Faceted Search for CUAHSI HIS Migration Document

Alex Bedig

September 2011

Tufts University

# Overview

This document describes the design and implementation of both client side and server side implementations of the faceted search web services that I have been prototyping for inclusion in CUAHSI HIS. It is intended for the developers and mangers of CUAHSI HIS, in order to facilitate the transition from my internal development to environment to a full integration with existing CUAHSI HIS services and products.

Faceted search describes a method for exposing multiple independent attribute collections of metadata to a user in order to facilitate data discovery. Faceted search implements a sieve filter in which user selections in one attribute collection reduce the options presented to a user in the other attribute collections by removing options that do not conform to the selections the user has already made. Benefits of this approach include a reduction of null search results by constraining the options the user can make by the data that is available, and improved accessibility for the naïve user, who is now able to identify data relevant to his question by drilling down across multiple attributes instead of being required to understand the domain-derived meaning of a single attribute. This latter point is particularly relevant to the problem of facilitating search and data discovery of inter-disciplinarily used data, such as hydrologic information.

# Server-side

### Objectives of Current Database Implementation

The roles of these tables, and how they fit with the original HISCentral database structure, are described below. The primary objectives behind creating this design were:

1. To enable faceted search over the SeriesCatalog in minimal time, and in a way that could be maintained when additional data is harvested or existing series metadata changes.
2. To avoid modifying the existing HISCentral tables such that existing services would not be interrupted were this database to support both versions.

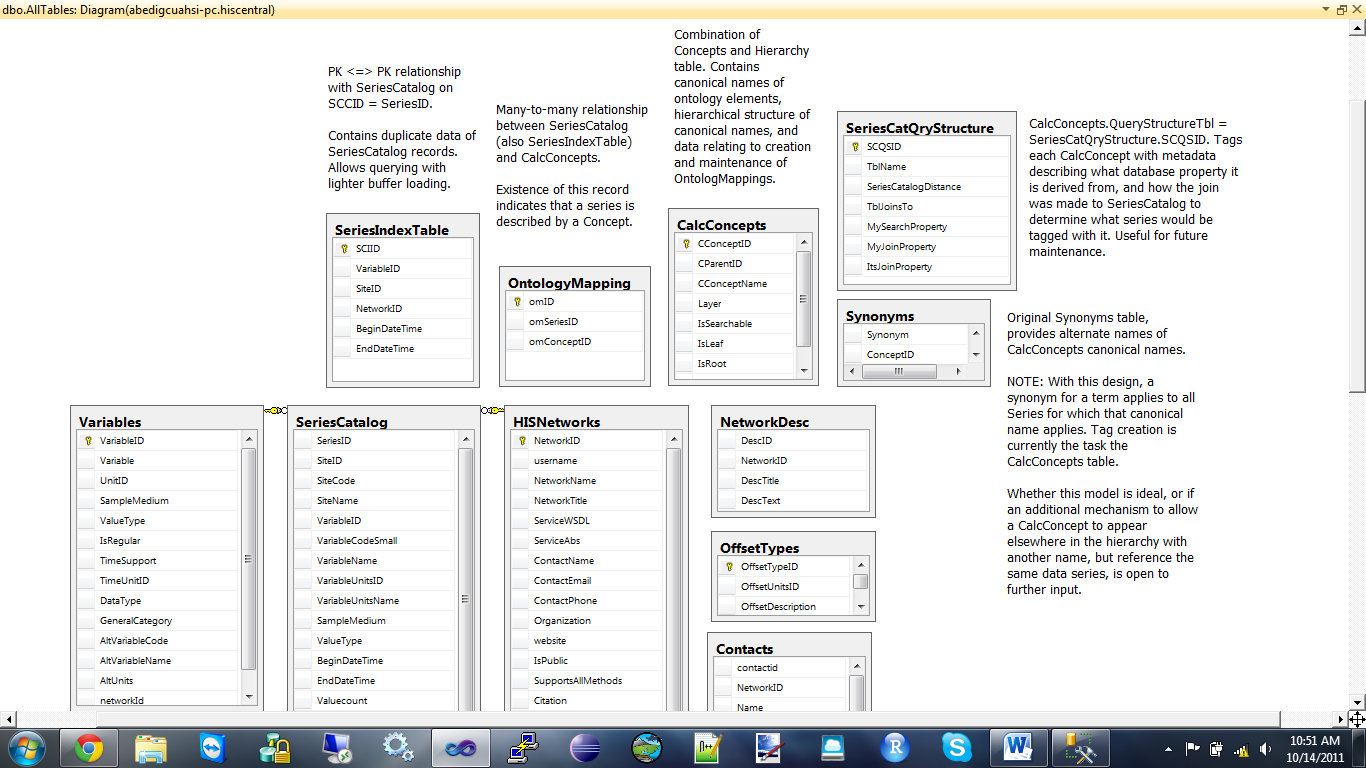
That being said, this implementation is meant to demonstrate one way to implement faceted search for the purpose of demonstrating what faceted search could add to the HIS user experience. It may be appropriate that the operational managers of this service alter the database schema.

