OASIS ebXML Registry

Proposal: Content-based Discovery

Category: New feature

Date: August 15, 2002

Version 0.2

Authors: Farrukh Najmi, Nikola Stojanovic, Matt MacKenzie

Table of Contents

Status of this Document

This note describes the initial proposal for the `Content-based Discovery` work item for OASIS ebXML Registry V3.0. It is expected that the Query sub-team of the OASIS ebXML Registry TC will improve upon this initial proposal and then submit it for consideration by ebXML Registry TC at large.

1 Abstract

This document proposes a new feature being added to the query capabilities of the OASIS ebXML Regis try targeted for version 3.0. The `Content-based Discovery` feature enables the registry to handle queries that can predicate their results on the content defined by the repository items in addition to predicating on the metadata defined by the RegistryObjects. The feature enables client to discover repository items based upon specific criteria matching their content.

2 Motivation

The following motivations drive this proposal:

1. Radically improve the discovery capabilities of the OASIS ebXML Registry.
2. Enable typical business use-cases that require discovery of content based upon the data within the content.

3 Assumptions

The following assumptions are made in this proposal:

1. The design center of the proposal will focus on content-based discovery of repository items that are XML documents.

The proposal will allow for supporting content-based discovery on other types of content besides XML documents.

4 Design Goals

The following design goals have been pursued in this proposal:

Require no new interfaces to allow this feature to be implemented in a V2.x registry.

5 External Dependencies

This proposal depends upon the following external artifacts and event:

- No external dependencies

6 Use Cases

There are many scenarios where content-based discovery is necessary.

6.1 Find All CPPs Where Role is “Buyer”

A company that sells a product using the RosettaNet PIP3A4 Purchase Order process wants to find CPPs for other companies where the Role element of the CPP is that of “Buyer”.

6.2 Find All XML Schema’s That Use Specified Namespace

A client may wish to discover all XML Schema documents in the registry that use an XML namespace containing the word “oasis”.

6.3 Find All WSDL Descriptions with a SOAP Binding

An ebXML registry client is attempting to discover all repository items that are WSDL descriptions that have a SOAP binding defined. Note that SOAP binding related information is content within the WSDL document and not metadata.

7 Content-based Discovery

```
[Note]The following will be a chapter in ebRS
     immediately following Chapter 8 on Queries.
```

This chapter describes the Content-based discovery facility of the ebXML Registry. This facility enables clients to discover repository items based upon the content contained within the repository item.

The essence of the content-based discovery features is based upon the ability to selectively convert repository content into RegistryObject metadata.

A registry uses one or more content indexing services to automatically index repository items when they are submitted to the registry. Indexing a repository item creates RegistryObject metadata such as Classification instances. The indexed metadata enables clients to discover the repository item using existing query capabilities of the registry.

```
[Note]The term index is used to refer to
     RegistryObject Metadata generated from
     selective repository item content. It should
     not be confused with databases indexes. It is
     named such because it is similar in concept to
     database indexes, which are metadata generated
     from content.
```

7.1 Content Indexing Service

Figure 1 shows that conceptually, a content indexing service (or indexer) accepts as input a repository item and generates as output one or more Classification instances that are used to classify the ExtrinsicObject for that repository item. In addition an indexer accepts as control input an index definition file, which is also a repository item.

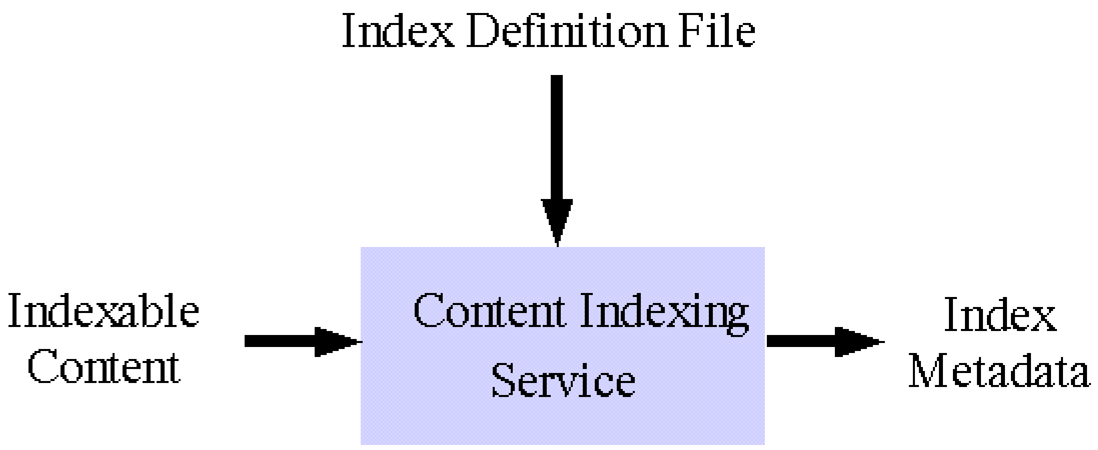


Figure 1: Abstract Content Indexing Service: Inputs and Outputs

7.1.1 Illustrative Example

Figure 2 shows a UML instance diagram to illustrate how a Content Indexing Service is used. The content indexing service is the normative Default XML Indexing Service described in section 7.9.

