Version 0.5

Content Management Interoperability Services

Appendices – Open Issues, Informative Examples & Important Decisions

8/28/2008

CONTENTS

APPENDIX A: OPEN ISSUES

This section lists the set of spec issues that were still under discussion/consideration at the time this version of the spec was published.

Issues are categorized by the spec section to which they apply, and each issue includes a proposed timeframe for resolution. The timeframes are:

- **Next minor version:** This issue SHOULD be resolved in the next minor version of the CMIS specification.
- **CMIS 1.0:** This issue SHOULD be resolved in (or before) the release of version 1.0 of the CMIS specification.
- **After CMIS 1.0:** This issue SHOULD be considered for a future version of the CMIS specification after the publication of CMIS 1.0 .

Note that issues may be "resolved" by changes to the specification, or by the CMIS TC voting NOT to modify the spec.

PART I – DATA MODEL AND SERVICES

Note: The issues included in this section MAY have a cascading impact onto the CMIS bindings listed in Part II of the specification.

FINAL URI FOR CMIS SCHEMAS

Timeframe: Next minor version

The CMIS specification currently lists "cmis.org" as the domain where the CMIS XML schemas will be available for use. However, this domain is unavailable.

This URI needs to be updated to refer to the location where the official CMIS XML schemas will reside.

DISCOVERING REPOSITORY LOCATIONS AND CONNECTION SETTINGS

Timeframe: After CMIS 1.0

The CMIS specification currently does NOT explicitly specify how an application should discover:

a) The URL/service endpoint where a CMIS repository can be located, or
b) The connection/authentication modes & settings to use for connecting to the repository

Instead, it is assumed that the application will acquire this knowledge in a way that is out-of-scope for CMIS.

It would potentially be useful to extend the CMIS specification to include an explicitly defined mechanism for discovering how to connect to a CMIS repository and/or how to infer the location of a CMIS service endpoint from a well-known URL or directory.

SUPPORT FOR FETCHING AN ARBITRARY COLLECTION OF OBJECTS BY OBJECTID

Timeframe: CMIS 1.0

The CMIS specification currently states that it is repository-specific whether the “ObjectId” property that is exposed on all four base object types is “queryable” (i.e. can be used in the WHERE clause of a query).

This means that there’s no guarantee that an application will be able to fetch the properties for an arbitrary collections of objects via their ObjectIds from *any* repository – only those repositories that have specified that ObjectIds are queryable.

There are several possible approaches to address this, including:

- Specifying that the “ObjectId” property MUST be queryable for all repositories
- Adding a new service for “getPropertiesForMultipleObjects” that takes as input the IDs for multiple objects.

APPENDIX B: INFORMATIVE EXAMPLES

The following sections provide informative examples of various elements of the CMIS specification.

VERSIONING EXAMPLE

The following example illustrates how the versioning of Document objects is supported in CMIS, specifically with respect to the following properties of each Document object:

Property Name	*Property Type*
OBJECT_ID	ID
IS_LATEST_VERSION	Boolean
IS_MAJOR_VERSION	Boolean
IS_LATEST_MAJOR_VERSION	Boolean
VERSION_SERIES_IS_CHECKED_OUT	Boolean
VERSION_SERIES_CHECKED_OUT_BY	String
VERSION_SERIES_CHECKED_OUT_ID	ID
CHECKIN_COMMENT	String

Note: All ObjectIDs shown in this example scenario are arbitrary and purely for illustrative purposes.

INITIAL STATE OF REPOSITORY/DOCUMENT

In this example, we will consider a repository containing only a single Document object that has been created using the CMIS *createDocument()* service with the following parameters:

- **versioningState:** CheckedInMinor.

As a result, the Document was created, assuming with a repository-assigned ObjectID of 10. Invoking the *getProperties()* service on that document would return the following values for these properties:

Property Name	*Property Value*
OBJECT_ID	*10*
IS_LATEST_VERSION	*True*
IS_MAJOR_VERSION	*False*
IS_LATEST_MAJOR_VERSION	*False*
VERSION_SERIES_IS_CHECKED_OUT	*False*
VERSION_SERIES_CHECKED_OUT_BY	*Value not set*
VERSION_SERIES_CHECKED_OUT_ID	*Value not set*
CHECKIN_COMMENT	*Value not set*

Note: *If the* **versioningState** *parameter above was instead set to "*CheckedInMajor*", then the* IS_MAJOR_VERSION *and* IS_LATEST_MAJOR_VERSION *properties would be "*True*", rather than "*False*".*

USER ACTION #1: CHECKOUT DOCUMENT 10

The user now invokes the *checkout()* service for Document 10.