## Database Design

Faceted search implements adjustments to the way that SeriesCatalog data are indexed, queried, and related to Concepts that allow for any property in the database to be the basis of a controlled-vocabulary search facet. We introduce four new tables to drive this procedure:

1. CalcConcepts
2. SeriesCatQryStructure
3. OntologyMapping
4. SeriesIndexTable

A schema diagram, in which all tables relevant to faceted search are commented, appears below:



CalcConcepts is an update of the Concepts table. CalcConcepts contains the canonical names of instances of the HIS Ontology, child-parent relationships within these instances, information describing the location of an instance within the hierarchy, and a key from the SeriesCatQryStructure table defining the facet from which each instance is derived.

OntologyMapping records relate SeriesCatalog records to CalcConcept records, defining whether a time series is described by that CalcConcept. Existence of these records is the underlying mechanism driving faceted search.

SeriesCatQryStructure defines database properties that are used as unique sub-ontologies within the larger CalcConcepts hierarchy. Every CalcConcept has a key to a record in the SeriesCatQryStructure table defining the sub-ontology it belongs to. In faceted search, a SeriesCatalog record passes the facet filter if it has OntologyMapping records joining it to one of the CalcConcepts selected in the UI for each distinct SeriesCatQryStructure table – or sub-ontology – from which any of user selections have been made. Further, the data in these records is useful for adjusting the OntologyMapping records that reference time series when the metadata of those time series, as represented by the records of the property from which the CalcConcept is derived, changes in the harvesting process.

SeriesIndexTable is a replication of the SeriesCatalog, only it contains only the keys necessary to facilitate faceted search, as well as the temporal BeginDateTime and EndDateTime properties to allow for temporal filtering of SeriesCatalog records. Its purpose is to support the necessary search pattern of faceted search with a minimum impact on the database buffer. It reduces the impact on the buffer by not selecting from the SeriesCatalog, which has other properties that are non-essential for search. The SeriesIndexTable must be updated to reflect the latest state of the SeriesCatalog whenever SeriesCatalog records are added or removed, after which its indexes must be rebuilt.

Descriptions of the non-obvious CalcConcepts.IsSearchable and CalcConcepts.IsRoot are below:

CalcConcepts.IsSearchable defines whether a Concept instance is tagged with data via OntologyMappings. This variable is used on the client-side to determine what ConceptIDs should be searched in response to a user’s selections. Inherently, a leaf Concept must be IsSearchable, because a leaf Concept only exists to reference SeriesCatalog data. However, mid-level or root instances may exist as a means of organizing and grouping other facets, and these facets do not necessarily have to be tagged with data. When a user selects one of these instances, all of its child instances are selected by default, meaning search will occur over all of the IsSearchable children of the not IsSearchable parent. However, a user is able to refine this behavior by deselecting individual child instances prior to committing to search or further pair the facet options. In the case where a user deselects all child instances of a not IsSearchable instance and then commits a Search or Next action, the client automatically searches for all children of the not IsSearchable instance that are IsSearchable. This behavior is transparent to the user.

CalcConcepts.IsRoot indicates whether a Concept instance is a top-level instance for a facet. The term “root” is conflated in this application, because a hierarchical ontology by definition can only have one root. In this application, the single root is assumed to exist but is not represented by a record, while Concept instances that have IsRoot = true specify 0 as their parent identity, meaning they reference no record as their parent.

## Database Construction and Maintenance

This section describes the procedures for building data to support faceted search.

