Integrating the Healthcare Enterprise

IHE IT Infrastructure (ITI)
Technical Framework
White Paper

Metadata Versioning

October 10, 2008

Table of Contents

1 Open Issues

MV003: Are there use cases for updating the metadata of Folder objects? Now that Folders are used in Content Profiles the ability to update their metadata may be necessary.

MV007: (The wording is weird because I also published this note on the Implementation Guide.) Document Source(s) can submit a document multiple times. This results in a XDSDocumentEntry.uniqueId being present on multiple XDSDocumentEntry objects in the Registry. As long as the size and hash attributes are the same, this is considered a normal condition that a Document Consumer must be prepared. This condition can occur for two other reasons. First a Provide and Register transaction can fail because the response message from the Repository back to the Document Source is not delivered. This can happen on a synchronous connection as well as with asynchronous web services as described in the new Async XDS Supplement. A natural response for the Document Source is to resubmit, again causing duplication.

A document replace (RPLC from LifeCycle Management option to XDS) has the known issue that a Document Source replacing a document is not required to find all copies of the document in the Registry. A replacement will be applied to only one copy of the document. Likewise the new topic of Metadata Versioning (currently a white paper) does not carry the requirement to find all copies of the document metadata.

MV012: Should the comment attribute on SubmitObjectsRequest be allowed or disallowed?

MV015: The new actor name, XDS Admin actor, is a horrible name. I include it for now but hope to get a better suggestion through public comment. An alternate name “Document Metadata Updater” was suggested. I made no changes so far since the suggestion came in a bit late.

MV016: It is now acceptable to update the XDSDocumentEntry.patientId attribute. But, there is no facility for updating the XDSSubmissionSet.patientId attribute (or other attributes).

MV017: An attempt to retrieve a deleted document should fail but there is no interaction between the Repository and Registry to allow enforcement.

MV018: Rob's coded availability attribute needs to be used and documented.

MV020: Audit event for Update transaction needs to be documented. Both lid and id attributes must be logged.

MV022: Should a named option be created for the Document Consumer actor? This option would signal that the implementation is compatible with a Document Registry that implements Metadata Versioning.

MV023: Should we control the result of a delete operation? If an implementer decides to allow real deletes (remove documents and DocumentEntry objects) should the profile require they also delete the SubmissionSet and Associations linked to the document?

2 Closed Issues

MV001: Should UpdateReason values be prefixed by a namespace? See MV013 for discussion.

MV002: Should UpdateReason, as specified in the SubmissionSet Association be a classification instead of a slot? See MV013 for discussion.

MV004: Should a Deprecate request use a (new) Deprecates Association Type instead of HasMember? The text has been updated to use a new Deprecate Association type.

MV005: Should a move to off-line request use a (new) Offline Association Type instead of HasMember? A new Association type of Offline has been created.

MV006: Should a deletion use a (new) Deletes Association Type instead of HasMember? A new Association type of Delete is used.

MV007: Could Metadata Versioning support Auto-Summary documents? No. We use the term auto-summary to refer to documents that are generated on demand with the most up to date information. A medical summary created at the time of retrieval is a good example. Currently several implementations are using the existing document replacement facility. This has been viewed by some as too heavy-weight. Such a replacement cannot be implemented without the use of a Submission Set since it is the creationTime attribute of the Submission Set that would be used to label when the summary was generated. If we created some new special semantic to attempt to simplify/minimize the metadata necessary to record the generation of a new version of an auto-summary document, it would only remove the need for the RPLC association linking the new version to the old version.

MV008: The use of a code to label the type of update has been questioned. Do we need to restrict the use of this mechanism to approved types of updates? If the coded labeling of update types is discarded then the UpdateReason Slot on the Association should be removed and instead we should require a free text comment in the //ExtrinsicObject/VersionInfo/@comment field

The current version of the paper removes the UpdateReason slot entirely and instead lists specific DocumentEntry attributes that must be maintained across versions.

MV009: In existing XDS, a document can be submitted multiple times with same uniqueID as long as hash is identical. How should this be handled if the multiply submitted document is version 3 of a document? Since the actual document is not being updated, the hash is not useful. Is it legal, according to ebRIM and ebRS for there to be two ExtrinsicObjects in the Registry with the same version? If id is same, if different? It seems that the second should be ignored if the metadata is exactly the same and rejected if it is not. The second case probably coming from two different stations trying to do different updates at the same time. Registry Adaptor could assign the version number. Seems that there could be pitfalls to this approach.

A closer reading of the standard (ebRIM) shows that the registry assigns the version numbers so this is an implementation issue for the Document Registry actor and not relevant to this specification.

MV010: Metadata versioning would seem to make obsolete the existing discussion on precedence of RPLC/XFRM/APND relationships and which ones deprecate previous content. Metadata versioning gives the opportunity to surgically deprecate documents, no need for fancy rules.

This is no longer an issue.

MV011: Offline (as in removed from available storage) needs to be coded separately from UpdateReason and status. Later if returned to available storage, need to restore its prior status.