- o In the center we see a Content Indexing Service name defaultXMLIndexer.
- o On the left side we see a CPP repository item and its ExtrinsicObject inputExtObjForCPP being input as Indexable Content to the defaultXMLIndexer.
- o On top we see an XSLT style sheet repository item and its ExtrinsicObject being sent as an Index Definition File to the defaultXMLIndexer.
- o On the right we see the outputExtObjForCPP, which is the modified ExtrinsicObject for the CPP. We also see a Classification roleClassification, which classifies the CPP by the Role ele ment within the CPP. These are the Index Metadata generated as a result of the indexer indexing the CPP.

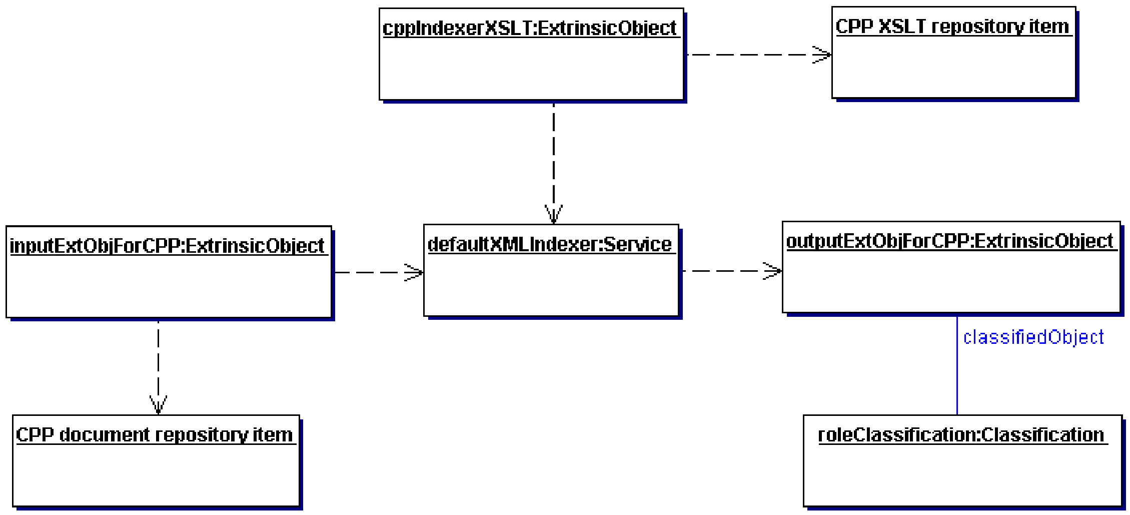

cppIndexerXSLT:ExtrinsicObject
CPP XSLT repository item
inputExtObjForCPP:ExtrinsicObject
defaultXMLIndexer:Service
outputExtObjForCPP:ExtrinsicObject
classifiedObject
CPP document repository item
roleClassification:Classification

Figure 2: Example of CPP indexing using Default XML Indexer

7.2 Index Definition File

The Index Definition File describes what information should the indexer extract from the repository item and subsequently map it to the generated Classification(s). This specification does not define the format of the Index Definition File. Each indexer is free to define its own Index Definition File format in an indexer specific manner. The only constraint in this specification is that the index definition file must be a repository item.

7.3 Index-able Content

The index-able content is the content that the client wishes to be indexed by the Content Indexing Service. As such it is the subject of the content indexing action.

This specification does not define the format of index-able content. This specification describes how a client may register arbitrary indexers for indexing arbitrary content types.

The most common use case for an indexer is for indexing XML documents. Therefore, this specification also provides a normative definition for a specialized XML Content Indexer in section 7.9.

An ebXML Registry must provide native built-in support for the normative XML Content Indexer.

In addition, an ebXML Registry must allow clients to register arbitrary indexers for arbitrary content. In either case the registry must use the appropriate indexer if one exists, to index a repository item when it is submitted.

