	400

API	Error ID	Reason	Status
StreamRenew	StreamHandleIDRequired	A stream handle ID is required.	400
StreamRenew	StreamNotActive	The stream is not active.	403
StreamRenew	StreamNotFound	The stream was not found.	404
StreamRenew	StreamOwnerMismatch	The stream's owner does not match.	403
StreamRenew	StreamRenewExceedsMaximumTime	The stream-renewal request exceeds the maximum allowable time.	409
StreamRenew	StreamRightsNotGranted	The logical asset (ALID) cannot be streamed.	403
StreamRenew	UserNotSpecified	A user ID is required.	400
StreamRenew	UserPrivilegeAccessRestricted	The user does not have permission to access this content.	403
StreamView	StreamHandleIDNotValid	The stream handle ID is not valid.	400
StreamView	StreamHandleIDRequired	A stream handle ID is required.	400
StreamView	StreamNotFound	The stream was not found.	404
StreamView	StreamOwnerMismatch	The stream's owner does not match.	403
StreamView	UserNotSpecified	A user ID is required.	400
UserCreate	AccountActiveUserCountReachedMaxLimit	The maximum number of active members allowed has been reached.	400
UserCreate	AccountStatusNotValid	The account status is not valid.	400
UserCreate	AccountUserAddressNotValid	The address is not valid.	400
UserCreate	AccountUserAlternateEmailNotValid	The alternate email address is not valid.	400
UserCreate	AccountUserCountryNotValid	The country is not valid.	400
UserCreate	AccountUserEmailAddressDuplicated	The email address is a duplicate.	400
UserCreate	AccountUserGivenNameNotValid	The given name is not valid.	400
UserCreate	AccountUserLanguageDuplicated	The language is a duplicate.	400
UserCreate	AccountUserLanguageNotValid	The language is not valid.	400
UserCreate	AccountUserMobilePhoneNumberNotValid	The mobile telephone number is not valid.	400
UserCreate	AccountUsernameRegistered	The sign-in name already exists.	400
UserCreate	AccountUserPasswordNotValid	The password is not valid.	400
UserCreate	AccountUserPrimaryEmailNotValid	The primary email address is not valid.	400
UserCreate	AccountUserPrimaryLanguageNotValid	The primary language is not valid.	400
UserCreate	AccountUserSecurityAnswerNotValid	The answer to the security question is not valid.	400
UserCreate	AccountUserSecurityQuestionDuplicated	The security question is a duplicate.	400
UserCreate	AccountUserSecurityQuestionIDNotValid	The security question is not valid.	400

API	Error ID	Reason	Status
UserCreate	AccountUserSurnameNotValid	The surname is not valid.	400
UserCreate	AccountUserTelephoneNumberNotValid	The telephone number is not valid.	400
UserCreate	CountryNotValid	The country is not valid.	400
UserCreate	FirstUserMustBeCreatedWithFullAccessPrivilege	The first member must be a full-access member.	403
UserCreate	PrimaryEmailConfirmationEndpointRequired	A confirmation endpoint is required for the member primary email address.	400
UserCreate	PrimaryEmailVerifiedAttributeMustBeTrue	If the member's primary email address has been verified by the node, the setting of the PrimaryEmailVerified attribute must be set to TRUE.	400
UserCreate	RequestorNotActive	The requestor is not active.	403
UserCreate	RequestorNotAllowedToCreateUsers	The requesting member does not have permission to create a member.	403
UserCreate	RequestorNotFound	The requestor was not found.	404
UserCreate	RequestorPrivilegeInsufficient	You do not have permission to perform this action. Ask a full access member of your account for help.	403
UserCreate	RequestorPrivilegeInsufficientToCreateFullAccessUser	The requesting member does not have permission to create a full-access member.	403
UserCreate	UserPrimaryEmailVerificationDateNotValid	The verification date for the member's primary email address is not valid.	400
UserCreate	UserPrimaryEmailVerificationEntityNotValid	The node that verified the member's primary email address is not valid.	400
UserCreate	UserPrimaryEmailVerificationEntityRequired	The node that verified the member's primary email address must be identified.	400
UserCreate	UserPrimaryEmailVerificationStatusRequired	The verification status is required.	400
UserCreate	ValidPrimaryEmailVerificationDateRequired	The verification date for the user primary email address is required.	400
UserCreate	VerificationStatusNotConsistentWithVerifiedAttributeDeclaration	The verification status is not consistent with the declaration of a verified attribute.	400
UserDelete	AccountUserAlreadyDeleted	The member has already been removed.	400
UserDelete	LastFullAccessUserofAccountCannotBeDeleted	The last remaining full-access member in an account cannot be removed.	403
UserDelete	NodeUnauthorizedToActOnAccount	The request is not authorized.	401
UserDelete	NodeUnauthorizedToDeleteSuspendedUsers	The request is not authorized.	401