### Creating a new Facet from existing HIS Central metadata

Facets consist of managed hierarchical vocabularies, where at least the leaves reference data in the SeriesCatalog table. To generate the list of leaves, identify a property of a table in the HIS Central database that can be joined to SeriesCatalog records to use as the basis for the facet. Create a record in the SeriesCatQryStructure table defining the following properties:

|  |  |
| --- | --- |
| Property | Description |
| TblName | The name of the table from which the search facet vocabulary is derived. |
| SeriesCatalogDistance | (Deprecated) The number of joins required to connect this table to the SeriesCatalog table. A facet from the SeriesCatalogTable has a SeriesCatalogDistance of 0. |
| TblJoinsTo | The table the table this search facet derives from immediately joins to on the path of joining to the SeriesCatalog table. |
| MySearchProperty | The property from which the search facet vocabulary is derived. |
| MyJoinProperty | The property from the TblName table that is used to join to the TblJoinsTo table. |
| ItsJoinProperty | The property from the TblJoinsTo table that is used to join the TblName table. |

Next, create a root ontology element for the facet. This is a record in the CalcConcepts table with the following distinguishing properties:

* IsRoot = true
* IsLeaf = false
* QryStructureTbl = integer ID of the SeriesCatQryStructure record that represents this facet.
* IsSearchable = false
* CparentID = 0

Last, create CalcConcepts records for each item in the facet. The generation of the canonical names of the facet comes from a distinct select operation on the property that defines the vocabulary of the facet. After the leaves have been created, additional mid-tier facets can be added and leaves assigned to them by modifying the CParentID properties of the leaves. As an example, the query that produced the CalcConcepts records for the SampleMedium facet is:

INSERT INTO CalcConcepts (CParentID, CConceptName, Layer, IsSearchable, IsLeaf, IsRoot, QueryStructureTbl)

SELECT DISTINCT 5050, Variables.SampleMedium, 9999, 1, 1, 0, 2

FROM Variables

WHERE Variables.SampleMedium <> 'Not Relevant' AND Variables.SampleMedium IS NOT NULL

where 5050 is the CalcConcepts.CConceptID of the root “Sample Medium” concept.

Note that certain instances of this facet, ‘Not Relevant’ and NULL, are not included in the CalcConcepts. Faceted search operates on the concept of existence defining meaning – if data exists for a given record in the CalcConcepts, then that is an approved way of accessing that information. Both ‘Not Relevant’ and NULL values were judged to be non-compliant with this concept; if a user wants data for which the observation sample medium is irrelevant, then the user should not specify a sample medium in his search.

### Creating OntologyMapping records to enable faceted search

Now that the ontology elements have been created in the CalcConcepts table, faceted search can be enabled by creating a reference between each of these ontology elements and the series that are described by it. The OntologyMapping table contains two properties to facilitate this reference:

* omSeriesID => the ID of a time series record.
* omConceptID => the ID of a concept that references the omSeriesID record.

The following SQL query can be executed to generate this data for information in the existing database (example for the Sample Medium facet):

INSERT INTO OntologyMapping (omSeriesID, omConceptID)

SELECT SeriesID, CConceptID

FROM SeriesCatalog

INNER JOIN Variables on Variables.VariableID = SeriesCatalog.VariableID

INNER JOIN CalcHierarchy ON Variables.SampleMedium = CalcHierarchy.CConceptName

WHERE CalcHierarchy.CParentID = 5050

### Maintaining faceted search in the face of future data changes

The underlying mechanism of this structure for faceted search is that each ontology element, whether it belongs to the original HydroSphere variable name ontology or from one of the other series-describing ontologies that have now been merged into the larger collection of facet-specific ontologies, is considered to be just a tag for SeriesCatalog records. To disassociate a CalcConcept with a series, delete the OntologyMapping record that joins the two IDs. There is no need to ever delete concepts from the CalcConcepts table, because their existence establishes the canonical name, and they do not interfere with the user interface if they do not reference SeriesCatalog records within the spatial, temporal, and ontological constraints defined by the user at any point in the interface.

If a canonical name is split into two concepts, the following procedure should be used:

1. Update the underlying data table referenced by the SeriesCatQryStructure so that the relational join between the series records and the new and old concepts is established.
2. Delete the OntologyMappings for the CalcConcepts record that represented the old concept.
3. Add new CalcConcepts record(s) to represent the new canonical name(s).
4. Create new OntologyMapping records between the new canonical names and the SeriesCatalog records referenced by them as specified in the above section “Creating OntologyMapping records to enable faceted search.”