7.4 Index Metadata

A content indexing service indexes a repository item by processing it and extracting specific information content as specified by the Index Definition File. The content indexing service must map the extracted content to index metadata in form of instances of RIM classes.

For example, the index metadata may consists of:
Classification instances

- ExternalIdentifier instances
- ExternalLink instances
- The name attribute for the ExtrinsicObject for the index-able content
- The description attribute for the ExtrinsicObject for the able-able content

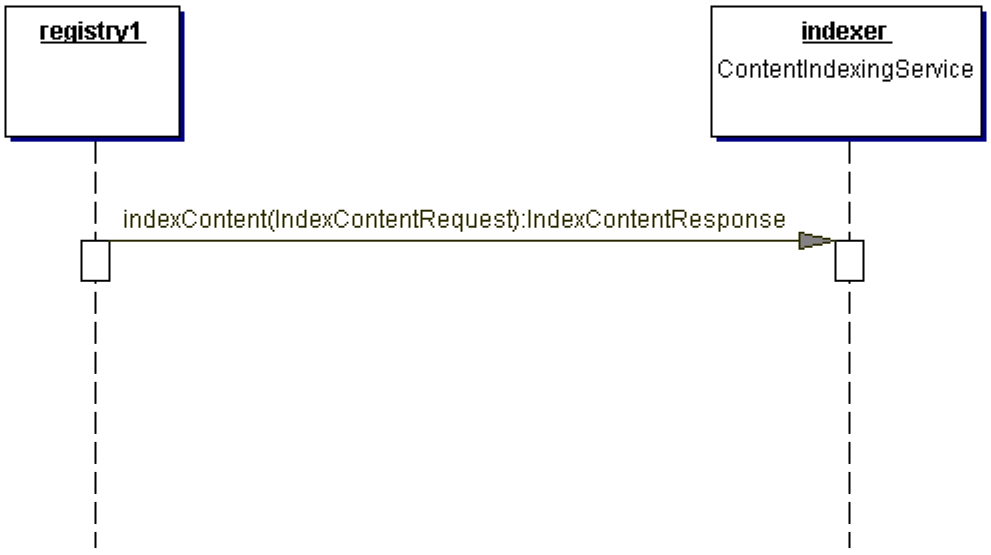
A content indexing service is free to generate any class defined by RIM as index metadata in an application specific manner.

7.5 Content Indexing Protocol

The interface of the content indexing service must implement a single method called indexContent. The indexContent method accepts an IndexContentRequest as parameter and returns an IndexContentResponse as its response if there are no errors.

The IndexContentRequest contains repository items that need to be indexed. The resulting IndexContentResponse contains the metadata that gets generated by the Content Indexing Service as a result of indexing the specified repository items.

The content indexing protocol is abstract and does not specify the interface or behavior of any specific Content Indexing Service.



☞☞**Figure 3: Content Indexing Protocol**

7.5.1 IndexContentRequest

The IndexContentRequest is used to submit repository items to a Content Indexing Service so that it can create index metadata for the specified repository items.

7.5.1.1 Syntax:

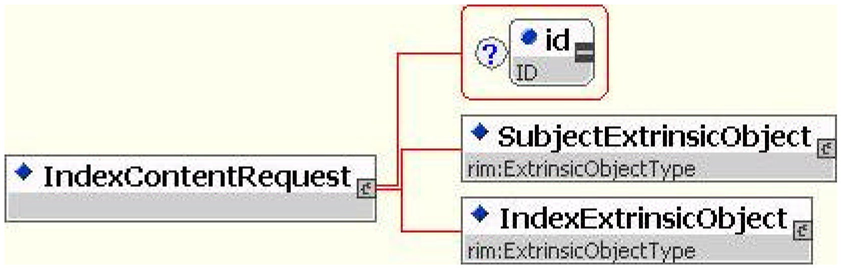


Figure 4: IndexContentRequest Syntax

7.5.1.2 Parameters:

- *id*: Inherited request id attribute common to all requests.
- *IndexExtrinsicObject*: This parameter specifies the ExtrinsicObject for the repository item that the caller wishes to specify as the Index Definition file. This specification does not specify the format of this repository item. There must a corresponding repository item as an attachment to this request. The corresponding repository item should follow the same rules as attachments in SubmitObjectsRequest.
- *SubjectExtrinsicObject:* This parameter specifies the ExtrinsicObject for the repository item that the caller wishes to be indexed. This specification does not specify the format of this repository item. There must a corresponding repository item as an attachment to this request. The corresponding repository item should follow the same rules as attachments in SubmitObjectsRequest.

7.5.1.3 Returns:

This request returns an IndexContentResponse upon success. See section 7.5.2 for details.