API	Error ID	Reason	Status
UserDelete	RequestorNotActive	The requestor is not active.	403
UserDelete	RequestorNotFound	The requestor was not found.	404
UserDelete	RequestorPermissionInsufficientToDeleteUser	The requesting member cannot delete the member.	400
UserDelete	RequestorPrivilegeInsufficient	You do not have permission to perform this action. Ask a full access member of your account for help.	403
UserDelete	RequestorPrivilegeInsufficientToDeleteFullAccessUser	The requesting member does not have permission to delete a full-access member.	403
UserDelete	UserSAMLTokenDeleteFailed	Deletion of the member's security token failed.	500
UserGet	AccountUserStatusDeleted	The member has been removed.	400
UserGet	NodeUnauthorizedToActOnAccount	The request is not authorized.	401
UserGet	RequestorNotActive	The requestor is not active.	403
UserGet	RequestorNotFound	The requestor was not found.	404
UserGetForDataSharing	DataSharingConsentDurationExceeded	The duration of the DataSharingConsent policy has been exceeded.	403
UserGetForDataSharing	DataSharingConsentRequired	The DataSharingConsent policy is required.	403
UserGetForDataSharing	NodeUnauthorizedToActOnAccount	The request is not authorized.	401
UserGetForDataSharing	RequestorNotFound	The requestor was not found.	404
UserListGet	AccountUserStatusDeleted	The member has been removed.	400
UserListGet	NodeUnauthorizedToActOnAccount	The request is not authorized.	401
UserListGet	RequestorNotActive	The requestor is not active.	403
UserListGet	RequestorNotFound	The requestor was not found.	404
UserResourceStatusUpdate	AccountActiveUserCountReachedMaxLimit	The maximum number of active members allowed has been reached.	400
UserResourceStatusUpdate	NodeUnauthorizedToActOnAccount	The request is not authorized.	401
UserResourceStatusUpdate	ResourceAlreadyInRequestedStatus	The resource is already in the requested status.	400
UserResourceStatusUpdate	ResourceStatusTransitionRequestedNotAllowed	The requested status transition is not allowed for the resource.	403
UserResourceStatusUpdate	StatusInvalid	The status is not valid.	400
UserResourceStatusUpdate	TOUPolicyRequiredToPromoteUserToActiveStatus	The Terms of Use have not been accepted.	403
UserUpdate	AccountUserAddressNotValid	The address is not valid.	400
UserUpdate	AccountUserAlternateEmailNotValid	The alternate email address is not valid.	400
UserUpdate	AccountUserCountryNotValid	The country is not valid.	400
UserUpdate	AccountUserEmailAddressDuplicated	The email address is a duplicate.	400
UserUpdate	AccountUserGivenNameNotValid	The given name is not valid.	400
UserUpdate	AccountUserLanguageDuplicated	The language is a duplicate.	400

API	Error ID	Reason	Status
UserUpdate	AccountUserLanguageNotValid	The language is not valid.	400
UserUpdate	AccountUserMobilePhoneNumberNotValid	The mobile telephone number is not valid.	400
UserUpdate	AccountUsernameRegistered	The sign-in name already exists.	400
UserUpdate	AccountUserPasswordNotValid	The password is not valid.	400
UserUpdate	AccountUserPrimaryEmailNotValid	The primary email address is not valid.	400
UserUpdate	AccountUserPrimaryLanguageNotValid	The primary language is not valid.	400
UserUpdate	AccountUserSecurityAnswerNotValid	The answer to the security question is not valid.	400
UserUpdate	AccountUserSecurityQuestionDuplicated	The security que`stion is a duplicate.	400
UserUpdate	AccountUserSecurityQuestionIDNotValid	The security question is not valid.	400
UserUpdate	AccountUserSurnameNotValid	The surname is not valid.	400
UserUpdate	AccountUserTelephoneNumberNotValid	The telephone number is not valid.	400
UserUpdate	CountryCannotBeChangedOnceSet	The country cannot be changed.	400
UserUpdate	CountryNotValid	The country is not valid.	400
UserUpdate	LastFullAccessUserCannotBeDemotedToStandardOrBasicPrivilege	The permission level of the last remaining full-access member in an account cannot be changed.	403
UserUpdate	NodeUnauthorizedToActOnAccount	The request is not authorized.	401
UserUpdate	NodeUnauthorizedToActOnUser	The request is not authorized.	403
UserUpdate	NodeUnauthorizedToUpdateUserCredentials	The node cannot change the member's security credentials.	403
UserUpdate	NodeUnauthorizedToUpdateUserInformation	The node is not authorized to update member information.	403
UserUpdate	NodeUnauthorizedToUpdateUserPassword	The node cannot change the member's password.	403
UserUpdate	PrimaryEmailConfirmationEndpointRequired	A confirmation endpoint is required for the member primary email address.	400
UserUpdate	PrimaryEmailVerifiedAttributeMustBeTrue	If the member's primary email address has been verified by the node, the setting of the PrimaryEmailVerified attribute must be set to TRUE.	400
UserUpdate	RequestorNotActive	The requestor is not active.	403
UserUpdate	RequestorNotAllowedToUpdateOtherUsers	The requesting member cannot update another member.	400