When new SeriesCatalog records are added, OntologyMapping records can be generated for just the new SeriesCatalog records by adding filter clauses to the OntologyMapping creation query:

INSERT INTO OntologyMapping (omSeriesID, omConceptID)

SELECT SeriesID, CConceptID

FROM SeriesCatalog

INNER JOIN Variables on Variables.VariableID = SeriesCatalog.VariableID

INNER JOIN CalcHierarchy ON Variables.SampleMedium = CalcHierarchy.CConceptName

WHERE CalcHierarchy.CParentID = 5050 AND SeriesCatalog.SeriesID >= @FirstRecordOfNewSeriesCatalogEntries

These queries will have to be run for each facet in the faceted search. Flat hierarchies are the easiest to maintain because all leaves can be referenced by specifying a single parent ID. Hierarchies with additional nesting should employ OR clauses in the SQL filter statement to ensure that all leaves of the ontology are included in the update.

### Facets derived from continuous properties

MultiFacetedHISSvc introduces a new mechanism for filtering continuous numerical variables, and implements it on the Valuecount property of the SeriesCatalog table. There are some properties where querying by the exact property is not more useful than querying for all matches within a range. Computationally, it requires less buffer memory to process a single record representing a range than to search over all instances of the SeriesCatalog itself, as is implemented with spatial and temporal filters in the current search design. To facilitate this query pattern, MultiFacetedHISSvc introduces canonical names that represent a transformation of the actual data, such that data matching a given set of criteria are considered to be a part of a canonical name. The query that created the OntologyMapping records for the ValueCount facet of this type employs the case statement as follows:

INSERT INTO OntologyMapping (omSeriesID, omConceptID)

SELECT SeriesCatalog.SeriesID,

CASE

WHEN Valuecount < 1 THEN 21468/\* 'Unknown' \*/

WHEN Valuecount < 11 THEN 21469 /\*'1-10' \*/

WHEN Valuecount < 101 THEN 21470 /\*'11-100' \*/

WHEN Valuecount < 1001 THEN 21471 /\*'101-1000' \*/

WHEN Valuecount < 5001 THEN 21472 /\*'1001-5000' \*/

WHEN Valuecount < 10001 THEN 21473 /\*'5001-10000' \*/

WHEN Valuecount < 50001 THEN 21474 /\*'10001-50000' \*/

WHEN Valuecount < 100001 THEN 21475 /\*'50001-100000' \*/

ELSE 21476 /\*'100001+' \*/

END AS ConceptID

FROM SeriesCatalog

The ConceptIDs specified as the outputs of the CASE statement reflect the CalcConcepts records that represent the OntologyElements that match the range specified in the commented code.

It is possible to use this mechanism to implement other kinds of data models to derive facets. For example, a search of locality to a stream reach could be used to tag series as being associated with a particular stream reach.

## Observed Database Behaviors

The MultiFacetedHIS search services have been developed and tested on a 64-bit Windows 7 laptop with 6 Gb of RAM, Intel i7-2630 CPU @ 2.00 GHz, and a 5.8 Windows Experience Index score. Furthermore, they have been deployed on a shared server on the Tufts EECS systems, where benchmarking of the latest system is under way.

By utilizing indexes, dedicated key-tables, and efficient query design, this implementation of Faceted Search has been tweaked to maximize the responsiveness of a search based on spatial-temporal constraints and selections of ontology instances. However, performance on the testing systems has been shown to be highly dependent on the buffer cache and procedural cache of the SQL Server engine supporting the search. Using the commands “dbcc dropcleanbuffers” and “dbcc freeproccache” increases the response time for the initial faceted search query for the state of California (Appendix A) from an average of 14 seconds to an average of 84 seconds on the development machine. While a dedicated database server would likely not run into this issue, production web services should be configured to handle this query duration in their timeout parameters. The prototype’s client-side behavior returns an error message in this case suggesting the user retry the query. Because the first effort typically restores the database server’s cache, retrying the search procedure typically returns a satisfactory result.

## Web Services

The XML documents of these web services are included in the Appendix of this paper.

MultiFacetedHISSvc exposes endpoints designed to facilitate the following operations:

* “Initialize” – return the entire collection of CUAHSI HIS ontology elements and their associated synonyms.
* “Next” – commit the spatial, temporal, and (optionally) ontological constraints that the user has selected on the user interface. Committing these constraints returns the next set of choices for further filtering the search. Only choices that reference data that is referenced by the committed choices are shown to the user.
* “Search” – commit the spatial, temporal, and (optionally) ontological constraints that the user has selected on the user interface, and return the time series that match these selections.