This has been resolved by storing the original DocumentEntry status in a Slot on the Offline Association and establishing rules for its restoration.

MV013: The UpdateReason attribute may be too restrictive and of very little value since the differences between to versions of an ExtrinsicObject can be determined by comparison.

The UpdateReason attribute of the Submission Set Association has been removed. First it restricts the use of this mechanism. To be useful from an interoperability perspective it would have to be a coded term. In its place is a list of ExtrinsicObject attributes that may not be updated.

MV014: The functionality described in this paper should become a named option on the XDS.b profile. A named option has been introduced.

MV014: (oops on duplicate number) The marking of a document as Offline or Online can be done by submitting an update to the DocumentEntry. It is currently specified as being controlled by the submission of an Offline Association. We need to determine which approach to stick with. I cannot see a strong reason to pick either one.

MV015: There is currently no way to issue a Stored Query asking for only online documents. Add a query parameter?

MV019: A Stored Query initiated by the Doc Con does not return DocumentEntry objects marked as deleted. If the request comes from the XDS Admin Client actor they are returned. There is currently no way to distinguish between these requests.

Stored Query operation does not depend on the actor initiating. DocumentEntry objects marked as Deleted are never returned.

MV021: For now sending metadata updates through the Repository actor is acceptable. This enables early testing and research. Should this capability be kept or abandoned long term thus requiring a direct connection between Document Source and Document Registry? The introduction of the XDS Admin Client and XDS Admin actors would imply the direct connection.

Update transaction always goes from XDS Admin actor directly to the Document Registry actor.

3 Introduction

We have many new Document Sharing use cases we cannot satisfy using the current XDS profile and the subset of features it uses from the OASIS ebXML Registry standard. The original XDS was based on ebRIM and ebRS version 2.1. XDS.b and XCA use ebRIM and ebRS 3.0. This white paper explores a collection of use cases that can be satisfied by introducing Metadata Versioning, a mechanism described in ebRIM 3.0 and ebRS 3.0, into XDS.b. Furthermore it makes recommendations on how to satisfy these use cases using Metadata Versioning along with current XDS.b features.

A current mechanism in XDS, called Document Replacement, is capable of replacing a document, its contents in the Document Repository, along with its metadata in the Registry. This new mechanism is focused on making updates to the metadata in the Registry while maintaining the existing Repository contents.

We start by introducing the target use cases, then describe the Metadata Versioning features available in ebXML Registry version 3.0 and how we plan on using them, and finally we revisit each use case and describe how each use case can be satisfied. Since this work depends on the version 3.0 registry standard, it does not attempt to update the XDS.a profile.

4 Use Cases

The following use cases can be satisfied by using metadata versioning and existing XDS.b features. These use cases have been proposed in IHE committees or national projects looking to adopt XDS.

4.1 Update Patient Demographics

This provides the ability to change key patient demographics attributes on a document. This would follow the lead set by Radiology that patient demographics are allowed to be more changeable than clinical information.

4.2 Update Confidentiality Code

The confidentialityCode attribute of a document may be changed many times over the useful life of a document to maintain the privacy aspects of the document.

4.3 Deprecate Document without Replace

The only current mechanism for deprecating a document, marking it as no longer useful, is to replace it with a new version. Documents become un-useful or not appropriate for day-to-day usage. Changing a document's status to Deprecated allows the document to be hidden from casual use but still be available when deeper investigation is called for. This use case discusses how to deprecate a document, changing its status from Approved to Deprecated, without replacing it.

4.4 Update Document Availability

Documents in a Document Repository actor can change availability. Examples are:

- Old documents are moved to 'less accessible' media
- Documents are permanently removed from service

A seemingly unrelated use case is important here because it also originates at the Document Repository actor. Repository maintenance, unrelated to document availability, can require updates to metadata. Two use cases are:

- A Repository server is split into two to manage a growing number of documents. It is decided to create a new network presence for the new server (the split could have been hidden using network routing tricks). This leads to the creation of a new repositoryUniqueId for the new Repository. The metadata for the documents moved must be updated.
- An extension of this use case is to consider that this Repository also offers the XDS.a Retrieve transaction which is dependent on the XDSDocumentEntry.URI attribute. The metadata update would update this attribute as well.

4.5 Delete Document

The French National Project has asked for a way to delete a document and its metadata. While more detail is needed to support this use case, the technical underpinnings are described.

5 OASIS ebXML Registry 3.0 support for Metadata Versioning

This work is enabled by new metadata management features introduced into version 3.0 of the ebXML Registry standard (ebRIM and ebRS). The key new concept is the ability to maintain multiple copies or versions of a metadata object, such as an ExtrinsicObject, which represents in metadata a single real document in a repository. In terminology of the standard, all objects stored in the registry are called Registry Objects. A new concept is the Registry Object Instance, a specific version of a Registry Object. Collectively, all the versions of a Registry Object are called a Logical Registry Object.