API	Error ID	Reason	Status
UserUpdate	RequestorNotAllowedToUpdateUserAccessLevel	The requesting member cannot update a member's permission level.	403
UserUpdate	RequestorNotAllowedToUpdateUserInformation	The requesting member cannot update member information.	403
UserUpdate	RequestorNotFound	The requestor was not found.	404
UserUpdate	RequestorPrivilegeInsufficientToUpdateUserClass	The requesting member does not have permission to change the member's permission level.	403
UserUpdate	StandardUserNotAllowedToUpdateFullAccessUserInformation	The member does not have permission to change the member's information.	403
UserUpdate	UserPrimaryEmailVerificationDateNotValid	The verification date for the member's primary email address is not valid.	400
UserUpdate	UserPrimaryEmailVerificationEntityNotValid	The node that verified the member's primary email address is not valid.	400
UserUpdate	UserPrimaryEmailVerificationEntityRequired	The node that verified the member's primary email address must be identified.	400
UserUpdate	UserPrimaryEmailVerificationStatusInvalid		
UserUpdate	UserPrimaryEmailVerificationStatusRequired	The verification status is required.	400
UserUpdate	UserPrivilegeCannotBeChanged	The member's permission level cannot be changed.	403
UserUpdate	UserStatusNotValid	The member's status is not valid.	400
UserUpdate	ValidPrimaryEmailVerificationDateRequired	The verification date for the user primary email address is required.	400
UserUpdate	VerificationStatusNotConsistentWithVerifiedAttributeDeclaration	The verification status is not consistent with the declaration of a verified attribute.	400
UserValidationTokenCreate	NodeUnauthorizedToActOnAccount	The request is not authorized.	401
UserValidationTokenCreate	RequestCannotBeServiced	The request cannot be serviced.	403
UserValidationTokenCreate	RequestorNotActive	The requestor is not active.	403
UserValidationTokenCreate	RequestorNotFound	The requestor was not found.	404
UserValidationTokenCreate	SecurityTokenResponseTypeNotValid	The security token response type is not valid.	400
UserValidationTokenCreate	TokenTypeNotValid	The token type is not valid	400
UserValidationTokenCreate	ULCPolicyMissingInAuthnRequest	The UserLinkConsent policy is missing.	403
UserValidationTokenCreate	UserIdentifierNotFound	The user ID was not found.	404
UserValidationTokenCreate	UserIdentifierRequired	A user ID is required.	400
UserValidationTokenCreate	UserStatusNotValid	The member's status is not valid.	400
UserValidationTokenCreate	ValidationTokenRetryLimitReached	The maximum number of validation token requests allowed for the member has been reached.	403

21.2 S-Host Error Messages

Error ID	Reason	Status
AccountMergeLoginNotAllowed	This account cannot be merged at this time. Please visit our Help & FAQs.	200
AccountMergeLoginNotAllowedUserPrivilegeNotFull	Only full members can merge accounts. To merge with this account, sign in as a full member of the second account.	200
AccountUserCredentialsInvalid	We don't recognize your sign-in name, your password, or both. Please try again.	200
AccountUserExceededAllowedFailedLoginAttempts	We don't recognize your sign-in name, your password, or both. Please try again.	200
AccountUserStatusLocked	Your membership is not in a valid status. Please contact Customer Support.	200
AccountUserStatusSuspended	Your membership is suspended. Please contact Customer Support.	200
CaptchaInputDoesNotMatch	The text you entered does not match the displayed image.	200
CaptchaInputRequired	Please enter the text you see in the image.	200
EmailNotValid	We don't recognize that email address. Please try again.	200
EmailNotVerified	The email address is unverified.	200
FormauthLaspBindingAccessPermission	You have not provided permission to use this service. Please contact Customer Support.	403
FormauthLaspFlippingLimit	You have switched back and forth too many times between two streaming services. Please try again later.	403
FormauthLaspLimitReached	You can only create two links to a streaming service that stays connected to devices such as a cable box, game console, smart TV, or connected Blu-ray player. To proceed, unlink one of your current links to this streaming service (from your Member Details page at uvvu.com) or check with the service for other options.	403
PasswordNotValid	We don't recognize your sign-in name, your password, or both. Please try again.	200
RequestorPrivilegeInsufficient	You do not have permission to perform this action. Ask a full access member of your account for help.	403
RequestorPrivilegeInsufficientToUpdateUserPolicies	You do not have permission to make this change. Ask a full member of your account for help.	403
SamlLogoutCancelledByUser	The request to unlink your UltraViolet account has been cancelled.	200
SignInCancelledByUser	The request to sign in to your UltraViolet account has been cancelled.	200
SubjectQueryNotSupported	Your request is not authorized. Please contact Customer Support.	200
TermsOfUseNotAcceptedByCLG	Your parent or legal guardian must accept the UltraViolet Terms of Use on your behalf before you can use this UltraViolet account.	400
TokenNotValid	The message you're using may have expired, or it may have been used before.	200