The *checkout()* service will then return the ObjectID of the "Private Working Copy" representing the checked-out copy of Document 10. In this example, we will assume that the ObjectID of the "Private Working Copy" is 11. As a result, there are now two Document objects in the Repository.

getAllVersions: Calling *getAllVersions()* for Document 10 or Document 11 (assuming that the user has sufficient rights to see both Documents 10 & 11) would return an ordered collection containing both Document 10 & Document 11.

getProperties: Calling *getProperties()* for either Document at this point would show the following properties:

getProperties(10):

Property Name	*Property Value*
OBJECT_ID	10
IS_LATEST_VERSION	True
IS_MAJOR_VERSION	False
IS_LATEST_MAJOR_VERSION	False
VERSION_SERIES_IS_CHECKED_OUT	True
VERSION_SERIES_CHECKED_OUT_BY	(check-out user id)
VERSION_SERIES_CHECKED_OUT_ID	11
CHECKIN_COMMENT	Value not set

getProperties(11):

Property Name	*Property Value*
OBJECT_ID	11
IS_LATEST_VERSION	False
IS_MAJOR_VERSION	False
IS_LATEST_MAJOR_VERSION	False
VERSION_SERIES_IS_CHECKED_OUT	True
VERSION_SERIES_CHECKED_OUT_BY	(check-out user id)
VERSION_SERIES_CHECKED_OUT_ID	11
CHECKIN_COMMENT	Value not set

Notes:

- If the user invoking the *getProperties()* service did not have sufficient rights on Document 11 (where the meaning of "sufficient rights" is determined by the repository), then the repository could return "Value not set" for VERSION_SERIES_CHECKED_OUT_ID (but would still return VERSION_SERIES_IS_CHECKED_OUT == "True").
- Note that IS_LATEST_VERSION is still "True" for Document 10. By definition this flag indicates that the Document is the latest "checked-in" version of the Document's Version Series.

USER ACTION #2: CHECKIN DOCUMENT 11

The user now invokes the *checkin()* service for Document 11, with the following parameters:

- **Major:** True
- **CheckinComment:** "Version 1.0"

The *checkin()* service will then return the ObjectID of the checked-in version of that Document object, which may or may not be the same as the ObjectID of the Private Working Copy. In this example, the service returns an ObjectID of 12 for the checked-in Document, indicating that the Private Working Copy (Document 11) no longer exists in the Repository.

getAllVersions: Calling *getAllVersions()* for Document 10 or Document 12 (assuming that the user has sufficient rights to see both Documents 10 & 12) would return an ordered collection containing both Document 10 & Document 12.

getProperties: Calling *getProperties()* for either Document at this point would show the following properties:

getProperties(10):

Property Name	*Property Value*
OBJECT_ID	*10*
IS_LATEST_VERSION	*False*
IS_MAJOR_VERSION	*False*
IS_LATEST_MAJOR_VERSION	*False*
VERSION_SERIES_IS_CHECKED_OUT	*False*
VERSION_SERIES_CHECKED_OUT_BY	*Value not set*
VERSION_SERIES_CHECKED_OUT_ID	*Value not set*
CHECKIN_COMMENT	*Value not set*

getProperties(12):

Property Name	*Property Value*
OBJECT_ID	*12*
IS_LATEST_VERSION	*True*
IS_MAJOR_VERSION	*True*
IS_LATEST_MAJOR_VERSION	*True*
VERSION_SERIES_IS_CHECKED_OUT	*False*
VERSION_SERIES_CHECKED_OUT_BY	*Value not set*
VERSION_SERIES_CHECKED_OUT_ID	*Value not set*
CHECKIN_COMMENT	*"Version 1.0"*

USER ACTION #3: CHECKOUT DOCUMENT 10 AGAIN

Note: It is repository-specific whether a user can check out ONLY the latest version of a Document in a Version Series, or ANY Document in the Version Series. (Repositories can expose whether or not checking out any particular Document is allowed using the *getAllowableActions*() service.) For the purposes of this example, we will assume that the Repository supports checking out ANY Document in the Version Series.

The user now invokes the *checkout*() service for Document 10, again.