All Registry Objects are identifiable by their id attribute. All objects in the registry must have a unique value for their id attribute. This sense of identity is maintained when metadata versioning is introduced. To allow a collection of registry objects to be grouped together and be identified as versions of the same object, two new attributes are introduced: the Logical ID and the VersionInfo element. While all objects must have unique value for their id attribute to maintain their identity, all objects that are versions of the same logical object have the same logical id or lid attribute.

The simplest form of an ExtrinsicObject, as specified in ebRIM 2.1, looks like:

```
<ExtrinsicObject id="urn:uuid:123...">
</ExtrinsicObject>
```

containing only an id attribute giving it its identity. Note that we use a shortened format for UUIDs to improve readability. In ebRIM 3.0 this same object could be coded as:

```
<ExtrinsicObject id="urn:uuid:123..." lid="urn:uuid:123...">
</ExtrinsicObject>
```

In this version of the standard, the registry object still maintains its identity by having a unique value for the id attribute. The presence of the lid attribute having the same value as the id attribute indicates that this is the first, the original, version of the object. Current XDS, XDS.a and XDS.b, return this format from a Stored Query since that transaction is coded in ebRIM 3.0. Future versions of an object, that is version 2..n of an object, will have new values for the id attribute but the original value for the lid attribute. The value of the lid attribute is always equal to the value of the id attribute of the first version of the registry object. To query, in SQL, for all versions of this ExtrinsicObject one would use an SQL clause of

lid='urn:uuid:123...'

and each ExtrinsicObject returned would have the same value for the lid attribute but a different value for its id attribute.

Note that while the lid attribute labels the various versions of a registry object and the id attribute allows each version to be addressable by being unique, these two attributes are inadequate for determining which is the first version, the second version, etc. For this the standard introduces the VersionInfo element which looks like:

```
<VersionInfo versionName="2" comment=""/>
```

This element is required in all registry objects. In XDS, this means it is required in ExtrinsicObject, RegistryPackage (submission set and folder), and Association. It is also required on Classification and ExternalIdentifier objects.

The ebRIM 3.0 specification introduces rules on the behavior of versions of registry objects. All versions of an ExtrinsicObject reference the same document in the repository. For XDS.b this implies that all versions of a DocumentEntry/ExtrinsicObject must carry the same value for the XDSDocumentEntry.uniqueId attribute since it is used to reference the actual document in the repository.

An Association always references a specific version of a registry object. Its attributes sourceObject and targetObject hold the id attribute of the object being pointed at.

The lid attribute is not required in the submission of a registry object. The value of the lid attribute defaults to the value of the id attribute thus creating the first version of a registry object by default.

The VersionInfo attribute is never submitted to the registry, it is generated by the registry and always returned from a query. The versionName portion of the VersionInfo attribute is automatically allocated and set by the registry. It defaults, in the standard, to versionName="1.1" but the registry implementation is not constrained in what numbering scheme it uses. The comment portion of the VersionInfo attribute is set by the comment attribute of the <rim:Request/> element. In XDS.b this corresponds to the <rim:SubmitObjectsRequest/> element that is coded in the Provide And Register Document Set and Register Document Set transactions. Note that this comment applies to all objects submitted in the request.

The ebRS 3.0 standard introduces a new request/verb, UpdateObjectsRequest, for submitting updates to a registry object. Note that updates can be carried in either UpdateObjectsRequest or SubmitObjectsRequest requests according to ebRS 3.0.

When a registry object (such as an ExtrinsicObject) is updated, the submitter must query the registry for the previous version, update the parts of the object as needed and resubmit the entire registry object. Individual attributes cannot be updated.

The ebRIM 3.0 standard allows for the creation of new status attribute values.

6 Proposal for Use in XDS.b

The existing transactions in XDS.b are adequate to support the new functionality presented in this paper. But, the new functionality can be categorized as being 'administrative' in nature. Each use case presented:

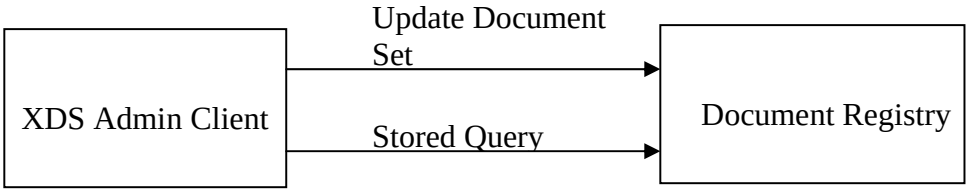
1. Is beyond the basic submit/query/retrieve semantics provided in the rest of the XDS.b profile.
2. Is likely to be governed by a different and more restrictive authorization rules

XDS actors that would potentially generate and submit the updates are:

- Document Source to update patient demographics and confidentiality code; and to deprecate and delete
- Document Repository to update the availability status of documents

In both cases an integrated Document Consumer would be needed to provide access to the Stored Query transaction.

For these reasons, a new actor is introduced with the name XDS Admin Client. A new named option to the Document Registry actor will be used to describe the added functionality in the Document Registry actor.