7.5.1.4 Exceptions:

In addition to the exceptions common to all requests, the following exceptions may be returned:

- MissingRepositoryItemException: signifies that the caller did not provide a required repository item as an attachment to this request.
- UnsupportedIndexException: signifies that this Content Indexing Service did not support the IndexExtrinsicObject provided by the client.
- UnsupportedSubjectException: signifies that this Content Indexing Service did not support the SubjectExtrinsicObject provided by the client.

7.5.2 IndexContentResponse

The IndexContentRequest is sent by the Content Indexing Service as a response to an IndexContentRequest.

7.5.2.1 Syntax:

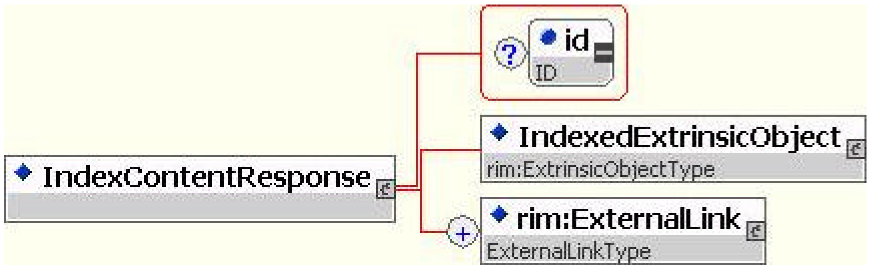


Figure 5: IndexContentResponse Syntax

7.5.2.2 Parameters:

- *ExternalLink:* This parameter specifies one or more ExternalLink elements that may be generated as index metadata during the indexing of the repository item.
- *id*: id attribute inherited from RegistryResponseType.
- *IndexedExtrinsicObject:* This parameter specifies the modified ExtrinsicObject for the repository item that has been indexed by the Content Indexing Service. The Content Indexing Service may add metadata such as Classifications, ExternalIdentifiers, name, description etc. to the IndexedExtrinsicObject element. There must not be an accompanying repository item as an attachment to this request.

7.6 Publishing a Content Indexing Service

Any publisher may publish an arbitrary content indexing service to an ebXML Registry. The content indexing service must be published using the existing LifeCycleManager interface. The publisher must use the existing SubmitObjectsRequest to publish:

A Service instances with two required slots named supportedObjectType and supportedMimeType. The values of these slots are explained in section 7.7.1.

- o A ServiceBinding instance contained within the Service instance
- o An ExternalLink instance on the ServiceBinding that must be resolvable to a web page describing:
  - o The format of the supported Index-able Content
  - o The format of the supported Index Definition File

Note that that no SpecificationLink is required since this specification is implicit for Content Indexing Services.

A content indexing service must be published with a default index definition file that must be an ExtrinsicObject and repository item pair. The ExtrinsicObject for the index definition must have two required slots named supportedObjectType and supportedMimeType. The values of these slots are explained in section 7.7.2.

7.6.1 Multiple Indexers and Index Definition Files

This specification allows clients to submit multiple indexers and index definition files for the same mimeType/objectType. How a registry handles multiple indexer and index definition file submission for the same type of content is a matter of registry specific policy. If a registry does not allow this then it must send an InvalidRequestException with a reason, when a duplicate indexer or index def is submitted. If a registry allows this then it must provide a conflict resolution mechanism to select the appropriate indexer and index definition file in some registry specific manner.

7.6.2 Restrictions On Publishing Content Indexing Services

A client may submit any content indexing service or index definition file. A registry may use registry specific policies to determine whether a client submitted content indexing service or index definition file are acceptable. For example a registry may require that the content indexing service or index definition file does not create excessive metadata. A registry may reject a SubmitObjectRequest with an InvalidRequestException and give a reason why the request was rejected, upon receiving requests publishing Content Indexing Service or Index Definition File that is unreasonable.

7.7 Dynamic Content Indexing

Some time during or after a publisher submits a repository item, the registry must check to see if there is a Content Indexing Service and index definition file registered for that type of repository item. This is referred to as Content Indexing Service resolution as described in section 7.7.1 and index definition file resolution as described in section 7.7.2.

If a Content Indexing Service and index definition file are found then the registry must invoke that service using the Content Indexing Protocol. In the invocation, it gives a repository item as Index-able Content and a repository item as Index Definition File within an IndexContentRequest. The Content Indexing Service must index the content and return the modified ExtrinsicObject for the Index-able Content such that it has index metadata generated from relevant portions of the Index-able Content.

The registry must store the repository item along with the modified ExtrinsicObject annotated with the index metadata once the Content Indexing Protocol is completed.