The *checkout()* service will then return the ObjectID of the "Private Working Copy" representing the checked-out copy of Document 10. In this example, we will assume that the ObjectID of the "Private Working Copy" is 13. As a result, there are now three Document objects in the Repository.

getAllVersions: Calling *getAllVersions()* for Document 10, 12, or 13 (assuming that the user has sufficient rights to see those Documents) would return an ordered collection containing Documents 10, 12, and 13 (in that order).

getProperties: Calling *getProperties*() for any of the Documents at this point would show the following properties:

getProperties(10):

Property Name	*Property Value*
OBJECT_ID	*10*
IS_LATEST_VERSION	*False*
IS_MAJOR_VERSION	*False*
IS_LATEST_MAJOR_VERSION	*False*
VERSION_SERIES_IS_CHECKED_OUT	*True*
VERSION_SERIES_CHECKED_OUT_BY	*(check-out user id)*
VERSION_SERIES_CHECKED_OUT_ID	*13*
CHECKIN_COMMENT	*Value not set*

getProperties(12):

Property Name	*Property Value*
OBJECT_ID	*12*
IS_LATEST_VERSION	*True*
IS_MAJOR_VERSION	*True*
IS_LATEST_MAJOR_VERSION	*True*
VERSION_SERIES_IS_CHECKED_OUT	*True*
VERSION_SERIES_CHECKED_OUT_BY	*(check-out user id)*
VERSION_SERIES_CHECKED_OUT_ID	*13*
CHECKIN_COMMENT	*"Version 1.0"*

getProperties(13):

Property Name	*Property Value*
OBJECT_ID	*13*
IS_LATEST_VERSION	*False*
IS_MAJOR_VERSION	*False*
IS_LATEST_MAJOR_VERSION	*False*
VERSION_SERIES_IS_CHECKED_OUT	*True*
VERSION_SERIES_CHECKED_OUT_BY	*(check-out user id)*
VERSION_SERIES_CHECKED_OUT_ID	*13*
CHECKIN_COMMENT	*Value not set*

Note: As indicated in the CMIS Data Model, the Version Series collection is one-dimensional, i.e. NOT a graph. As a result, there is no indication that Document 13 was created by checking out Document 10 – Document 13 is simply the current Private Working Copy for the Version Series that includes Documents 10 and 12.

USER ACTION #4: CHECKIN DOCUMENT 13

The user now invokes the *checkin*() service for Document 13, with the following parameters:

- **Major:** False

The *checkin*() service will then return the ObjectID of the checked-in version of that Document object, which may or may not be the same as the ObjectID of the Private Working Copy. In this example, the service returns an ObjectID of 14 for the checked-in Document, indicating that the Private Working Copy (Document 13) no longer exists in the Repository.

getAllVersions: Calling *getAllVersions*() for Document 10, 12, or 14 (assuming that the user has sufficient rights to see all the Documents) would return an ordered collection containing Documents 10, 12, and 14 (in that order).

getProperties: Calling *getProperties*() for any Document at this point would show the following properties:

getProperties(10):

Property Name	*Property Value*
OBJECT_ID	*10*
IS_LATEST_VERSION	*False*
IS_MAJOR_VERSION	*False*
IS_LATEST_MAJOR_VERSION	*False*
VERSION_SERIES_IS_CHECKED_OUT	*False*
VERSION_SERIES_CHECKED_OUT_BY	*Value not set*
VERSION_SERIES_CHECKED_OUT_ID	*Value not set*
CHECKIN_COMMENT	*Value not set*

getProperties(12):

Property Name	*Property Value*
OBJECT_ID	*12*
IS_LATEST_VERSION	*False*
IS_MAJOR_VERSION	*True*
IS_LATEST_MAJOR_VERSION	*True*
VERSION_SERIES_IS_CHECKED_OUT	*False*
VERSION_SERIES_CHECKED_OUT_BY	*Value not set*
VERSION_SERIES_CHECKED_OUT_ID	*Value not set*
CHECKIN_COMMENT	*"Version 1.0"*

getProperties(14):

Property Name	*Property Value*
OBJECT_ID	*14*
IS_LATEST_VERSION	*True*
IS_MAJOR_VERSION	*False*
IS_LATEST_MAJOR_VERSION	*False*
VERSION_SERIES_IS_CHECKED_OUT	*False*
VERSION_SERIES_CHECKED_OUT_BY	*Value not set*
VERSION_SERIES_CHECKED_OUT_ID	*Value not set*
CHECKIN_COMMENT	*Value not set*

APPENDIX C: IMPORTANT DECISIONS

The following sections summarize the rationale for various important decisions made in designing the CMIS specification. This section is intended to inform both implementers and future specification authors about decisions made in creating CMIS version 1.0.