The XDS Admin Client actor uses the Stored Query [ITI-18] and a new Update Document Set [ITI-XX] transactions to query and issue metadata updates to the Document Registry actor. The XDS Admin Client actor is the only actor authorized to perform metadata updates. The new Update Document Set transaction uses the ebRS 3.0 SubmitObjectsRequest.

The need for access to the Stored Query transaction is motivated by the nature of the update metadata mechanism:

1. Read existing metadata object (XDSDocumentEntry) through Stored Query
2. Update object XML
3. Submit updated object as new version.

The Update Document Availability use case requires a binding between the Document Repository and XDS Admin Client actors. The Document Repository has the knowledge of what and how to update while the XDS Admin Client actor has the ability to perform the update.

In all cases, the Document Registry actor is the recipient of the Stored Query and Update Document Set transactions issued by the XDS Admin Client actor.

The following issues motivate the creation of the new actors:

1. Makes the description of functionality and mechanism easy since it is bound to a specialized admin actor.
2. Allows documentation of new functionality as an option on the XDS.b profile for which vendors can formally declare their support.
3. New and different risk analysis is necessary. This is to be kept separate from concerns regarding the 'plain' XDS actors. It is forecast for instance that a separate or enhanced authentication may be required to perform these administrative function because of specific risk.

6.1 A new perspective on metadata

In the past a single DocumentEntry object in the Document Registry represented a single Document in the Document Repository. With the Document Registry maintaining multiple versions of metadata, some basic premises about metadata change.

1. It has always been the case that a document can be registered multiple times and as long as the size and hash attributes are identical, it can be registered multiple times with the same XDSDocumentEntry.uniqueId
2. Now with Metadata Versioning, an additional DocumentEntry exists in the Registry for each version of the metadata. All versions of the DocumentEntry carry the same XDSDocumentEntry.uniqueId.

6.2 Mechanisms for Updating Metadata

Three different mechanisms are introduced:

Metadata Versioning where a new version of a registry object is submitted.

Status Updating where the submission of a Association object triggers a side-effect in the registry. The Association attribute targetObject identifies the object to be updated and the name of the Association indicates the nature of the side-effect. The most common side-effect is to change the status attribute of the targetObject.

Document Status Slot is new and is used to record the status of the document in the repository. This slot can take on two values, Online and Offline. If the slot is not present then the default value is Online. The Online value indicates that the document in the Repository is available for retrieve.

6.3 Rules for use of Updating Metadata

The key issue in introducing metadata updating features into XDS.b is to make them work well with other design aspects of XDS.b. The following rules shall govern the operation of Metadata Versioning:

1. SubmissionSet objects are not versioned. The lid attribute shall always be equal to the id attribute or not present which implies that lid is equals to id. This does not change the rules for lid management as documented in ebRIM 3.0. It does acknowledge that lid is optional on submit.
2. Folder objects are not versioned.

3. The first version of a DocumentEntry, in a Register Document Set transaction, can be identified by the lack of a lid attribute or by having a lid attribute equal to its id attribute.

4. DocumentEntry objects (ExtrinsicObjects) are versioned. Submitting a new version of a DocumentEntry requires a SubmissionSet and an Association just like the original submission.

5. The Document Registry actor, upon receipt of new version of a DocumentEntry, shall label the previous version with status of Deprecated if that previous version has status of Approved. If the previous DocumentEntry has a status other then Approved then the status shall not be altered. The result is that at most one version of a DocumentEntry shall have status of Approved.

6. An update, a submission of a newer version of a DocumentEntry object, shall have a lid attribute in UUID format that matches the id attribute of a DocumentEntry already in the Registry. The id attribute of the new DocumentEntry may be in UUID or symbolic format. If it is in UUID format, it must not exist in the registry and it must not be equal to the lid attribute. If the id attribute has a symbolic format, the Document Registry actor shall assign a new UUID.

7. The Document Registry actor shall allocate values for the versionName (version number) attribute based on the order of arrival. The first version of an ExtrinsicObject shall have version 0. Subsequent versions shall increment this value treating it as an integer.

8. If a DocumentEntry contains a versionInfo attribute in the Register Document Set or Update Document Set transaction, the Registry actor shall ignore and overwrite its contents.

9. Updates shall be submitted using the SubmitObjectsRequest ebRS 3.0 request but labeled in the ws:Action as an update. See the 'Web Services Definitions' section for details. Update requests may only contain DocumentEntry updates and the necessary SubmissionSet and Association objects.

10. The following rules govern what shall not be altered between versions of a DocumentEntry. The Document Registry actor shall validate that these attributes are consistent across versions and reject submissions that violate this rule. The following attributes shall not be altered between versions of a DocumentEntry.

    a) Unique ID

    b) Size

    c) Hash

    d) Logical ID (lid)

11. When a document is submitted as a replacement, using a RPLC Association to an existing document, the new DocumentEntry shall be a first version.