Error ID	Reason	Status
TokenNotValidForDelegation	The message you're using to link your account didn't work correctly. It may have expired, or it may have been used before.	200
TokenNotValidForResetPassword	The message you're using to recover your password didn't work correctly. It may have expired, or it may have been used before.	200
TokenNotValidForValidateEmail	The message you're using to validate your email didn't work correctly. It may have expired, or it may have been used before. Try requesting another message.	200
Unauthorized	The request is not authorized.	401
UnexpectedError	An unexpected error has occurred. Please try again.	200
UserCredentialRecoveryComplete	The request to recover your sign-in credentials for your UltraViolet account has been completed.	200

21.3 Security Layer Error Messages

Error ID	Reason	Status
bad_request	The request is not valid.	400
certificate_not_provisioned	The security token is required.	403
forbidden	The request is not authorized.	403
InvalidAssertion	The security token is required.	403
invalidDurationvalue	The security token's duration is not valid.	403
invalidtoken	The security token is not valid.	403
InvalidUserStatus	The request is not authorized.	403
SecTokenMergeReplacementRequired	A replacement security token is required.	403
token_rejected	The request is not authorized.	403
unauthorized	The request is not authorized.	403
UnsupportedHTTPMethod	The method is not supported.	501

22 Appendix C: Protocol Versions

DECE Protocol versions indicate the version of the Coordinator API specification, and are mapped to specific Coordinator API versions. The following table indicates the version URN, the corresponding Coordinator Specification, and the API endpoint BaseURL version.

Protocol Version	Specification Version	BaseURL	Description
urn:dece:protocolversion:legacy	v1.0	/rest/1/0	Applies to Device resources: indicates that the Device is a Legacy Device.
urn:dece:protocolversion:1.0	v1.0	/rest/1/0	Corresponds to the Coordinator specification versions 1.0 and 1.0.1.
urn:dece:protocolversion:1.0.2	v1.0.2	/rest/1/02	Corresponds to the Coordinator specification version 1.0.2.
urn:dece:protocolversion:1.0.5	V1.0.5	/rest/1/02	Corresponds to the Coordinator specification version 1.0.5.
urn:dece:protocolversion:1.0.6	V1.0.6	/rest/1/06	Corresponds to the Coordinator specification version 1.0.6.
urn:dece:protocolversion:1.0.7	V1.0.7	/rest/1/07	Corresponds to the Coordinator specification version 1.0.7.
urn:dece:protocolversion:1.1.1	V1.1.1	/rest/1/11	Corresponds to the Coordinator specification version 1.1.1 .
urn:dece:protocolversion:1.1.1	V1.2	/rest/1/11	Corresponds to the Coordinator specification version 1.2.
urn:dece:protocolversion:2.0	V2.0	/rest/2015/02	Corresponds to the Coordinator specification version 2.0.
urn:dece:protocolversion:2.1	V2.1	/rest/2015/03	Corresponds to the Coordinator specification version 2.1.
urn:dece:protocolversion:2.x	V2.x	/rest/2015/03	Corresponds to the Coordinator specification version 2.1x

Table 105: Protocol Versions

23 Appendix D: Policy Examples (Informative)

This Appendix intentionally left blank.

23.1 Parental-Control Policy Example

23.2 LockerDataUsageConsent Policy Example

23.3 EnableUserDataUsageConsent Policy Example

24 Appendix E: Coordinator Parameters

This section describes the operational usage model parameters used elsewhere in this document. Additional usage model variables are defined in Appendix A of [DSystem].