MULTIPLE CONTENT STREAMS FOR DOCUMENT OBJECTS

One request that we've received as part of the feedback on version 0.1 of the CMIS spec is to include the ability to have Document objects that can have more than one content stream (i.e. file).

While it's not necessarily the case that we feel this is a required capability for version 1.0 of the CMIS specification, it is worth consideration despite the additional complexity it brings.

This addendum describes a high-level proposal for extending the CMIS domain model & services to enable Document objects that can have 0, 1, or more than one content stream. (Note: This addendum does NOT include the corresponding changes that would then cascade out to each of the protocol bindings.)

The intent of this addendum is to be input into further investigation/discussion by future authors revising the specification, so that that group can consider including these changes in future versions of the CMIS specification.

TARGET SCENARIOS & USE CASES

MULTI-PAGE IMAGING/ARCHIVAL APPLICATIONS:

Many business applications produce high volumes of documents per day, often in a print format such as PDF or PostScript. These documents need to be captured from other business systems, indexed, and stored into archival repository such that they can be easily retrieved by end users as needed for business functions.

In many cases the documents being imaged/archived are large, and consisting of many pages. Often the optimal way for a repository to store these documents is with a single overall "document" object representing the object, that contains multiple "page" files. This allows the application to maintain a single set of semantic metadata that describes the overall document (e.g. customer ID, archival date, etc.), while allowing users of the images to be able to retrieve only the pages they need when they need them, rather than having to download a potentially large file representing the entire document.

CONTENT RENDITIONING/PUBLISHING

In many ECM applications, a customer will want to store multiple formats of a single document as a logical unit within the repository that stores multiple formats/renditions of the document for use by various audiences. (For example, a Microsoft Word document for internal users, a PDF rendition for distribution outside the organization, and an HTML version for use on a web site.)

Again, in these cases the optimal storage model in the repository is to have a single logical "document" that stores the semantic metadata about the document, along with multiple content streams for each format.

GOALS

- Provide an additive extension to the existing CMIS domain model for repositories that can store multiple content streams per document.
- This extension will be an optional capability.
- This extension will NOT add complexity to repositories wishing to implement CMIS that support only one content stream per document *or* to applications wishing to work with CMIS repositories but will only ever need to store one content stream per document.

EXTENSIONS TO THE CMIS SPECIFICATION

In order to enable the scenarios described above the following additions to the CMIS domain model are proposed:

DATA MODEL

CHANGES TO "REPOSITORY" OBJECT

We will add a new repository-specific capability flag, called "SupportsAdditionalContentStreams".

NEW BASE OBJECT "ADDITIONAL CONTENT STREAM"

We will create a new base object called "AdditionalContentStream". The usage of this object, as its name implies is to represent an additional content stream (other than the primary content stream) stored as part of a document object.

The Additional Content Stream object type is by definition not an independent object, meaning:

- It does not have a separate ObjectID from the Document object to which it belongs (more on how it is identified below)
- It can not be moved between Documents
- The properties of the AdditionalContentStream object cannot be queried outside the scope of the Document object. (More on how querying for additional content streams via the Document object will work below).
- It does not have a formal CMIS object type, which could then be inherited/specialized. The schema for an AdditionalContentStream is always exactly as defined below.

***Note**:* The choice of a user-settable name (rather than a system-generated one) is to avoid having to define a more formal/extensible object type for the additional content streams (which would be required if we took the "system-generated" name approach – so that applications could maintain some information about the meaning of each content stream.) This approach is potentially simpler in that it gives applications a place to store info about what the stream means, while minimizing the complexity addition to our domain model. Also, it avoids potential application confusion around figuring out when they want properties on a content stream vs. on a document.

Here are the properties of the AdditionalContentStream object:

Property Name	*Display Name*	*Desc.*	*Property Type*	*S-/M-Value*	*Max Lngth*	*Choice*	*Req'd*	*Dfult Value*	*Upd*	*Query*	*Ord*
NAME	Name	*RS*	String	S	*RS*	*RS*	*RS*	*RS*	RO *	N*	N
URI	URI	*RS*	URI	*S*	*RS*	*na*	*RS*	*na*	RO	*RS*	*RS*
CONTENT_STREAM_LENGTH	Content Stream Length	*RS*	Integer	M	*RS*	*Na*	*RS*	*na*	RO	*RS*	*RS*
CONTENT_STREAM_MIME_TYPE	Content Stream MIME Type	*RS*	String	M	*RS*	*RS*	*RS*	*RS*	RO	*RS*	*RS*

RS: Repository-specific
na: Not applicable

*: While the name property of an AdditionalContentStream is not directly update-able or query-able, it is set indirectly (when the object is created) and queried indirectly (as a projection onto the Document object).

