CentraSite

XQuery Navigation Tutorial

Version 9.0.1

June 2013

This document applies to CentraSite XQuery Navigation Tutorial Version 9.0.1.

Document ID: IINM-NAVIG-TUTORIAL-901-20130618

Table of Contents

Preface

This page discusses navigation in CentraSite using **XQuery**, the **W3C**'s query language for XML data. CentraSite is Software AG's SOA registry and repository. The following is based on CentraSite version 8.2.

1 Introduction

CentraSite registry data is stored as XML; therefore, the **W3C** standard XML query language **XQuery** is well suited to retrieving information from CentraSite. This chapter summarizes examples for querying CentraSite using XQuery. XQuery is normatively described in a number of W3C Recommendations. The two documents that are sufficient to the CentraSite XQuery user are the following (both published 14 December 2010):

- **XQuery 1.0: An XML Query Language (Second Edition) (http://www.w3.org/TR/xquery/)**
- **XQuery 1.0 and XPath 2.0 Functions and Operators (Second Edition) (http://www.w3.org/TR/xquery-operators/)**

CentraSite implements all of XQuery, with a few additional features. One of these extensions is that the CentraSite implementation of XQuery includes update facilities. The current W3C XQuery standard defines only retrieval (read-only) facilities, though update extensions are in preparation and will be part of the XQuery standard from version 2.0 onwards (see **XQuery Update Facility 1.0**). Note, however, that update support in CentraSite XQuery differs slightly from the W3C proposal. This chapter only discusses the retrieval features of CentraSite XQuery; in principal, it is also possible to use XQuery to update registry contents, but this should be used with caution.

This chapter describes how to query the CentraSite data store. It also presents numerous samples that illustrate how queries are formulated. Finally, it briefly explains the concept of XQuery modules and, in particular, it provides an introduction to the modules that are pre-loaded in the data store; they are primarily intended for use with the CentraSite predefined reports but they can also be used in the context of user-written queries.

2 How to Query CentraSite

This chapter is about the technical means of addressing queries to the CentraSite server. Although the techniques discussed here are not all there is to say about this topic, they suffice to use CentraSite XQuerying in many scenarios.

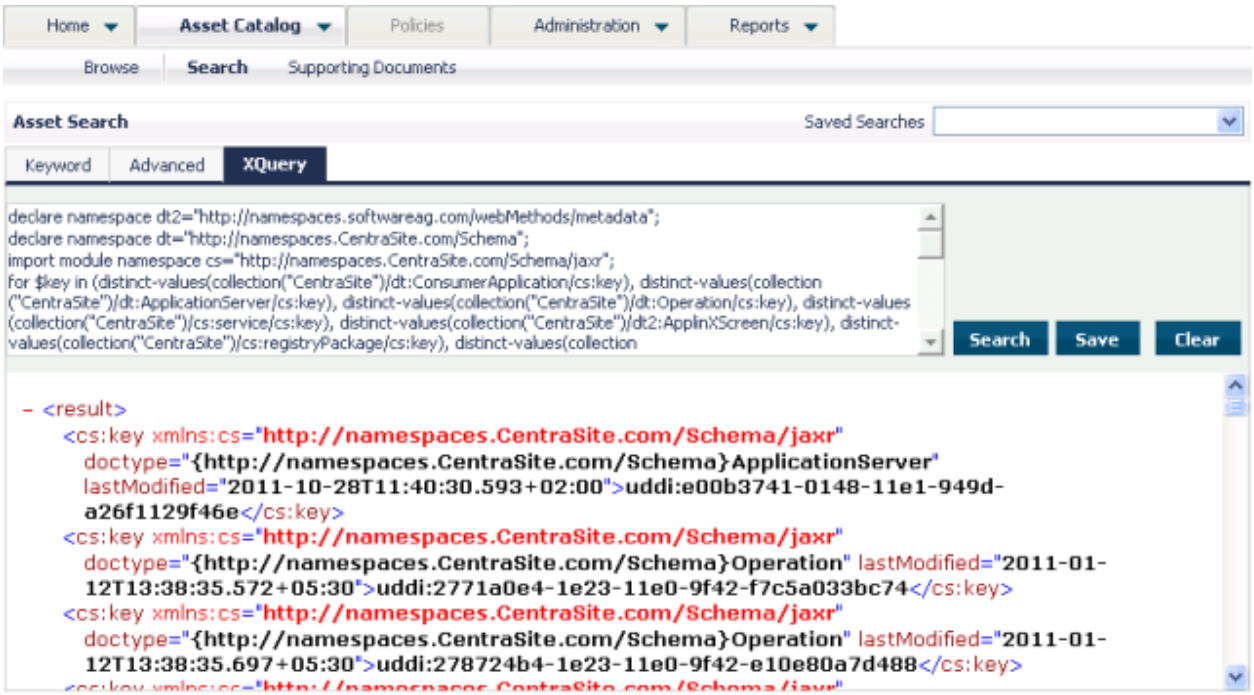
This chapter does not describe XQuerying in connection with CentraSite reporting. We recommend that you use one of the methods described below to develop your queries, then paste the queries into the reports at the appropriate places.

Using a CentraSite GUI

The most convenient way of querying the CentraSite registry/repository with an XQuery is to use one of the CentraSite GUIs, namely CentraSite Control or CentraSite Eclipse Perspective.