Parameter	Value	Description
DCOORD_DELETION_RETENTION	90	The retention period for a deleted User or Account resource.
DCOORD_DISCRETEMEDIA_LEASE_DURATION	6 hours	The maximum time the Coordinator shall allow a Discrete Media Lease to endure.
DCOORD_DISCRETEMEDIA_LEASE_EXPIRE_LIMIT	5	The maximum number of Discrete Media Rights that are allowed to expire automatically before the Node's ability to invoke the Coordinator's Discrete Media APIs is suspended.
DCOORD_DISCRETEMEDIA_LEASE_MAXTIME	24 hours	The maximum time a lease on a Discrete Media Right can be extended (renewed by).
DCOORD_EMAIL_ADDRESS_MAXLENGTH	256 characters	The maximum length allowed for an email address field.
DCOORD_E-MAIL_CONFIRM_TOKEN_MAXLIFE	72 hours	The maximum time the Coordinator shall allow an e-mail confirmation token be considered active and available for use.
DCOORD_E-MAIL_CONFIRM_TOKEN_MINLENGTH	16 characters	The minimum allowed length for the e-mail confirmation token created by the Coordinator
DCOORD_E-MAIL_CONFIRM_TOKEN_MINLIFE	24 hours	The minimum time the Coordinator shall allow an e-mail confirmation token to be considered active and available for use.
DCOORD_MAX_USERS	6	The maximum number of users in a single account.
DCOORD_MAX_PENDING_USER_TOKEN_DURATION	DCOORD_E-MAIL_CONFIRM_TOKEN_MAXLIFE	The maximum token duration for a user in `pending` status. Note that when the Coordinator automatically validates email this parameter is irrelevant (See Section 14.1.3).
DGEO_AGEOFMAJORITY	See applicable Geography Policy	the age of a majority for that particular jurisdiction, such that at or above this value, the User is considered to have reached the age of majority

Parameter	Value	Description
DGEO_CHILDUSER_AGE	See applicable Geography Policy	the age of a User, such that for users under this value, the Coordinator can implement special legal or operational considerations when providing services to children.
DCOORD_FAU_MIN_AGE	See applicable Geography Policy	The minimum age required to allow a User to be granted the Full Access User role
DCOORD_SAU_MIN_AGE	See applicable Geography Policy	The minimum age required to allow a User to be granted the Standard Access User role
DCOORD_BAU_MIN_AGE	See applicable Geography Policy	The minimum age required to allow a User to be granted the Basic Access User role
DCOORD_STREAM_INFO_MIN_RETENTION	30 days	The minimum duration of Stream information retention
DCOORD_STREAM_RENEWAL_MAX_ADD	6 hours	The maximum duration a Stream can be renewed for.
DCOORD_STREAM_MAX_TOTAL	24 hours	The overall maximum duration of a Stream
DCOORD_STREAM_CREATED	30 days	Threshold for how long ago an already deleted Stream was created.
DCOORD_JOIN_CODE_MAX_ACTIVE	6	The maximum number of allowed outstanding active Join Codes for an Account
DCOORD_VALIDATION_TOKEN_RETRY_LIMIT	3	The maximum number of consecutive UserValidationTokenCreate API invocations allowed per email address
DCOORD_VALIDATION_TOKEN_RETRY_TIMEOUT	15 minutes	The time after which the retry counter is reset by the Coordinator for the UserValidationTokenCreate API and supplied `UserIdentifier` parameter.
DCOORD_VALIDATION_TOKEN_MAX_LENGTH	12 bytes	The maximum length of a validation token in bytes. User interfaces implement to this length.
DCOORD_VALIDATION_TOKEN_TYPICAL_LENGTH	8 bytes	The typical length of a validation token in bytes. This is to be used except under circumstances where this length will result in tokens that are not sufficiently unique. The Coordinator need not generate tokens longer than this value.

Parameter	Value	Description
DCOORD_VALIDATION_DELEGATIONTOKEN_MAXLIFE	6 hours	The maximum token validity period for verification tokens of type `urn:dece:type:token:delegationtokenrequest`
DCOORD_CONFIRMATION_AGE	3 years	The maximum amount of time that is allowed to have transpired since a previous email confirmation. See sections 14.1.3.3 and 14.2.12
DCOORD_MERGE_SESSION_AGE	24 hours	The maximum age of a User Agent (session) between a Node and the User Agent.
DCOORD_MERGE_UNDO_PERIOD	180 days	The maximum duration of the period during which a Merge operation may be undone.
DCOORD_DATA_SHARING_CONSENT_DURATION	15 minutes	The maximum duration following the creation of `DataSharingConsent` policy that a Node can request User data for the purpose of creating a remote (i.e., Node) user account .
DCOORD_USERNAME_SEARCH_MIN_LENGTH	3 characters	The minimum length of a username substring search value
DCOORD_EMAIL_SEARCH_MIN_LENGTH	7 characters	The minimum length of an email substring search value
DCOORD_USERLIST_SEARCH_MAX_SIZE	256	The maximum number of elements in the UserList that may be returned following a ResourcePropertyQuery request.
DCOORD_TEST_INDICATOR	`test_-_`	The match strings for identification of test Accounts and test Users that may be physically removed by the Coordinator
DCOORD_TEST_PRESERVE_INDICATOR	`test_-keep-`	The set of match strings for identification of test Accounts and Users that will not be physically removed by the Coordinator
DCOORD_USERLESS_ACCOUNTS_THRESHOLD	60 days	The duration after which a userless account can be physically removed
DCOORD_INACTIVITY_THRESHOLD	730 days	The duration after which an account with rights tokens is considered inactive.