Note that a registry may do dynamic content indexing synchronous with the original SubmitObjectRequest request or it may do so asynchronously sometime after the request is committed. It is suggested that asynchronous indexing latency should be no more than 24 hours.

The result of dynamic content indexing is that index-able content gets indexed dynamically when it is submitted. Once indexed it is possible to use the index metadata to do dynamic content-based discovery of the index-able content.

7.7.1 Content Indexing Service Resolution Algorithm

When a registry receives a submission of an ExtrinsicObject EO1 and repository item pair, it must use the following algorithm to determine or resolve the content indexing service to be used to index that content:

1. Check if a Service instance for the indexer exists that has a slot named supportedObjectType with value matching the objectType of the ExtrinsicObject EO1, AND a slot named supportedMimeType with value matching the mimeType of the ExtrinsicObject. If so use that indexer.

2. Otherwise, Check if a Service instance for the indexer exists that has a slot named supportedObjectType with value matching the objectType of the ExtrinsicObject EO1. If so use that indexer.

3. Check if a Service instance for the indexer exists that has a slot named supportedMimeType with value matching the mimeType of the ExtrinsicObject EO1. If so use that indexer.

If no indexer is found then content should not be indexed. If an indexer is found then the registry must resolve an index definition file as defined next.

7.7.2 Index Definition File Resolution Algorithm

When a registry receives a submission of an ExtrinsicObject instance EO1 and repository item pair, it must first resolve a content indexing service as described in section 7.7.1.

If a content indexing service has been resolved then the registry must use the following algorithm to determine or resolve the index definition file to be used to index that content:

1. Check if an ExtrinsicObject instance for the index definition file exists that has a slot named supportedObjectType with value matching the objectType of the ExtrinsicObject EO1, AND a slot named supportedMimeType with value matching the mimeType of the ExtrinsicObject. If so use that index definition file.

2. Otherwise, Check if an ExtrinsicObject instance for the index definition file exists that has a slot named supportedObjectType with value matching the objectType of the ExtrinsicObject EO1. If so use that index definition file.

3. Check if an ExtrinsicObject instance for the index definition file exists that has a slot named supportedMimeType with value matching the mimeType of the ExtrinsicObject EO1. If so use that index definition file.

If no index definition file is found then content should not be indexed.

7.8 Dynamic Content-based Discovery

As described earlier, index-able content is automatically indexed when it is submitted to the registry. This content may subsequently be dynamically discovered using the index metadata within existing AdhocQueryRequest. Because the index metadata is based upon index-able content, an AdhocQueryRequest can perform dynamic content- based discovery.

7.9 Default XML Content Indexer

An ebXML Registry must provide the XML Content Indexing Service natively as a built-in service. The XML content indexing service accepts an XML instance document as its input and it accepts an XSLT Style sheet as a Content Definition File. Each type of content should have its own unique XSLT style sheet. For example and ebXML CPP document should have a specialize ebXML CPP index definition style sheet. The XML content indexing service must apply the XSLT style sheet to the XML instance document input to generate the index metadata. Since a single style sheet must be applied to both the ExtrinsicObject and the Index-able Content, we must assume the two documents to be composed within a single virtual document the schema for which is as follows:

Do we need the outer <somename> tag at all?

```
<somename>
        <ExtrinsicObject/>
        <CPP/>
</somename>
```

7.9.1 Publishing of Default XML Content Indexer

The default XML Content Indexing Service need not be explicitly published to an ebXML Registry. An ebXML Registry must provide the XML Content Indexing Service natively as a built-in service. This built-in service must be published to the registry as part of the intrinsic bootstrapping of required data within the registry.

8 Notes

These notes are here to not lose the thought and will be merged into the proposal later.

- o Need replacement term for index. Choices suggested so far are: promotion,
- o Need illustrative example to include sample Index Def file, sample CPP and sample output.
- o Do we need to replace repository item with ExtrinsicObject for inedex-able content? Reason, EO can be without repository item. [FN] No I don't think so. Nikola do you still want this?
- o How to establish ClassificationScheme, identificationScheme etc.
- o Instead of using slots maybe we can do Classifications to pre-defined schemes/nodes. Nikola please explain this. Do we still need it?
- o Same client goes to 2 registries that have 2 different indexer. User may be confused by different output for same input and control.
- o IndexContentResponse: how to handle any non-composed metadata such as ExternalIdentifier, Package etc.?

- How to track generated metadata separate from submitted metadata? Should we also log which indexer and index file created it?
- Should we allow a way for client to override default index def file and/or default indexer?