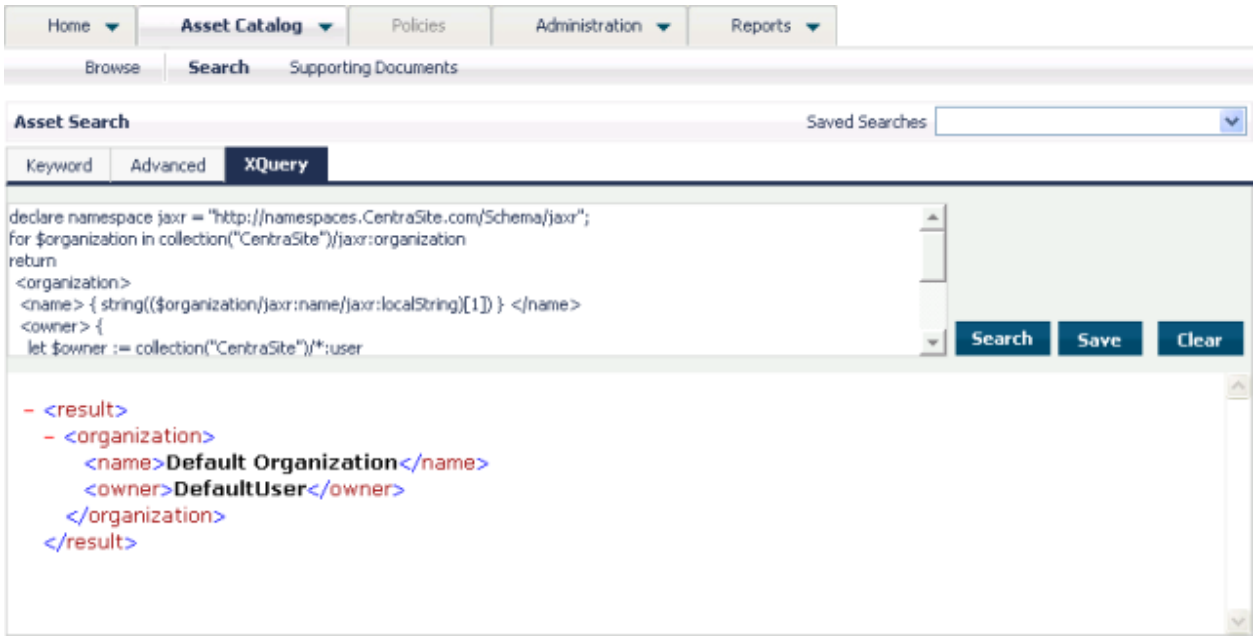
Using CentraSite Control

Using CentraSite Control, click the menu item **Asset Catalog** > **Search**, then select the **XQuery** tab. A window in which you can edit and execute queries opens. A first sample query is provided. Execute the sample query by clicking the **Search** button.



The **screenshot below** shows a query that retrieves the name and owner of all organizations, and also the result of executing the query. The query is listed below:

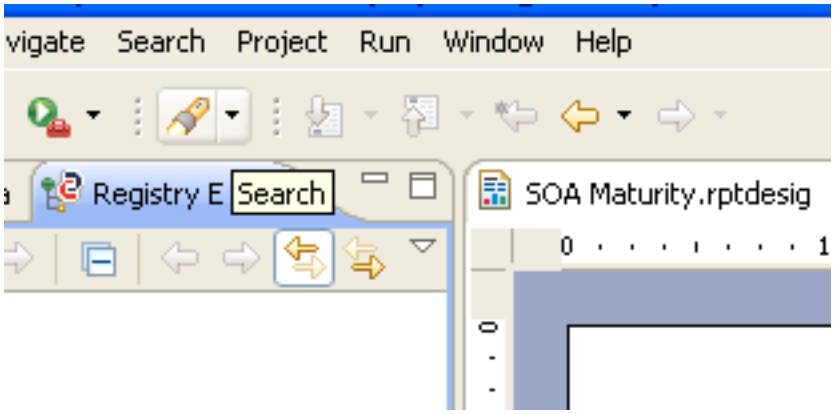
```
declare namespace jaxr = "http://namespaces.CentraSite.com/Schema/jaxr";
for $organization in collection("CentraSite")/jaxr:organization
return
 <organization>
  <name> { string(($organization/jaxr:name/jaxr:localString)[1]) } </name>
  <owner> {
    let $owner := collection("CentraSite")/*:user
      [jaxr:key = $organization/jaxr:owner]
    return string(($owner/jaxr:name/jaxr:localString)[1])
  } </owner>
 </organization>
```



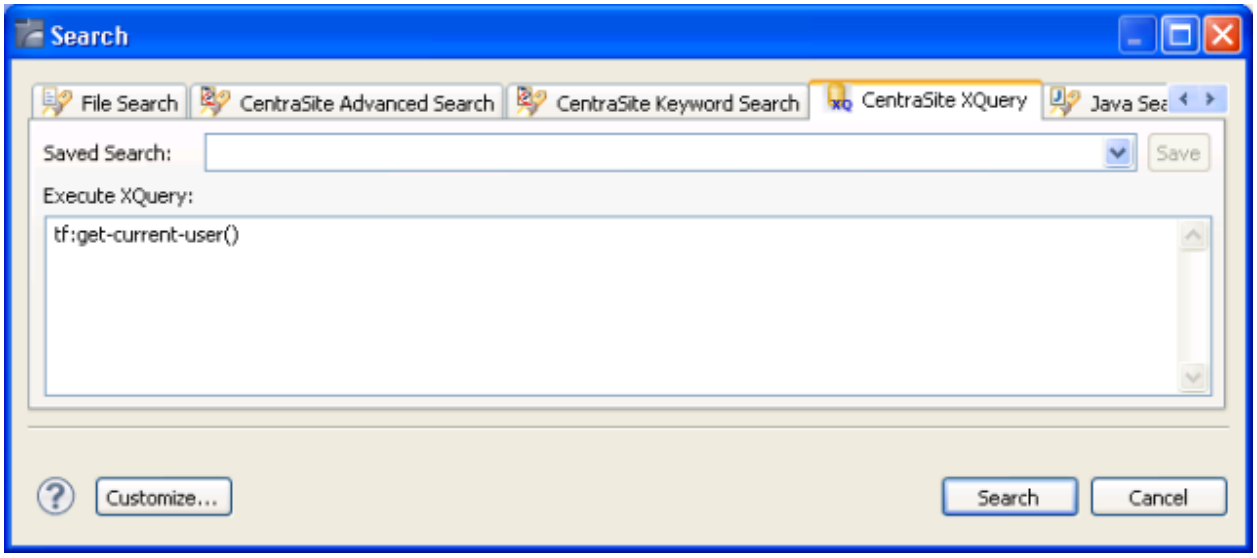
For additional queries, please see the chapter **Sample Queries**.

Using the Eclipse CentraSite Perspective

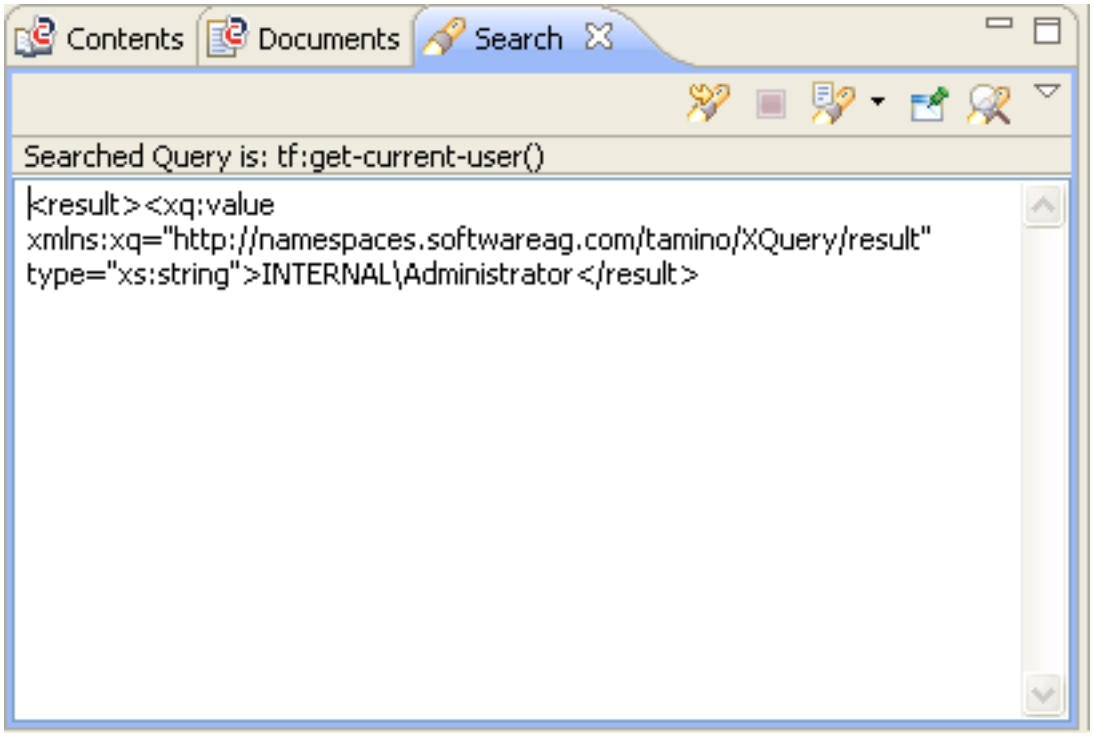
The Eclipse CentraSite Perspective offers the same functionality as CentraSite Control, therefore it can also be used to searching the registry/repository by XQuery. The “flashlight” icon that is also available in CentraSite Control is visible in the Eclipse menu:



Clicking on this icon opens the CentraSite Search dialog, which is slightly different from the CentraSite Control Search dialog: the Eclipse search dialog offers some Java-specific features and also it shows a different sample query, namely `tf:get-current-user()`, a predefined function that returns the identity of the current user:



Clicking on the **Search** button opens a view that reveals the results of the search:



Using XQJ

CentraSite offers an implementation of the Java™ XQJ specification, see **JSR 225: XQuery API for Java - XQJ (http://www.jcp.org/en/jsr/detail?id=225)**. The XQJ interface is an implementation of a specification developed under the Java Community Process (JCP), a mechanism to allow open source development of Java standards and software. For more information about the JCP, see **Java Community Process Home (http://www.jcp.org/en/home/index)**.

The following Java class, XQJClient.java, queries the CentraSite registry and outputs the result to standard output (STDOUT).

```
import com.softwareag.xqj.extension.TXQDataSourceFactory;
import javax.xml.xquery.*;


public class XQJClient {

  public static void main(String[] args)
  {
    String dbUrl = "http://cshost:53305/CentraSite/CentraSite";
    String userid = "Administrator";
    String password = "manage";
    String query = "collection('CentraSite')/*:organization/*:name";
```

```
    try {

      // get a new data source instance
      XQDataSource dataSource
        = TXQDataSourceFactory.createDataSource(dbUrl,userid,password);
      dataSource.setProperty("com.softwareag.tamino.xqj.defaultCollection",
                             "CentraSite");

      // establish a connection to the XQuery engine
      XQConnection conn = dataSource.getConnection();
      System.out.println("Connected to " + dbUrl);

      // create an expression object that is later used
      // to execute an XQuery expression
      XQExpression expr = conn.createExpression();
      XQResultSequence result = expr.executeQuery(query);
      System.out.println("Executed query:" + query);

      // output query results
      while (result.next())
      {
        try
        {
         String str = result.getAtomicValue();
         System.out.println(str.trim());

        } catch (Exception ex)
        {
          conn.rollback();
          ex.printStackTrace(System.err);
          System.exit(1);
        }
      }

      // free all resources
      result.close();
      expr.close();
      conn.rollback();

    } catch (XQException ex)
    {
      ex.printStackTrace(System.err);
      System.exit(1);
    }
  }
}
```

To run this class, set the strings `dbUrl`, `userid` and `password` to suitable values, and set the string `query` as desired (the query shown in the listing above should return a reasonable result).

Ensure that the following Java classes are included in the classpath (they can all be found in the *redist* subdirectory of the CentraSite home directory):

CentraSiteDynLoader.jar
cstUtils.jar
inmUtil.jar
inmUtilConf.jar
log4j.jar
stax-api.jar
TaminoAPI4J.jar
TaminoAPI4J-l10n.jar
wstx-asl.jar
xqj-api.jar
xqj-ino-api.jar

The following listing shows sample output after running the query:

```
Connected to http://pccentrasite:53305/CentraSite/CentraSite
Executed query:collection('CentraSite')/*:organization/*:name
<localString xml:lang="en-US">Default Organization</localString>
```

For a complete discussion of CentraSite's XQJ facilities, in particular the update feature, which is a Software AG-specific enhancement to the standard, refer to the **section CentraSite XQJ Interface of the CentraSite documentation**.

Using the HTTP GET Method

This access method allows commands to be sent to the CentraSite server using URLs sent to the server by means of the HTTP `GET` method. Queries are addressed by sending them via HTTP to the CentraSite server URL, which is *http://hostname:53305/CentraSite/CentraSite*, assuming that the port on which the CentraSite server listens is 53305 as suggested by the installation. The query is formed by concatenating:

1. the server URL (see above);
2. the collection (in the example above, this is */CentraSite*);
3. the `xquery` server command, which is "?_xquery=" followed by the query.

The following query requests a specific service named MusicQuoteService:

```
declare namespace cs="http://namespaces.CentraSite.com/Schema/jaxr";
collection("CentraSite")/cs:service
                   [xs:string(cs:name/cs:localString) = "MusicQuoteService"]
```

Open a browser, type *http://hostname:53305/CentraSite/CentraSite/CentraSite?_xquery=declare ...* into the address field, and press Enter. The query should not contain any spaces. If spaces are necessary (for example replacing the linebreak and around "namespace"), use the escape sequence "%20".

Assuming that the host has a CentraSite installed and assuming that it contains such a service, the following response appears in the browser window:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<ino:response xmlns:ino="http://namespaces.softwareag.com/tamino/response2"
              xmlns:xql="http://metalab.unc.edu/xql/">
 <xq:query xmlns:xq="http://namespaces.softwareag.com/tamino/XQuery/result">
  ...
 </xq:query>
 <ino:message ino:returnvalue="0">
  <ino:messageline>XQuery Request processing</ino:messageline>
 </ino:message>
 <xq:result xmlns:xq="http://namespaces.softwareag.com/tamino/XQuery/result">
  <service crt="2009-06-05T09:43:15.406+02:00"
           mod="2009-06-05T09:43:15.406+02:00"
           xmlns="http://namespaces.CentraSite.com/Schema/jaxr">
   <key v2Key="874ba2e3-51a4-11de-9115-ba2e8418f294">
    uddi:874ba2e3-51a4-11de-9115-ba2e8418f294
   </key>
   <owner>uddi:7470f530-4b6a-11de-850f-80568acb7b53</owner>
   <name>
    <localString xml:lang="en-GB">MusicQuoteService</localString>
   </name>
   <submittingOrganization>
    uddi:4299de0d-51a4-11de-9115-9970cdcef42
   </submittingOrganization>
   <externalLinks>
    ...
   </externalLinks>
   <classifications>
    ...
   </classifications>
   <majorVersion>1</majorVersion>
   <minorVersion>0</minorVersion>
   <stability>DYNAMIC</stability>
   <status>SUBMITTED</status>
   <providingOrganization>
    uddi:4299de0d-51a4-11de-9115-9970cdcef42
   </providingOrganization>
   <serviceBindings>
    ...
   </serviceBindings>
   <instanceSlots>
    ...
   </instanceSlots>
  </service>
 </xq:result>
 <ino:message ino:returnvalue="0">
  <ino:messageline>XQuery Request processed</ino:messageline>
 </ino:message>
</ino:response>
```

3 Sample Queries

This chapter provides a collection of sample queries that can be submitted, either as-is or modified, to a CentraSite server.

The first section deals with the so-called primary objects that are represented as XML documents in the database. These are Services, Organizations, ClassificationSchemes, Concepts, and instances of user-defined ObjectTypes. Subsequent sections cover the embedded objects, i.e. the objects that are contained in documents representing primary objects. These are Associations, Classifications, ExternalLinks, Slots and the like.

The queries deal with the data pertaining to the fictitious *Mozart Music GmbH*.

Retrieving Primary Objects

There are a number of so-called primary object types in the way the JAXR datamodel is represented in the CentraSite datastore. Instances of these object types are represented as XML documents in the datastore's *CentraSite* collection. This means to say that in order to find, for example, all organizations in a CentraSite installation the following XQuery is appropriate:

```
for $organization in collection("CentraSite")/*:organization
return $organization
```

The document element name pattern "*:organization" finds all documents with a local-name "organization" in any namespace. Since using the proper namespace allows the XQuery processor more scope for optimization (index utilization, for example), we can enhance the query:

```
declare namespace jaxr = "http://namespaces.CentraSite.com/Schema/jaxr"
for $organization in collection("CentraSite")/jaxr:organization
return $organization
```

To find a specific organization, add a "where" clause. Keep in mind that most elements to be retrieved (for example the "key" element, which holds the unique UDDI key) are also in the JAXR namespace.

```
declare namespace jaxr = "http://namespaces.CentraSite.com/Schema/jaxr";
for $organization in collection("CentraSite")/jaxr:organization
where $organization/jaxr:key = "uddi:4299de0d-51a4-11de-9115-99700cdcef42"
return $organization
```

To return a more sophisticated result than just the retrieved element as it is contained in the database, define a result structure as the query result and insert data taken from the element.

The query **shown below** returns a generated element "organization", which contains the organization's name and the name of its owner. The owner is found as the "user" (another primary object type) that is pointed to by the organization's "owner" sub-element.

```
declare namespace jaxr = "http://namespaces.CentraSite.com/Schema/jaxr";
for $organization in collection("CentraSite")/jaxr:organization
where $organization/jaxr:key = "uddi:4299de0d-51a4-11de-9115-99700cdcef42"
return
 <organization>
  <name> { string(($organization/jaxr:name/jaxr:localString)[1]) } </name>
  <owner> {
    let $owner := collection("CentraSite")/*:user
      [jaxr:key = $organization/jaxr:owner]
    return string(($owner/jaxr:name/jaxr:localString)[1])
  } </owner>
 </organization>
```

This results in a newly-created element node that contains elements for the organization's name and owner, for example:

```
- <xq:result xmlns:xq="http://namespaces.softwareag.com/tamino/XQuery/result">
  - <organization>
      <name>Mozart Music GmbH</name>
      <owner>INTERNAL\Administrator</owner>
    </organization>
  </xq:result>
```

Since finding an object's name is something that occurs frequently, it is advisable to introduce a user-defined function for this task:

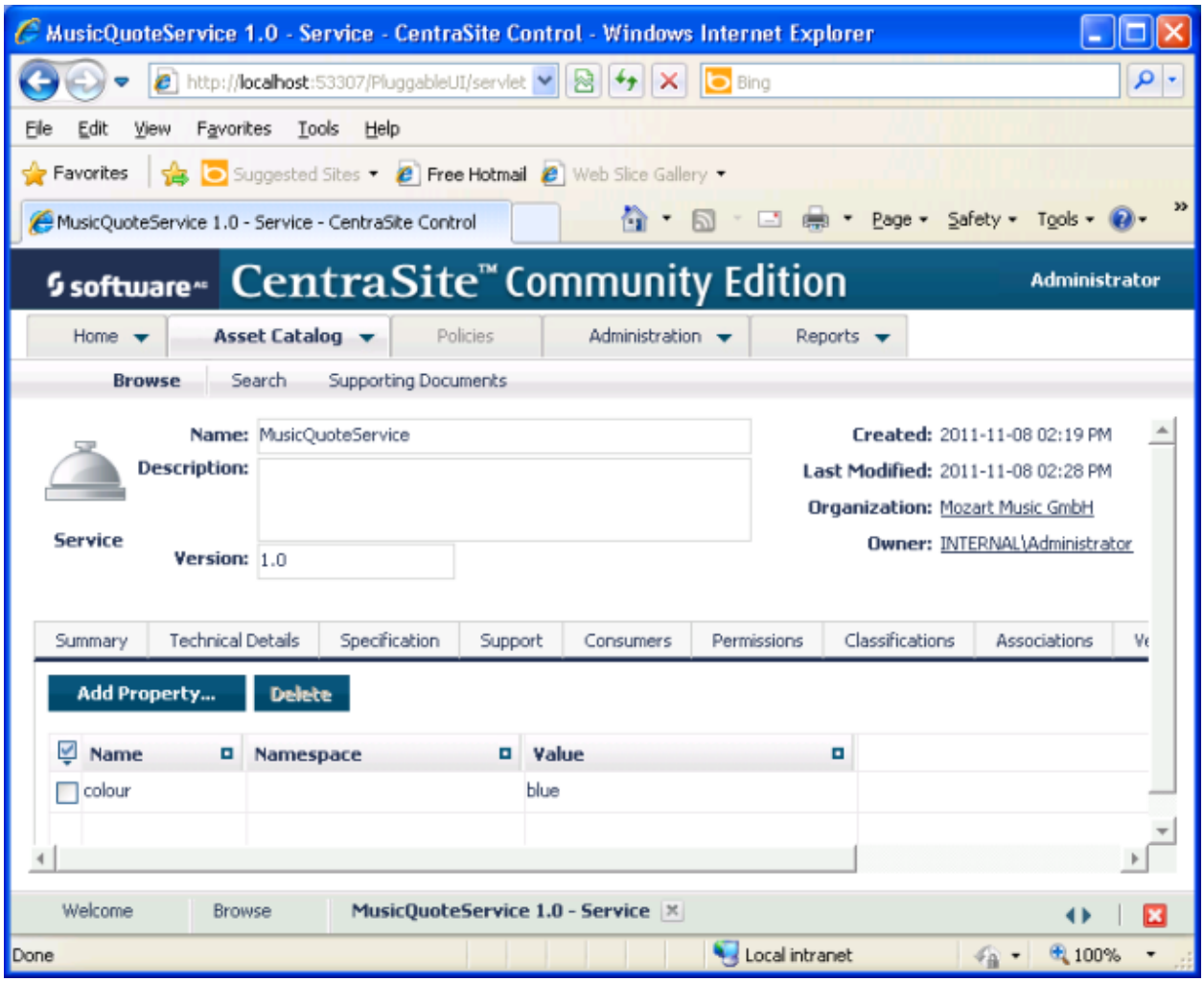
```
declare namespace jaxr = "http://namespaces.CentraSite.com/Schema/jaxr";
declare function local:getName($node as node()) as xs:string {
  string(($node/jaxr:name/jaxr:localString)[1])
};
for $organization in collection("CentraSite")/jaxr:organization
where $organization/jaxr:key = "uddi:4299de0d-51a4-11de-9115-99700cdcef42"
return
 <organization>
  <name> { local:getName($organization) } </name>
  <owner> {
    let $owner := collection("CentraSite")/*:user
      [jaxr:key = $organization/jaxr:owner]
    return local:getName($owner)
  } </owner>
 </organization>
```

Other primary object types, besides "organization" and "user", are "service", "package", "classificationScheme", "concept", and "externalLink". In addition, instances of user-defined object types become primary objects, since a corresponding schema that defines the new type as a new doctype to the database is created whenever a new type is added to CentraSite.

Retrieving Embedded Objects

Examples of instances of embedded object types are "Classifications", "Associations", and "Slots". These three kinds of objects may occur within primary objects of different types; the way they are included is always the same regardless of the primary object's type.

Firstly, we consider the slots provided with "MusicQuoteService". For this, we assume that the company offering these kinds of services has bundled its services into different colour schemes, and therefore attaches a property named colour to the services that are associated with a particular colour scheme, as shown below:



The following query would retrieve that property:

```
declare namespace jaxr = "http://namespaces.CentraSite.com/Schema/jaxr";
declare function local:getName($node as node()) as xs:string {
  string($node/jaxr:name/jaxr:localString)
};
for $service in collection("CentraSite")/jaxr:service
where local:getName($service) = "MusicQuoteService"
return $service/jaxr:instanceSlots/colour
```

This would return:

```
<colour xmlns="">blue</colour>
```

When dealing with function calls, the optimizer sometimes decides to inline the function. In other words, before the query is executed, a function call is replaced by the body of the function. However, a function call may or may not be inlined; in the case of the current query, if the function call to `local:getName` is not inlined, this might inhibit the use of an index on the service's name. If the performance of a query is unsatisfactory, it might be worth trying to force function inlining by preceding the function declaration with an `inline` query pragma as follows:

```
declare namespace jaxr = "http://namespaces.CentraSite.com/Schema/jaxr";
{?inline?}
declare function local:getName($node as node()) as xs:string {
  string($node/jaxr:name/jaxr:localString)
};
for $service in collection("CentraSite")/jaxr:service
where local:getName($service) = "MusicQuoteService"
return $service/jaxr:instanceSlots/colour
```

Likewise, the optimization pragma {`?optimization inline="full"?`} at the very beginning of the query would cause all user-defined functions to be inlined except those that directly or indirectly reference themselves.

Since dealing with the other two general embedded object types, Classification and Association, is more complicated, they are dealt with in the separate sections **Using Classifications** and **Navigating along Associations**.

Other embedded object types are proprietary to certain primary object types, for example "serviceBindings", which only occur as parts of "services". To find out about the structure, use queries to return whole objects. A good method to find out which elements exist below a certain point in an XML document is to use the `local-name` function:

```
declare namespace jaxr = "http://namespaces.CentraSite.com/Schema/jaxr";
declare function local:getName($node as node()) as xs:string {
  string($node/jaxr:name/jaxr:localString)
};
for $service in collection("CentraSite")/jaxr:service
where local:getName($service) = "MusicQuoteService"
return for $subelement in $service/*
       return local-name($subelement)
```

The result is:

```
<xq:result xmlns:xq="http://namespaces.softwareag.com/tamino/XQuery/result">
  <xq:value>key</xq:value>
  <xq:value>owner</xq:value>
  <xq:value>name</xq:value>
  <xq:value>description</xq:value>
  <xq:value>submittingOrganization</xq:value>
  <xq:value>externalLinks</xq:value>
  <xq:value>classifications</xq:value>
  <xq:value>majorVersion</xq:value>
  <xq:value>minorVersion</xq:value>
  <xq:value>stability</xq:value>
  <xq:value>status</xq:value>
  <xq:value>providingOrganization</xq:value>
  <xq:value>serviceBindings</xq:value>
  <xq:value>instanceSlots</xq:value>
</xq:result>
```

Using Classifications

Objects are classified in order to allow a means of semantic retrieval. With the "music" sample data, we have established the following taxonomy describing kinds of music:

e	Classical music
ec	Classical chamber music
ecb	Brass chamber music
eo	Classical music for orchestra
u	Non-classical music
us	Music for small bands
ub	Big-band music

Later on, the services that enable the potential customer to search through the available sheet music are classified using this classification. The following query finds the "Concept" object representing the taxonomy entry "ecb" (i.e. brass chamber music).

```
declare namespace jaxr = "http://namespaces.CentraSite.com/Schema/jaxr";
for $concept in collection("CentraSite")/jaxr:concept
where $concept/jaxr:name/jaxr:localString = "ecb"
return $concept
```

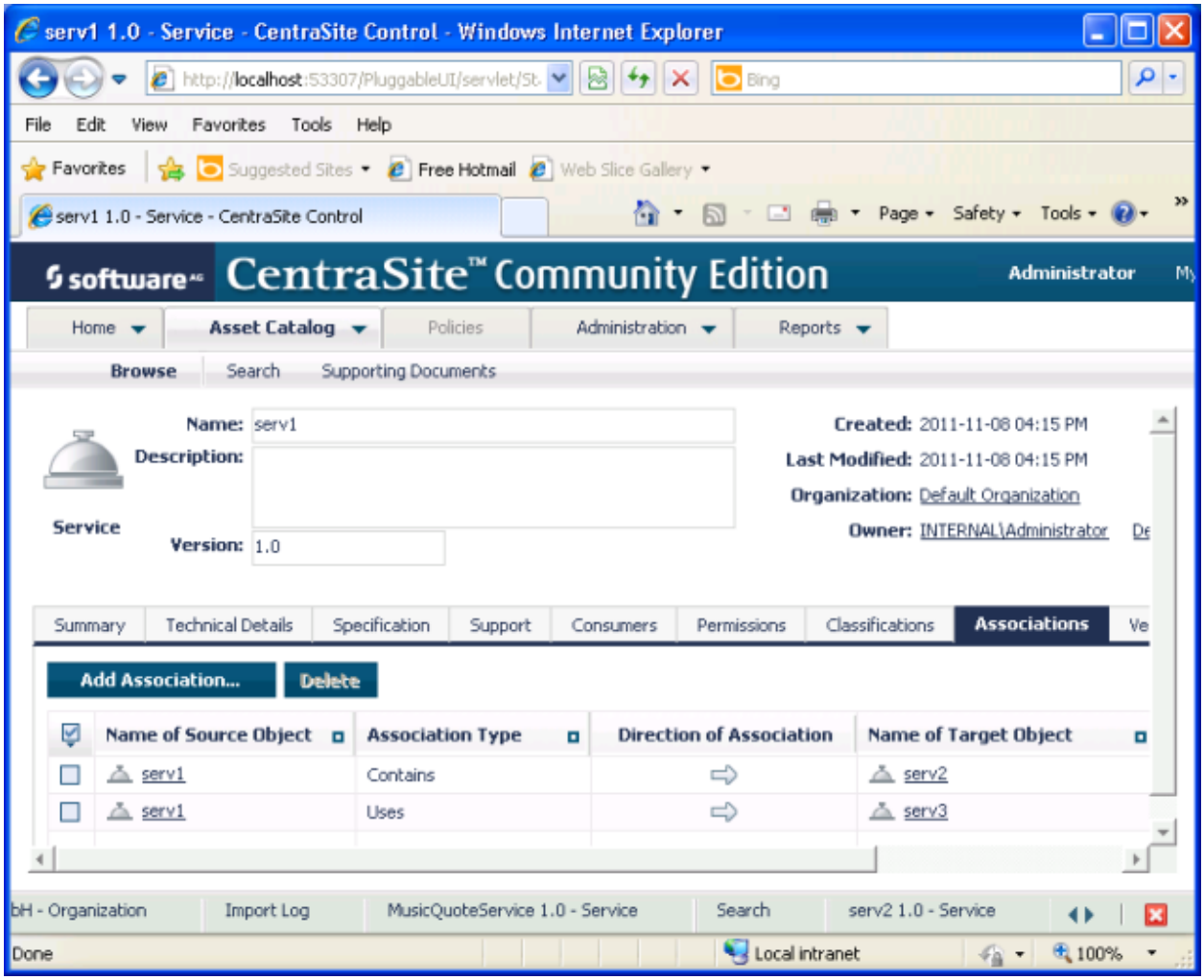
Note that we do not use the user-defined function local:getName, since this would assume that each concept has only one name; this is not necessarily true in an internationalized environment.

The next query uses this concept to retrieve those services that are classified by this concept, meaning they are appropriate to search the company's catalogue for sheet music applicable to brass chamber music:

```
declare namespace jaxr = "http://namespaces.CentraSite.com/Schema/jaxr";
declare function local:getName($node as node()) as xs:string {
  string($node/jaxr:name/jaxr:localString)
};
let $concept := ↩
collection("CentraSite")/jaxr:concept[jaxr:name/jaxr:localString="ecb"]
for $service in collection("CentraSite")/jaxr:service
where $service//jaxr:classification/jaxr:concept = $concept/jaxr:key
return local:getName($service)
```

Navigating Along Associations

As a first example, assume that a service "serv1" points towards two other services "serv2" and "serv3" using the predefined association types "Contains" and "Uses" respectively.



The following query finds all registry entries pointed to from "serv1" that use the association "Contains" once. Firstly, the query finds the concept corresponding to the association type "Contains" and the service "serv1". Associations are stored with the primary objects where they originate, i.e. the two associations starting from "serv1" are stored as "cs:association" descendants of "serv1".

```
declare namespace cs="http://namespaces.CentraSite.com/Schema/jaxr"
let $concept := collection("CentraSite")/cs:concept
                [cs:name/cs:localString = "Contains"]/cs:key
let $service := collection("CentraSite")/
                                cs:service[cs:name/cs:localString="serv1"]
for $target in $service//cs:association[cs:associationType=$concept]/
                                cs:targetObject
return collection("CentraSite")/cs:*[cs:key=$target]
```

This returns the complete service "serv2".

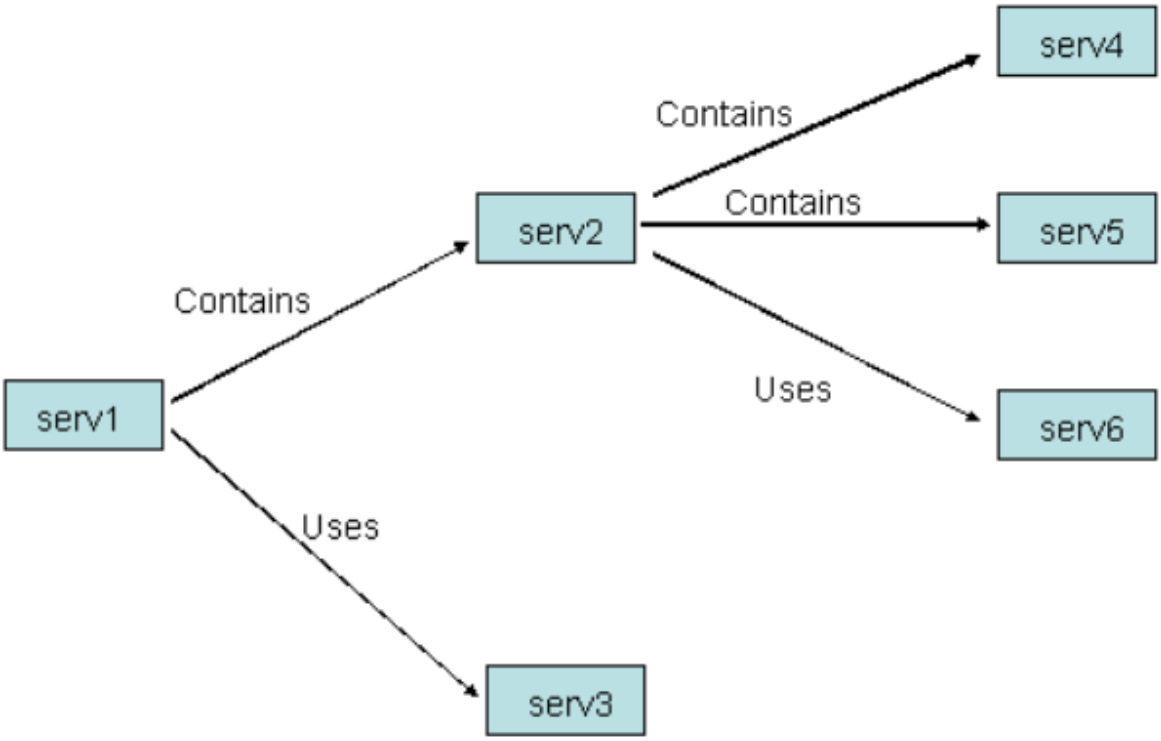
The next query uses a local function "getName" to return the name of the service:

```
declare namespace cs="http://namespaces.CentraSite.com/Schema/jaxr"
declare function local:getName($node as node()) as xs:string
{
  xs:string($node/cs:name/cs:localString)
}
let $concept := collection("CentraSite")/cs:concept
              [cs:name/cs:localString = "Contains"]/cs:key
let $service := collection("CentraSite")/cs:service
              [cs:name/cs:localString="serv1"]
for $target in $service//cs:association
              [cs:associationType=$concept]/cs:targetObject
return local:getName(collection("CentraSite")/cs:*[cs:key=$target])
```

The following query does the same, but uses another local function "followAssocOnce" to follow the association:

```
declare namespace cs="http://namespaces.CentraSite.com/Schema/jaxr"
declare function local:getName($node as node()) as xs:string
{
  xs:string($node/cs:name/cs:localString)
}
declare function local:followAssocOnce($entry as node(),
                                       $assoc as xs:string)
  as node()*
{
  for $key in $entry//cs:association
              [cs:associationType=$assoc]/cs:targetObject
  return collection("CentraSite")/*[cs:key=$key]
}
let $concept := collection("CentraSite")/cs:concept
              [cs:name/cs:localString = "Contains"]/cs:key
let $service := collection("CentraSite")/cs:service
              [cs:name/cs:localString="serv1"]
for $target in local:followAssocOnce($service,$concept)
return local:getName($target)
```

We assume furthermore that "serv2" in turn points to three other entries, namely to two services "serv4" and "serv5", again along "Contains"; and to another service "serv6" along "Uses", as shown below:



The following query introduces a recursive local function "followAssocMultiple" that retrieves all entries reachable by following a given association starting from a given entry. Given the starting entry "serv1" and the association "Contains", it yields the three services "serv2", "serv4" and "serv5":

```
declare namespace cs="http://namespaces.CentraSite.com/Schema/jaxr";
declare function local:getName($node as node()) as xs:string
{
  xs:string($node/cs:name/cs:localString)
}
declare function local:followAssocOnce($entry as node(),
                                       $assoc as xs:string)
                  as node()*
{
  for $key in $entry//cs:association
                           [cs:associationType=$assoc]/cs:targetObject
  return collection("CentraSite")/*[cs:key=$key]
}
declare function local:followAssocMultiple($entry as node(),
                                           $assoc as xs:string)
                  as node()*
{
```

```
  for $target in local:followAssocOnce($entry,$assoc)
  return ($target,local:followAssocMultiple($target,$assoc))
}
let $concept := collection("CentraSite")/cs:concept
                            [cs:name/cs:localString = "Contains"]/cs:key
let $service := collection("CentraSite")/cs:service
                            [cs:name/cs:localString="serv1"]
for $target in local:followAssocMultiple($service,$concept)
return local:getName($target)
```

If the above query is applied to an entry that is part of or leads to a loop, stack overflow occurs. The next query does the same, but can cope with loops. This requires that the recursive function takes three parameters. The first parameter is the list of previously found nodes. The second parameter holds the current nodes, i.e. the nodes found in the previous step. Each time the function is called, it finds the nodes reached directly from the current nodes. These become the new current nodes unless they have previously been visited.

```
declare namespace cs="http://namespaces.CentraSite.com/Schema/jaxr";
declare function local:getName($node as node()) as xs:string
{
  xs:string($node/cs:name/cs:localString)
}
declare function local:followAssocOnce($entry as node(),
                                       $assoc as xs:string)
                  as node()*
{
  for $key in $entry//cs:association
    [cs:associationType=$assoc]/cs:targetObject
  return collection("CentraSite")/*[cs:key=$key]
}
declare function local:followAssocMultiple($current as node()*,
                                           $assoc as xs:string)
                  as node()* {
  let $first := for $node in $current
                return local:followAssocOnce($node,$assoc)
  return local:followAssocMultipleRecursive((),$first,$assoc)
}
declare function local:followAssocMultipleRecursive(
                            $old as node()*,
                            $current as node()*,
                            $assoc as xs:string) as node()* {
  if ($current)
  then let $new := for $entry in $current
                   return local:followAssocOnce($entry,$assoc)
       let $nextold := $old union $current
       let $nextcurrent := $new except $nextold
       return local:followAssocMultipleRecursive($nextold,
                                                 $nextcurrent,
                                                 $assoc)
  else $old
}
```

```
let $concept := collection("CentraSite")/cs:concept
                          [cs:name/cs:localString = "Contains"]/cs:key
let $service := collection("CentraSite")/cs:service
                          [cs:name/cs:localString="serv1"]
for $target in local:followAssocMultiple($service,$concept)
return local:getName($target)
```

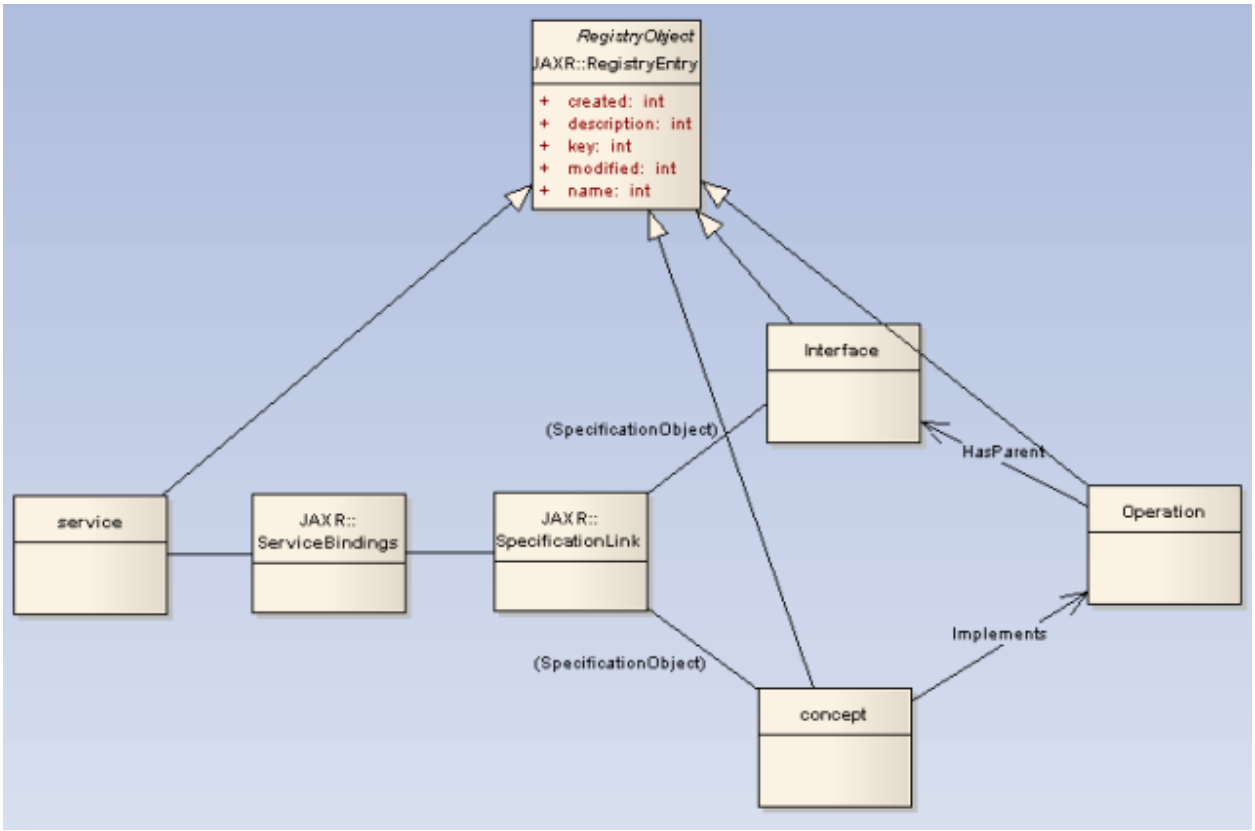
Things are slightly more complicated when following relations backwards. To retrieve all entries from which a given entry "$entry" can be reached via a relation "$assoc", find each entry and check its relations. The following query uses a local function "followAssocOnceBackwards" to find all entries starting from "serv4" and traversing "Contains" backwards, yielding "serv2".