Parameter	Value	Description
DCOORD_DEIDENTIFY_ACCOUNT_THRESHOLD	365 days	The duration after which an account and all users under it can be deidentified.
DCOORD_DEIDENTIFY_USER_THRESHOLD	365 days	The duration after which a user account can be deidentified.
DCOORD_DEIDENTIFIED	@@@deidentified@@@	The string with which selected properties of a deidentified account/user need to be replaced.
DCOORD_PAGINATION_THRESHOLD	1000	The maximum number of Rights Tokens (or references) returned by the Coordinator.

25 Appendix F: Geography Policy Requirements (Normative)

DECE services shall be launched to serve specific geographic regions that may include one or more countries, provinces, or other jurisdictional regions. The provision of services in each of these regions may require modifications to the operational characteristics of the Coordinator and the Nodes it serves.

Because of these differences, each operating region will require the creation of jurisdiction-specific profile of this specification, and potentially other specifications. [DGeo] addresses the mandatory and optional information that needs to be defined in order to operate within the requirements and obligations of these regions. Implementations will be required to consult [DGeo] for their applicable region(s).

26 Appendix G: Field Length Restrictions

While the XML Schema defined in this specification does not limit CDATA lengths, there are practical limitations required to be enforced by the Coordinator. This Appendix documents those length restrictions.

26.1 Limitations on the User Resource

Property Name	Maximum length	Comments
GivenName	64 characters	
SurName	64 characters	
PrimaryEmail - Value	256 bytes	
AlternateEmail – Value *1	256 bytes	
Address – PostalAddress *2	256 characters	(limit number of address lines to 3)
TelephoneNumber - Value	17 bytes	
MobileTelephoneNumber - Value	17 bytes	
Username	64 bytes	
Password	256 bytes	
DeviceJoinCode	15 bytes	
EmailConfirmationToken	16 bytes	
Language	16 bytes	predefined list
Country	2 bytes	predefined list
Display Image URL (or)	256 bytes	
Display Image Data		5MB (will be resized)
Locality (city)	128 characters	
PostalCode	16 bytes	
StateOrProvince	128 characters	

26.2 Limitations on the Account Resource

Property Name	Maximum length	Comments
DisplayName	256 characters	

Country	2 bytes	(predefined list)

26.3 Limitations on the Rights Resource

Property Name	Maximum length	Comments
ALID	256 bytes	
ContentID	256 characters	
LicenseAcqBaseLoc	256 bytes	
MediaProfile	64 bytes	
DisplayName(RightsSoldAs)	256 characters	
BundleID	256 bytes	
ProductID	128 bytes	
Location	256 bytes	
RetailerTransaction	256 bytes	
TransactionType	256 bytes	
StreamClientNickname	256 bytes	
CalculationMethod	128 characters	
ViaProxy	32 characters	
Confidence	20 characters	
Resource	128 bytes	
RequestingEntity	128 bytes	

26.4 Limitations on the DigitalAsset Resource

Property Name	Maximum length	Comments
APID	256 bytes	
ContentID	256 bytes	
Description	256 bytes	
Audio-Type	16 bytes	
Audio-Codec	32 bytes	
Audio-CodecType	256 bytes	
Audio-BitrateMax	8 bytes	
SampleRate	8 bytes	

SampleBitDepth	8 bytes	
Audio-Language	16 bytes	
Channels	16 bytes	
Audio-TrackReference	256 bytes	
Video-Type	16 bytes	
Video-Codec	32 bytes	
Video-CodecType	256 bytes	
MPEGProfile	256 bytes	
MPEGLevel	16 bytes	
Video-BitrateMax	8 bytes	
AspectRatio	16 bytes	
PixelAspect	16 bytes	
WidthPixels	16 bytes	
HeightPixels	8 bytes	
ActiveWidthPixels	8 bytes	
ActiveHeightPixels	8 bytes	
FrameRate	8 bytes	
ColorType	16 bytes	
SubtitleLanguage	16 bytes	predefined language list (metadata)
Video-TrackReference	256 bytes	
Format	16 bytes	
Subtitle-Description	64 bytes	
Subtitle-Type	32 bytes	
FormatType	16 bytes	
Subtitle-Language	16 bytes	
Subtitle-TrackReference	256 bytes	
Image-Width	8 bytes	
Image-Height	8 bytes	
Image-Encoding	256 bytes	
Image-TrackReference	256 bytes	
Interactive-Type	256 bytes	
Interactive-Language	16 bytes	(predefined list)