MultiFacetedHISSvc exposes the following endpoints:

* GetAllTypedOntologyElements() – returns the entire collection of ontology elements and synonyms . Used to pre-cache the ontology on the client for faster browsing and autocomplete capability.
* GetTypedOntologyElementsGivenConstraints(List<OntologyElement> Selections, DateTime BeginTime, DateTime EndTime, double MinY, double MaxY, double MinX, double MaxX, Boolean IncludeSpatialResults) – returns the collection of ontology elements and synonyms that reference the dataset constrained by the spatial, temporal, and ontology constraints specified in the input parameters. This is the functional addition necessary to support faceted search.
* ConductFacetedSearch(List<OntologyElement> Selections, DateTime BeginTime, DateTime EndTime, double MinY, double MaxY, double MinX, double MaxX) – returns the SeriesCatalog records and affiliated data for the dataset constrained by the spatial, temporal, and ontology constraints specified in the input parameters. The properties of the return comply with HydroDesktop’s Data Download Plugin specification (<http://hydrodesktop.codeplex.com/Download/AttachmentDownload.ashx?ProjectName=hydrodesktop&WorkItemId=8204&FileAttachmentId=261723>).
* GetSQLOfNextQuery(List<OntologyElement> Selections, DateTime BeginTime, DateTime EndTime, double MinY, double MaxY, double MinX, double MaxX, Boolean IncludeSpatialResults) – returns the SQL string that would result from choosing “Next” on the user interface. Useful for debugging and understanding the functionality of the service, but should probably be excluded from a production implementation.
* GetSQLOfSearchQuery(List<OntologyElement> Selections, DateTime BeginTime, DateTime EndTime, double MinY, double MaxY, double MinX, double MaxX, Boolean) – reutrns the SQL string that would result from choosing “Search” on the user interface. Useful for debugging, but should probably be excluded from a production implementation.

Other endpoints exist in the MultiFacetedHISSvc, however, they are either building blocks for future research projects or ready to become deprecated.

MultiFacetedHISSvc is a SOAP-enabled web service that uses several custom types in its SOAP envelope. These custom types, including SiteData for spatial points, OntologyElement for ontology instances, and SeriesCatalogResult for search records compliant with HydroDesktop’s Data Download Plugin specification, are simple data transfer objects that could be replaced with another transfer mechanism. MultiFacedtedHISSvc is built to run on the .NET 4 Framework. One caveat of the web services implementation is that the DataContractSerializer does not seem to respond to OperationBehavior settings stored in Web.Config as it typically would. A workaround has been implemented on the client-side (CUAHSISearchForm.ConfigureCUAHSIChannelFactory) to set the MaxItemsInObjectGraph behavior property to allow for the message size the SOAP service returns to be serialized correctly.

In its current implementation, security has not been a focus of this web service (although it only currently exposes read endpoints, so there is less risk to mitigate).

# Client-side

The faceted search UI developed for this project uses a wizard-like interface to help the user specify the spatial, temporal, and conceptual constraints of the search. At the outset of the form load, the entire set of existing OntologyElements (CalcConcepts records) are loaded into the UI memory via a call to GetAllTypedOntologyElements(). This allows the GetOntologyElementsGivenConstraints() web service method to return only those OntologyElements referenced by the SeriesCatalog records referenced by the OntologyElements, spatial, and temporal constraints passed as parameters in the web service call, letting the UI construct the necessary checktree or other visual display structure by filling in parent OntologyElements as desired. This design mostly circumvents the difficulty of implementing self-referencing hierarchies in a relational database table, which can result in the need for n inner joins to return the entire tree, where n is the number of levels in the hierarchy.

Once the user has specified spatial constraints (selected at least one shape on the map) and temporal constraints (entered a date/time range in the input boxes), clicking “Begin Faceted Search” calls the GetOntologyElementsGivenConstraints() web service with parameters for the Lat/Long bounding box and Begin and End DateTimes specified, and with an empty set of selected OntologyElements. The return is a set of OntologyElements containing all facet instances across all available facets that describe the SeriesCatalog records in HIS Central that match the spatial and temporal constraints. This information, along with the cached set of the entire OntologyElement corpus is put into a SearchFacetSpecifier control, which renders a treeview of the remaining facets the user can choose to bound the search.