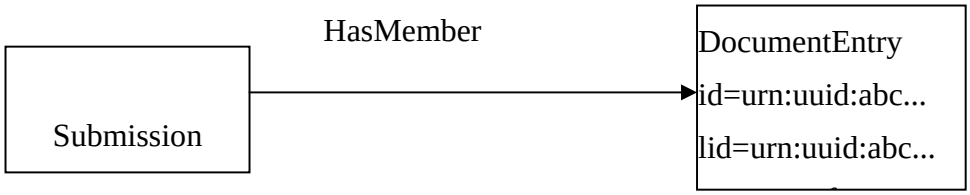
12. When queried via the Query transaction (SQL), metadata is returned in ebRIM 2.1 format. This format does not provide the lid or VersionInfo attributes.

13. The ebRIM 3.0 standard requires that if a registry receives a Registry Object in a submission containing a status attribute that the attribute be ignored. The status attribute is only set by the registry. For XDS, to request a DocumentEntry be deprecated, the Document Source shall

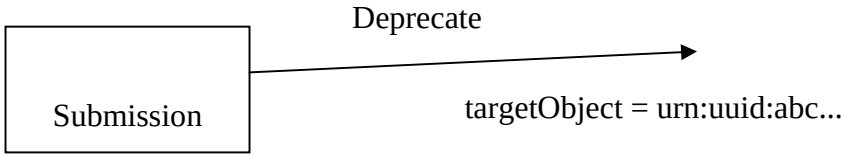
submit a Submission Set and an Association in an Update Document Set transaction [ITI-XX]. The Association shall have an association type of Deprecate (new value) and its targetObject shall reference a DocumentEntry already in the registry. Upon receipt, the Document Registry actor shall change the status on the targeted DocumentEntry to Deprecated.

14. This introduces the new status updating mechanism.

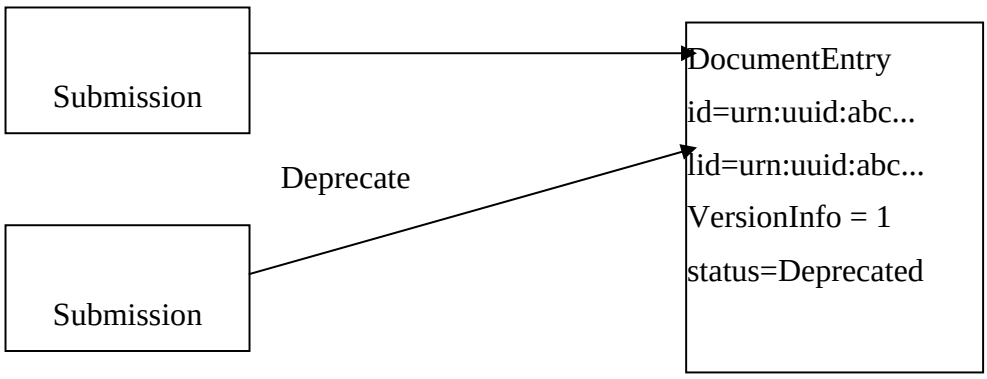
Registry contents
before submission



Submission



Registry Contents
after Submission

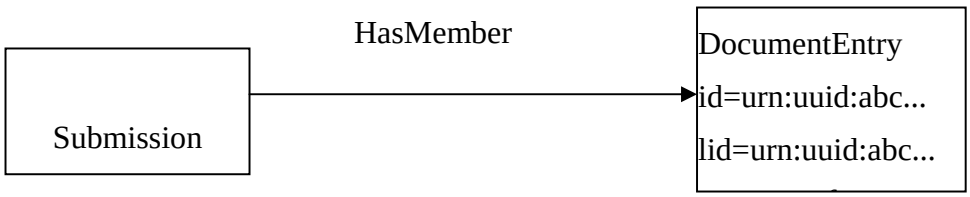


n

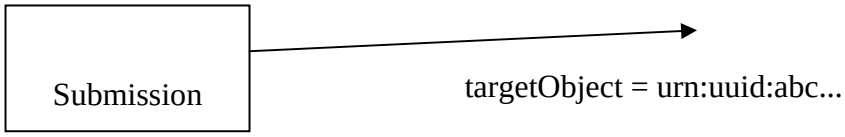
---

15. To label a document as offline, the document still exists but is not accessible, submit a Submission Set and Association. The Association shall have an association type of Offline (new value) and its targetObject shall reference a DocumentEntry already in the registry which represents this document. If multiple versions of metadata (DocumentEntry) for this document exist, the latest version must be the target of the Association. Upon receipt, the Document Registry shall store the status of the document (Offline). It is the Document Registry implementer's choice whether all versions of the DocumentEntry objects are updated or whether a single status is maintained governing all versions the Document (independent of how many DocumentEntry objects exist for the document in the registry). Either way, in response to a Stored Query for any version of this DocumentEntry object, the document status is reported as the value of the (new) documentStatus Slot on the DocumentEntry object. This documentStatus attribute is new and different from the existing DocumentEntry status attribute. The new documentStatus attribute describes the physical availability of the document in the repository. The existing status attribute documents the administrative status from the point of view of the registry. The difference between status=Approved and status=Deprecated in the registry describes the current relevance of the document and not its physical availability. If this documentStatus slot is not present in a Stored Query response its default value is Online. Note that the documentStatus is recorded and reported (Stored Query) independent of the version of the DocumentEntry metadata object. The documentStatus represents the status of the document in the repository and multiple versions of the DocumentEntry (metadata) may reference a single repository document.