Interactive-TrackReference	256 bytes	

26.5 Limitations on the LogicalAsset Resource

Property Name	Maximum length	Comments
Version	8 bytes	
ALID	256 bytes	
ContentID	256 bytes	
ContentProfile	64 bytes	
DiscreteMediaFulfillmentMethods	256 bytes	
AssentStreamLoc	256 bytes	
FulfillmentGroupID	128 bytes	
LatestContainerVersion	32 bytes	
ActiveAPID	256 bytes	
ReplacedAPID	256 bytes	
RecalledAPID	256 bytes	
ReasonURL	256 bytes	
country	2 bytes	Predefined list
countryRegion	32 bytes	
allowedDiscreteMediaProfile	64 bytes	

26.6 Limitations on the RightsToken Resource

Property Name	Maximum length	Comments
ALID	256 bytes	
ContentID	256 bytes	
BundleID	256 bytes	
DisplayName	256 characters	
Language	16 bytes	Predefined list
ProductID	128 bytes	
MediaProfile	256 bytes	

26.7 Limitations on the BasicAsset Resource

Property Name	Maximum length	Comments
ContentId	256 characters	
UpdateNum	8 bytes	
WorkType	32 bytes	
PictureFormat	16 bytes	
ReleaseYear	16 bytes	
RunLength	32 bytes	
SequenceNumber	8 bytes	
HouseSequenceNumber	32 characters	
BasicAsset LocalizedInfo		
Language	16 bytes	
TitleDisplay19	19 characters	
TitleDisplay60	60 characters	
TitleSort	256 characters	
Summary190	190 characters	
Summary400	400 characters	
Summary4000	4000 characters	
VersionNote	256 characters	
OriginalTitle	256 characters	
CopyrightLine	512 characters	
Genre	64 characters	
Keyword	64 characters	
ArtReference/Value	256 bytes	
ArtReference/Resolution	32 bytes	
People/Name/SortName	256 characters	
People/Name/DisplayName	256 characters	
People/Name/FirstGivenName	64 characters	
People/Name/SecondGivenName	64 characters	
People/Name/FamilyName	64 characters	
People/Name/Suffix	16 characters	
People/Name/Moniker	64 characters	

People/Job/JobFunction	16 bytes	
People/Job/@scheme	32 bytes	
People/Job/JobDisplay	64 bytes	
People/Job/BillingBlockOrder	8 bytes	
People/Job/Character	64 bytes	
Region-type/Country	2 bytes	Predefined values
Region-type/CountryRegion	32 bytes	Predefined values
ReleaseHistory-type/ReleaseType	32 bytes	
AssociatedOrg/DisplayName	256 characters	
AssociatedOrg/SortName	256 characters	
AssociatedOrg/@OrganizationID	256 bytes	
AssociatedOrg/@role	256 bytes	
ContentRatingDetail-type/System	32 bytes	
ContentRatingDetail-type/value	32 bytes	
AltIdentifier/Namespace	256 bytes	
AltIdentifier/Identifier	256 bytes	
AltIdentifier/Location	256 bytes	
People/Identifier/Identifier	256 bytes	
People/Identifier/Namespace	256 bytes	
People/Identifier/ReferenceLocation	256 bytes	

26.8 Limitations on the Bundle Resource

Property Name	**Maximum length**	**Comments**
BundleID	256 byte	
DisplayName	256 characters	

26.9 Limitations on CompObj Resource

Property Name	**Maximum length**	**Comments**
DisplayName	256 characters	

26.10 Limitations on Legacy Device Resource

Property Name	Maximum length	Comments
DeviceID	256 bytes	
DisplayName	128 characters	
Model	64 characters	
SerialNo	64 bytes	
MimeType	32 bytes	Predefined list
Brand	128 characters	
Manufacturer	256 characters	
ManagingRetailer	128 characters	
Width	10 bytes	
Height	10 bytes	
Image	256 bytes	
ManageRetailerURL	256 bytes	

27 Appendix H: User Status and APIs Availability

The following represents whether the Coordinator will accept a call to the listed API based on the status of the User as determined from the ResourceStatus field of the User Resource; that User being the subject of the Delegation Token used in an API request.

Note that in the case of Customer Support (CS) subrole, the agent identifies the User, then the Node obtains a Delegation Token.

In the table below:

- a dot indicates the API is accessible.
- “NA” means not applicable
- “portal” means the API is only accessible to the portal Role

Where APIs can be invoked with either User or Account Security Token Subject Scope, the table only applies when that scope is User.