- **Name:** This property, which is user-settable only when the additional content stream is created, is the string that will be used to uniquely refer to one of the additional content streams for the Document object.
    - The name of each additional content stream is persisted by the repository throughout the content stream's lifecycle.
    - It is required that the repository ensure that the name of each Additional Content Stream is unique within the AdditionalContentStreamCollection for that Document.
- **URI:** The URI to the content stream.
- **Length:** The length of the content stream.
- **MIME Type:** The MIME type of the Content Stream.

Note about immutability: It was a deliberate choice NOT to include an "immutable" property on each additional content stream. The CMIS version 0.2 spec authors suggest that the right design here is to have immutability controlled at the Document level (i.e. for all content streams atomically) rather than per content stream.

NEW BASE OBJECT: ADDITIONALCONTENTSTREAMCOLLECTION

This is an un-ordered collection of AdditionalContentStream objects. It will be returned as part of operations that retrieve the additional content streams for a Document object.

CHANGES TO "DOCUMENT" OBJECT

We will add the following additional properties to the Document object. These properties will always be not set for repositories that do not support the "AdditionalContentStream" capability.

Property Name	*Display Name*	*Desc.*	*Property Type*	*S-/M-Value*	*Max Lngth*	*Choice*	*Req'd*	*Dfult Value*	*Upd*	*Query*	*Ord*
CONTENT_STREAM_NAME	Content Stream Name	*RS*	String	S	*RS*	*RS*	*RS*	*RS*	RS	RS	RS

ADDITIONAL_CONTENT_STREAM_NAMES	Additional Content Stream Names	*RS*	String	*M*	*RS*	*N*	*N*	*na*	RO	*RS*	*RS*
ADDITIONAL_CONTENT_STREAM_LENGTHS	Additional Content Stream Lengths	*RS*	Integer	M	*RS*	*Na*	*N*	*na*	RO	*RS*	*RS*
ADDITIONAL_CONTENT_STREAM_MIME_TYPES	Additional Content Stream MIME Types	*RS*	String	M	*RS*	*RS*	*N*	*na*	RO	*RS*	*RS*
ADDITIONAL_CONTENT_STREAM_URIS	Additional Content Stream URIs	*RS*	URI	M	*RS*	*na*	*Na*	*na*	RO	*RS*	*RS*

RS: Repository-specific
na: Not applicable

*: While the name property of an AdditionalContentStream is not directly update-able or query-able, it is set indirectly (when the object is created) and queried indirectly (as a projection onto the Document object).

- **Content Stream Name:** This is a read-write property that is used to identify the *primary* content stream for the Document Object. (I.e. if the Document object includes multiple content streams, then the application will want to have a name to identify the main content stream.)
    - It is repository-specific whether a name is required even when the Document does not include additional content streams.
        - It is recommended that:
            - Repositories that support the additional content stream capability always require a primary content stream name for the primary stream.
            - Repositories that do NOT support the capability do not require a primary content stream name.
- **Additional Content Stream Names:** This is a read-only system property that provides an ordered list of the names of the additional content streams currently stored for this Document. It is in effect a projection of all of the single-valued "name" properties for each "AdditionalContentStream" properties, to facilitate querying.
    - It will always be not set if the repository does not support the additional content stream capability.
    - The list is **not** ordered, but it is a requirement that *all* of the additional content stream multi-valued properties are ordered consistently for a given request.
        - Example:
            - if a caller fetches the properties for a given Document with 3 additional content streams, then the additional content stream whose name is at position 2 within the "additional content stream names" property should have its corresponding length at position 2 within the "content stream lengths" property.
            - Furthermore, there is no requirement that subsequent requests to get the properties of the same Document will return the content stream properties in the same order.
        - However, it is NOT required that the order in which the additional content streams are presented be persistent across requests. (I.e. in the previous example, if a 4th additional

content stream is then added, that additional stream's properties may appear at any of the 4 positions of the array.)