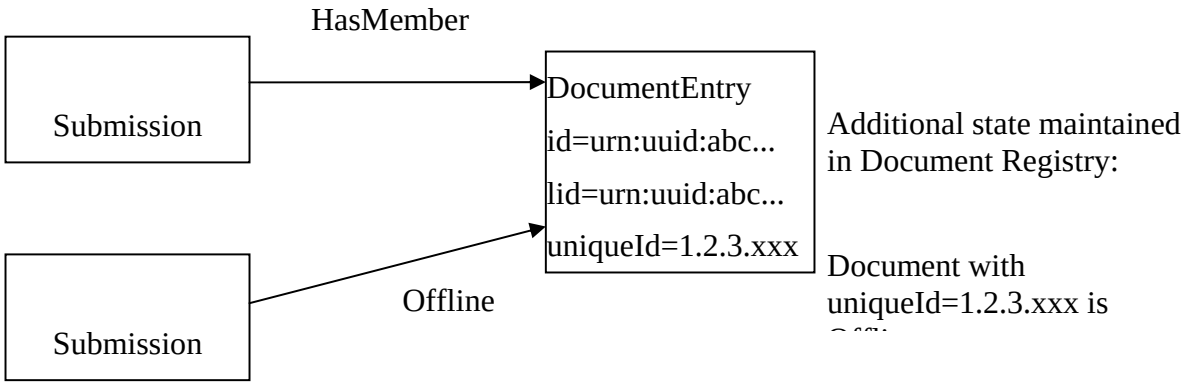
Registry contents
before submission



Submission



Registry Contents
after Submission

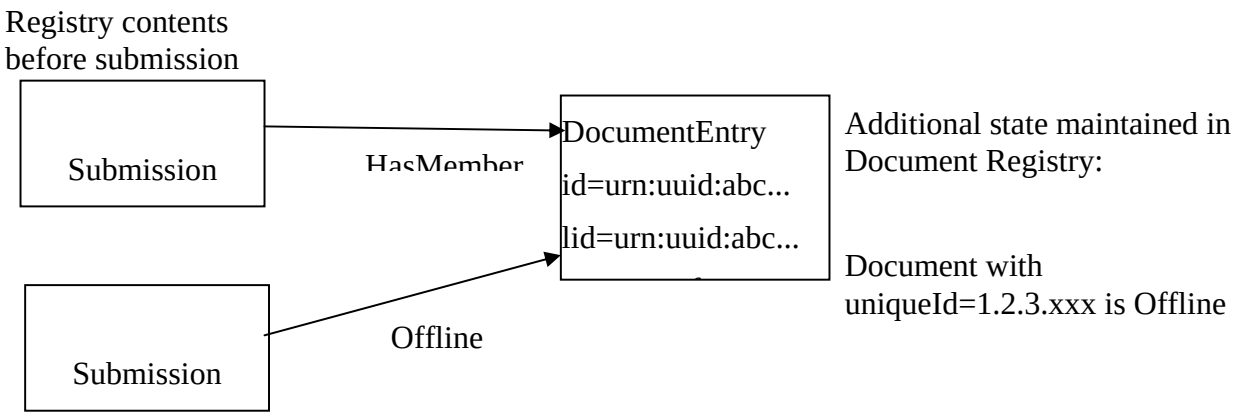


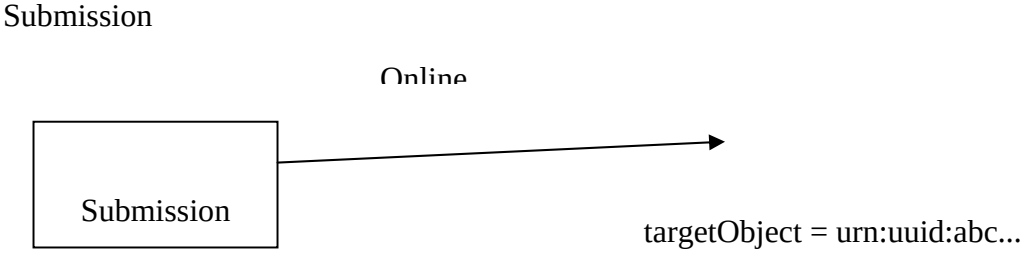
Later Stored Query result:

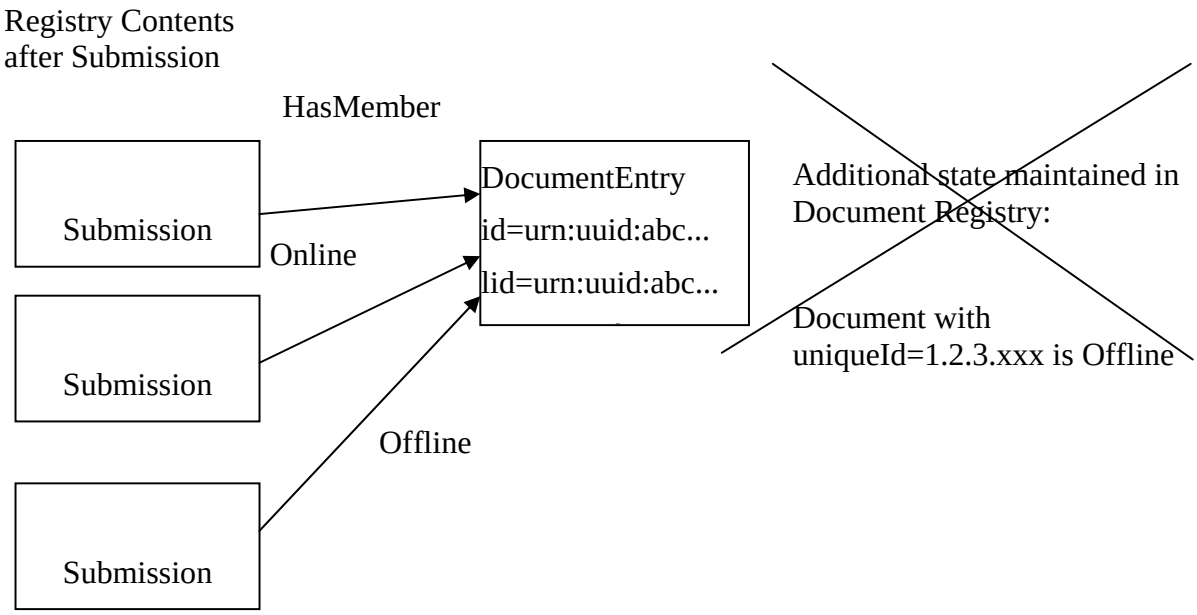
```
<ExtrinsicObject id=”urn:uuid:abc...” lid=”urn:uuid:abc...”>
  <Slot name=”documentStatus”>
    <ValueList>
      <Value>Offline</Value>
    </ValueList>
```

16. If the documentStatus slot is received in a metadata submission, it will be ignored (not saved). The value may only be manipulated through the submission of Online and Offline Associations to the Registry.

1. To label a document as online, documenting its accessibility for retrieval, submit a Submission Set and Association. The Association shall have an association type of Online (new value) and its targetObject shall reference the most recent version of the DocumentEntry already in the registry. The documentStatus attribute held by the Registry for this document is changed to Online. Retrieval of DocumentEntry objects representing this document shall return either documentStatus = Online or no documentStatus slot in the metadata.







To label a Document as deleted, submit a Submission Set and Association. The Association shall have an association type of Delete (new value) and its targetObject shall reference a DocumentEntry already in the registry. Upon receipt, the Document Registry shall change the value of the status attribute of the DocumentEntry to Deleted.

1. DocumentEntry objects with status of Deleted shall not be returned from a Stored Query.

Acceptable operations based on current document status

Document Status	documentStatus slot	Mark Offline	Mark Online	Delete	Update metadata attributes	Replace,Append,Translate
Submitted						
Deprecated	Online	Yes[2]	Yes[2]	Yes[2]	No	No[1]
Deprecated	Offline	Yes[2]	Yes[2]	Yes[2]	No	No[1]
Approved	Online	Yes	Yes	Yes	Yes	Yes
Approved	Offline	Yes	Yes	Yes	Yes	Yes
Deleted[3]	any	No	No	No	No	No

1- No - Prohibited by ebRIM

2- According to ebRS 3.0 new Associations cannot be accepted by the Registry for Deprecated objects. Therefore, for these operations, it can be assumed that the Registry Adaptor momentarily labels the necessary objects as non-Deprecated to allow these updates.

3- No operations are allowed on Deleted DocumentEntries.

7 Web Services Definitions

The new Update Document Set transaction [ITI-XX] uses the ws:Action urn:ihe:iti:2008:UpdateDocumentSet for the request and urn:ihe:iti:2008:UpdateDocumentSetResponse for the response. Both request and response are packaged as SIMPLE SOAP messages. They never carry attachments (documents).

8 Examples

Examples can be found in the online ITI Implementation Guide at http://wiki.ihe.net/index.php?title=Metadata_Versioning_Implementation. The top level page for the Implementation Guide can be found at http://wiki.ihe.net/index.php?title=ITI_Implementation_Guide.

9 Required changes to existing XDS facilities

This section documents the required changes to existing XDS facilities. It does not include the implementation of the XDS Admin Client actor or the update to the Document Registry actor since they represent separate implementation decisions/options. This section is intended as a warning to implementers of effects this specification may have on their existing software even if they do not implement this new facility.