API \ User Status	pending		active		blocked :clg		blocked :tou		deleted		merge deleted		suspended	
	Role	CS	Role	CS	Role	CS	Role	CS	Role	CS	Role	CS	Role	CS
AccountGet		●	●	●		●		●		●		●		●
AccountDelete		●	●	●		●		●		●		●		●
AccountUpdate		●	●	●		●		●		●		●		●
AccountMerge		●	●	●		●		●		●		●		●
AccountMergeTest		●	●	●		●		●		●		●		●
RightsTokenCreate	●	●	●	●										
RightsTokenGet	●	●	●	●										
RightsTokenDelete	●	●	●	●										
RightsTokenUpdate	●	●	●	●										
RightsTokenDataGet	●	●	●	●										
RightsTokenDataGet (DRMClientID)			NA	NA										
RightsLockerDataGet	●	●	●	●										
DiscreteMediaRightCreate	●	●	●	●										
DiscreteMediaRightGet	●	●	●	●										
DiscreteMediaRightConsume	●	●	●	●										
DiscreteMediaRightList	●	●	●	●										
DiscreteMediaRightLeaseCreate			●	●										
DiscreteMediaRightLeaseRelease			●	●										
DiscreteMediaRightLeaseRenew			●	●										
DiscreteMediaRightLeaseConsume			●	●										
DiscreteMediaRightUpdate			●	●										
DiscreteMediaRightDelete			●	●										
PolicyCreate	●	●	●	●			●	●						
PolicyGet	●	●	●	●			●	●						
PolicyDelete			●	●										
PolicyUpdate	●	●	●	●			portal							
StreamCreate			●	●										
StreamView			●	●										
StreamListView			●	●										

API \ User Status	pending		active		blocked :clg		blocked :tou		deleted		merge deleted		suspended	
	Role	CS	Role	CS	Role	CS	Role	CS	Role	CS	Role	CS	Role	CS
StreamRenew	[2]	[2]	●[2]	●[2]	[2]	[2]	●[2]	[2]	[2]	[2]	[2]	[2]	[2]	[2]
StreamDelete	[2]	[2]	●[2]	●[2]	[2]	[2]	●[2]	[2]	[2]	[2]	[2]	[2]	[2]	[2]
UserCreate		●	●	●		●		●		●		●		●
UserGet	portal	●	●	●		●	portal	●		●		●		●
UserList		●	●	●		●		●		●		●		●
UserDelete		●	●	●		●		●		●		●		●
UserUpdate	portal	●	●	●		●		●		●		●		●
UserValidationTokenCreate (with security token)		●	●	●		●	●[3]	●[3]		●		●		●
UserValidationTokenCreate (no security token)		●	●	●		●	●[3]	●[3]		●		●		●
AssetMapALIDToAPID/APIDToALID Get (User level)			●	●										
Security Token Service (user password profile)	●	●[3]	●	●[1]		●[1]	●			●[1]		●[1]		●[1]
Security Token Service (Device Auth profile)			●				●							
Security Token Service (SAML2 profile)	●		●				●				●			
Authentication (S host)	●		●				●							

[2] DLASPs have access only where indicated. Other LASPs access this API with Account level scope so User status is irrelevant.
[3] Only for the urn:dece:role:dece:customersupport Role. See [DsecMech] section 8.1.4 for special considerations.

28 Appendix I: Requirements for Google Pub/Sub support

The new Publish and Subscribe service supported by the Coordinator may support multiple established services, however this initial release of the new service will incorporate the Google Pub/Sub service. As additional services are supported, this section will be updated to include the necessary information for each service.

28.1 Requirements for the Google Pub/Sub service

The Google Pub/Sub service is documented at [1] below. There are restrictions on its' use such as a requirement to supply an HTTPS [TLS] endpoint, however the x509 certificate that enables the TLS connection, is arbitrary. Its' purpose is purely for message confidentiality.

The body of the message received from Google will match the data model described in section 18.1.3 base64 encoded within the standard PubSub JSON object.

In order to use this service, the Coordinator manages the subscription process on behalf of all Nodes eligible to subscribe for notifications.

During initial Node provisioning, Nodes may indicate interest to the Coordinator in using the notification service, at which point the following information needs to be provided (via a separate document):

- The event types the Node wishes to receive
- The TLS-protected URL(s) to which notifications will be sent
- Timeout value for messages. This value also controls how frequently missed messages are held by the Pub/Sub system before any reattempts at delivery are made.

Should a Node wish to terminate their Subscription they are required to contact the Coordinator.

[1] Google Pub/Sub Documentation: https://cloud.google.com/pubsub/docs

END