```
declare namespace cs="http://namespaces.CentraSite.com/Schema/jaxr"
declare function local:getName($node as node()) as xs:string
{
  xs:string($node/cs:name/cs:localString)
}
declare function local:followAssocOnceBackwards($entry as node(),
                                                 $assoc as xs:string)
                       as node()*
{
  for $everyentry in collection("CentraSite")/*
  for $target in $everyentry//cs:association
              [cs:associationType=$assoc]/cs:targetObject
  where $target = $entry/cs:key
  return $everyentry
}
let $concept := collection("CentraSite")/cs:concept
              [cs:name/cs:localString = "Contains"]/cs:key
let $service := collection("CentraSite")/cs:service
              [cs:name/cs:localString="serv4"]
for $target in local:followAssocOnceBackwards($service,$concept)
return local:getName($target)
```

Operations and Services

The representation of services together with their accompanying artifacts in the registry and repository is a topic of its own. When a service is added by importing its WSDL, usually a series of mutually-connected entries is created. Each service that is included this way has at least one `ServiceBinding`. This binding, which is a JAXR object of its own, is referred to from the service by an implicit reference. In the database, the bindings are contained in the service document. The binding points to two `SpecificationLink` JAXR objects that, in CentraSite, also come with the service document, as descendants of their bindings. Each of the two specification links has an attribute `SpecificationObject`. The specification object of the first specification link points to the operations belonging to that binding via an association of type `Implements`.

Much more could be said about what happens to a registry when a service's WSDL is imported. In particular, there is another way to get to a service's operations, namely by following the bindings' second specification links. The complete infomodel for a service is shown in the following diagram:



We do not pursue this further, since the information collected so far is sufficient for the following query. The query finds all organizations, their services and these services' operations. Note that `Operation` is a primary object type, but does not reside in the JAXR namespace *http://namespaces.CentraSite.com/Schema/jaxr*; rather, it is in the namespace *http://namespaces.CentraSite.com/Schema*, as are many CentraSite-specific extensions to the JAXR datamodel.

Note also the slightly modified user-defined function `getName`, which can handle the case in which an object has more than one name (internationalization).

```
declare namespace csjaxr = "http://namespaces.CentraSite.com/Schema"
declare namespace jaxr = "http://namespaces.CentraSite.com/Schema/jaxr"
declare function local:getName($node as node()) as xs:string {
  string(($node/jaxr:name/jaxr:localString)[1])
};
let $implKey := string(collection('CentraSite')
                /jaxr:concept[local:getName(.)="Implements"]/jaxr:key)
for $org in collection('CentraSite')/jaxr:organization
return
 <org>
```

```
 <name>{local:getName($org)}</name>
 <services> {
  for $serv in collection('CentraSite')/jaxr:service
  where $serv/jaxr:providingOrganization = $org/jaxr:key
  return
   <service>
    <name>{local:getName($serv)}</name>
    <operations> {
     let $so := ($serv//jaxr:serviceBinding//jaxr:specificationLink
              /jaxr:specificationObject)[1]
     for $assoc in collection('CentraSite')/jaxr:concept[jaxr:key=$so]
              //jaxr:association[jaxr:associationType = $implKey]
     let $target := collection('CentraSite')
              /csjaxr:Operation[jaxr:key = $assoc/jaxr:targetObject]
     return
      <operation> { local:getName($target) } </operation>
    } </operations>
   </service>
 } </services>
</org>
```

The following is an excerpt of the result after querying a registry into which the service MusicQuoteService had previously been imported:

```
<org>
    <name>Mozart Music GmbH</name>
    <services>
        <service>
            <name>MusicQuoteService</name>
            <operations>
                <operation>MusicQuoteMethod</operation>
            </operations>
        </service>
```

4 XQuery Modules

Mainly for use in its predefined reports, CentraSite comes with a collection of XQuery modules that contain predefined XQuery functions. These functions can be used in other queries, either contained in reports or written by CentraSite users. These modules are stored as text (i.e. non-XML) documents in the CentraSite registry and can again be browsed by means of XQuery.

An overview of all the modules that come with CentraSite can be obtained by the following query:

```
declare namespace ino="http://namespaces.softwareag.com/tamino/response2"
for $mod in collection('ino:source')/ino:module/..
return tf:getDocname($mod)
```

An excerpt of the query result might look as such.

```
<xq:result xmlns:xq="http://namespaces.softwareag.com/tamino/XQuery/result">
  <xq:value>BusinessQueryManager.xquery</xq:value>
  <xq:value>SchemaTransformer.xquery</xq:value>
  <xq:value>NotificationAssessories.xquery</xq:value>
  <xq:value>JAXRConstants.xquery</xq:value>
  <xq:value>uddiJaxrMapping.xquery</xq:value>
  <xq:value>sin.xquery</xq:value>
  <xq:value>CentraSiteDateTime.xquery</xq:value>
  <xq:value>CentraSiteUtil.xquery</xq:value>
  <xq:value>CentraSiteReportAssetChanges.xquery</xq:value>
  <xq:value>CentraSiteReportOrganizationSummary.xquery</xq:value>
```

The next query returns the source code of such a module, in this case the module `CentraSiteUtil.xquery`. This module contains many functions that may prove generally useful.

```
declare namespace ino="http://namespaces.softwareag.com/tamino/response2"
for $mod in collection('ino:source')/ino:module/..
where tf:getDocname($mod) = "CentraSiteUtil.xquery"
return tf:text-content($mod)
```

For additional helpful examples of XQueries dealing with CentraSite, please refer to the reports, which contain many queries accessing CentraSite either directly, or using functions from modules, or both. Reports can be viewed from CentraSite Control from the report's detail view via **Actions** > **Show Template File**, or more conveniently in the Designer's BIRT perspective. For more information on reports, refer to CentraSite Reporting.

5 Links

- XQuery 1.0: An XML Query Language (Second Edition) (http://www.w3.org/TR/xquery/)
- XQuery 1.0 and XPath 2.0 Functions and Operators (Second Edition) (http://www.w3.org/TR/xquery-operators/)
- XQuery Update Facility 1.0 (http://www.w3.org/TR/xquery-update-10)

- JSR 225: XQuery API for Java - XQJ (http://www.jcp.org/en/jsr/detail?id=225)
- Java Community Process Home (http://www.jcp.org/en/home/index)
- CentraSite Documentation: CentraSite XQJ Interface (http://documentation.softwareag.com/webmethods/wmsuites/wmsuite8-2_ga/CentraSite/8-2-SP1_CentraSite/dg-xqj/overview.htm)