9.1 Changes to the Life-Cycle Management facility

None for the Document Consumer actor. Replacement operates the same, deprecating the old version of the document. For the Document Source and Document Registry actors, only the most recent version of a DocumentEntry can be the target of a Replace/ Transform/ Append operation. This is natural since it will be the only version with status = Approved.

9.2 Added Requirements for Folder Management

When a new version of a DocumentEntry is registered, the Document Registry actor shall add the new version to all folders in which the previous version was a member. This is the same behavior that is specified when a DocumentEntry, as a member of a folder, is replaced. Since the previous version is labeled Deprecated, there will still only be a single Approved version of the document in the folder.

9.3 New Stored Query parameter

A new parameter, $XDSDocumentEntryLid, is added to the GetDocuments Stored Query. This parameter is mutually exclusive with the $XDSDocumentEntryEntryUUID and $XDSDocumentEntryUniqueId parameters (only one one of the three may be specified). When $XDSDocumentEntryLid is specified all matching DocumentEntry objects are returned. Multiple values may be specified for this parameter.

9.4 Changes to Document Source actor implementations

There are no required changes to the Provide and Register transaction. The lid attribute is optional on this transaction.

9.5 Changes to Document Consumer actor implementations

Given that a typical Document Consumer is not interested in documents with status other than Approved, most Document Consumer operations will not be affected.

New Association types will have to be ignored.

For implementations that wish to distinguish metadata versions, sensitivity to the new status codes as well as the lid, and versionInfo metadata is required.

All Document Consumer actor implementations will want to be sensitive to the presence of offline documents. To do so the Document Consumer must understand new attribute documentStatus.

9.6 Changes to Document Repository actor implementations

None

9.7 Changes to Document Registry actor implementations

Document Registry actor implementations which support this function will need to:

- Accept UpdateObjectsRequest
- Manage lid and versionInfo attributes
- Implement new status codes for ExtrinsicObjects
- Implement the new byUid parameter to the GetDocuments Stored Query
- Accept new Association types
- Change status on DocumentEntry objects triggered by submission of new Association types.
- Maintain Online/Offline status of documents independent of how many DocumentEntry versions are present
- Manage document membership in Folders

9.8 Namespace Issues

The new Association types introduced shall have a namespace prefix of:
urn:ihe:iti:2008:AssociationType:

The new status codes introduced shall have the namespace prefix of:
urn:ihe:iti:2008:ResponseStatusType:

10 Use Cases Revisited

These use cases describe basic operations and do not yet address security or audit issues.

10.1 Update Patient Demographics

Patient Demographics and most other attributes of the DocumentEntry object can be updated using the Metadata Versioning functionality.

10.2 Update Confidentiality Code

Confidentiality Codes are coded as Classification objects. An update would allow zero or more new Confidentiality Codes to be added to the ExtrinsicObject and zero or more existing Confidentiality Codes to be removed. Multiple ExtrinsicObjects could be updated in a single request.

10.3 Deprecate Document without Replace

A metadata update is submitted containing a Submission Set and Association of type Deprecate. The existing Approved document is targeted by the Association's targetObject attribute. The registry labels the targeted DocumentEntry with status = Deprecated.

10.4 Off-line Archival of Document Repository Contents

A metadata update is submitted containing a Submission Set and Association of type Offline. The existing document is targeted by the Association's targetObject attribute. The registry updates the DocumentEntry with (new) documentStatus = Offline. In a query response, the documentStatus=Offline indicates that the Repository data exists but is not available for retrieval. No information is given on how to get the document back from off-line archive. A query for status = Approved will see off-line documents. There is currently no way to filter out Offline documents in a Stored Query.

A DocumentEntry is relabeled as Online by submitting a Submission Set and Association of type Online with the targetObject pointing to the DocumentEntry of interest.

10.5 Delete Document

A metadata update is submitted containing a Submission Set and Association of type Delete. The existing DocumentEntry is targeted by the Association's targetObject attribute. The registry labels the targeted ExtrinsicObject with status = Deleted. Stored Queries never return DocumentEntry objects with status of Deleted. There is no mechanism for UnDelete in this specification.

10.6 Change Summary

New Association Types

Association Type	New status attribute of targetObject	New documentStatus attribute of targetObject	Action
Deprecate	Deprecated	No Change	Update status attribute
Offline	No Change	Offline	Update documentStatus in Registry. Return documentStatus attribute with value Offline in Stored Query when asked about any version of the document's DocumentEntry objects
Online	No Change	Online	Remove the Offline status held in the registry for this document.
Delete	Deleted	No Change	Refuse to return DocumentEntry in Stored Query response

605

New Status Codes

Status Code	Meaning
Deleted	DocumentEntry is not longer detectable by a Stored Query

New Slot on DocumentEntry

Slot Name	Meaning
documentStatus	Status of the document in the repository. This slot has a single value which can be Offline or Online. If the slot is not present then its value is Online by default.