- **Additional Content Stream Lengths:** Analogous to the previous "names" property, this read-only system property provides an ordered list of the lengths of the additional content streams currently stored for this document.
    - It will always be not set if the repository does not support the additional content stream capability.
    - As noted above, it is a requirement that *all* of the additional content stream multi-valued properties are ordered consistently for a given request.
- **Additional Content Stream MIME Types:** Analogous to the previous "names" property, this read-only system property provides an ordered list of the MIME types of the additional content streams currently stored for this document.
    - It will always be not set if the repository does not support the additional content stream capability.
    - As noted above, it is a requirement that *all* of the additional content stream multi-valued properties are ordered consistently for a given request.
- **Additional Content Stream URIs:** Analogous to the previous "names" property, this read-only system property provides an ordered list of the MIME types of the additional content streams currently stored for this document.
    - It will always be not set if the repository does not support the additional content stream capability.
    - As noted above, it is a requirement that *all* of the additional content stream multi-valued properties are ordered consistently for a given request.

CHANGES TO VERSIONING

AdditionalContentStream objects are not versioned *independently* of the Document objects to which they belong. However, AdditionalContentStream objects are implicitly versioned along with the Document objects. (I.e. each additional content stream is specifically bound to a version of the Document object.)

CHANGES TO QUERY

The changes to the data model identified above require no changes to the existing CMIS query model. But the following behaviors are worth noting:

- The CONTAINS function is still scoped to the Document object. This means that the function will return true if ANY of the Document's primary or additional content streams contain the specified parameters.
- The SCORE function is still scoped to the Document object. That the scalar value returned by the function will need to incorporate all of the Document's primary and additional content streams.
- If a consumer wishes to query a CMIS repository to find out whether a given predicate condition is met for any of the Document's content streams, they will need to include clauses that check the single-valued "primary content stream" property as well as the multi-valued "alternate content stream" property.
    - Example: For a query trying to find all Documents that have at least one content stream of MIME_TYPE "MSWORD":
        - SELECT *
          FROM DOCUMENT
          WHERE ((CONTENT_STREAM_MIME_TYPE = 'MSWORD') OR
                (ANY ADDITIONAL_CONTENT_STREAM_MIME_TYPES IN ('MSWORD')))

- It is a known limitation of this model that a user can NOT issue a query that checks for a correlation of multiple properties for a single additional content stream. (E.g. find me all Documents that have a content stream of type MSWORD AND is bigger than 10MB.)

CHANGES TO SERVICES

The following changes will be made to the CMIS services listed below.

REPOSITORY SERVICES

GETREPOSITORYINFO METHOD

Outputs: NEW capability: AdditionalContentStreams

OBJECT SERVICES

GETCONTENTSTREAM METHOD

Inputs: NEW (optional) contentStreamName: The name of the content stream to be returned for the document.

Notes:

- If no contentStreamName is provided, then the primary content stream will be returned.
- If an invalid contentStreamName is provided, then the repository will throw an InvalidArgumentException.

SETCONTENTSTREAM METHOD

Inputs: NEW (optional) contentStreamName: The name of the content stream to be written.

Notes:

- If no contentStreamName is provided, then the primary content stream will be used.
- If an invalid contentStreamName is provided, then the repository will throw an InvalidArgumentException.

ADDCONTENTSTREAM METHOD (NEW)

This is an entirely new method, so its full table is below:

Description	Creates a new "AdditionalContentStream" for the specified Document object.
Inputs	• ID documented • String additionalStreamName • ContentStreamcontentStream
Outputs	
Exceptions	• Common Exceptions • InvalidArgumentException (if the additionalStreamName is already in use for the specified Document) • StorageException • ConstraintViolationException • AlreadyExistsException • ObjectNotFoundException • StreamNotSupportedException
Notes	• This is considered an update of the document. • As mentioned in the data model, the additionalStreamName must not already be in use by another content stream for the specified Document.
Questions	

DELETECONTENTSTREAM

Inputs: NEW (optional) contentStreamName: The name of the content stream to be deleted.

Notes:

- If no contentStreamName is provided, then the primary content stream will be deleted.
- If an invalid contentStreamName is provided, then the repository will throw an InvalidArgumentException.
- It is repository specific what happens when the primary content stream for a Document is deleted, but the Document has additional content streams. (E.g. whether this is allowed, whether one of the "additional" content streams now becomes the primary or not, etc.)

VERSIONING SERVICES

CHECKOUT

Notes: While it is repository-specific whether or not the content of that Document is copied into the private working copy, repositories that support additional content streams MUST copy in both the primary and ALL additional content streams for the Document into the Private Working Copy if they return the “contentCopied” flag on the “checkout” service.