The SearchFacetSpecifier allows the user to select OntologyElement instances, and either further filter the search via the “Next” button, or return the SeriesCatalog search results as a HydroDesktop search layer. In this non-production version, additional buttons and web services for viewing the SQL string that is generated by the combination of query constraints the user has entered exist as well. These web service methods and SearchFacetSpecifier buttons should be removed prior to public release. Invoking a “Next” button on a SearchFacetSpecifier instance collects all of the selected OntologyElements in all SearchFacetSpecifier instances on the FacetedSearchForm, and makes a GetOntologyElementsGivenConstraints() web services request with the collected information. The return of this call is the set of remaining, relevant OntologyElements, which are rendered in a new SearchFacetSpecifier and shown to the right of the first SearchFacetSpecifier. In this way, the user constrains the dataset from left to right, always able to search from any point in the structure, and always able to back up and start over.

### Known Issues of Client Side Implementation

The major issue of this implementation is that HydroDesktop requires the addition of the CUAHSIFacetedSearch service reference in order to allow the faceted search SOAP service to be called by the Faceted Search Plugin. This behavior is a symptom of the .NET security model, in which a child .dll cannot invoke a service reference unless its parent .dll also references it. A path for HydroDesktop plugins to use service references without modifying the main project should be determined to maintain the integrity of the plugin architecture. The workaround is described in Appendix C.

The code of the client UI requires a measure of refactoring, as the form methods of both the SearchFacetSpecifier and the FacetedSearchForm contain some of the business logic, rather than a clean separation between form code and business logic code via a data object as requested by the HydroDesktop coding practices.

Furthermore, the auto-complete functionality of the SearchFacetSpecifier relies on Win32 API calls to make the TreeView control that displays the OntologyElements scroll to the top and left after completing an autocomplete action. This was the only way I could figure out how to implement this behavior in .NET 4. I mention it here because it may have portability implications, and I want to highlight it.

## Areas of Needed Functional Improvement

The spatial filter implemented by MultiFacetedHISSvc currently employs a bounding box search, in which the extents are the minimum box required to contain all selected regions of the active search layer. Production search services have already improved on this method, and they should be incorporated into the spatial filter employed by Faceted Search. This detail was not seen as a focus of this research. These changes will affect both client and server operations.

Additional testing is needed to determine whether minimum constraints should be required for a “Search” action to be conducted. Searching across a sufficiently large region in space and time can produce an unwieldly-large resultset, which taxes server resources and often ends in a service-layer timeout. Furthermore, a dimensional constraint would encourage the use of faceted search features.

# Appendix A - Initial query for the state of California bounding-box without any ontology instances selected:

SELECT DISTINCT OntologyMapping.omConceptID, CalcConcepts.CParentID, CalcConcepts.CConceptName, Synonyms.Synonym, CalcConcepts.Layer, CalcConcepts.IsSearchable, CalcConcepts.IsLeaf, CalcConcepts.IsRoot, CalcConcepts.QueryStructureTbl

FROM OntologyMapping

INNER JOIN

(SELECT DISTINCT omSeriesID, SitesTbl.SiteID, SitesTbl.Latitude, SitesTbl.Longitude from ontologymapping

INNER JOIN

(SELECT SCIID, VariableID, SiteID, NetworkID, BeginDateTime, EndDateTime

FROM SeriesIndexTable

WHERE SeriesIndexTable.BeginDateTime > '1/1/1911 1:00:00 AM' AND SeriesIndexTable.EndDateTime < '10/5/2011 4:09:42 PM' ) SCITbl

ON OntologyMapping.omSeriesID = SCITbl.SCIID

INNER JOIN

(SELECT Sites.Latitude, Sites.SiteID, Sites.Longitude

FROM Sites

WHERE (Sites.Latitude BETWEEN 32.5333633489217 AND 42.0007699651706 AND Sites.Longitude BETWEEN -124.371653612142 AND -114.125018073364)) SitesTbl

ON SitesTbl.SiteID = SCITbl.SiteID

) SeriesList ON SeriesList.omSeriesID = OntologyMapping.omSeriesID

INNER JOIN CalcConcepts

ON CalcConcepts.CConceptID = OntologyMapping.omConceptID

LEFT JOIN Synonyms

ON Synonyms.ConceptID = CalcConcepts.CConceptID

SELECT DISTINCT OntologyMapping.omConceptID, CalcConcepts.CParentID, CalcConcepts.CConceptName, Synonyms.Synonym, CalcConcepts.Layer, CalcConcepts.IsSearchable, CalcConcepts.IsLeaf, CalcConcepts.IsRoot, CalcConcepts.QueryStructureTbl

FROM OntologyMapping

INNER JOIN

(SELECT DISTINCT omSeriesID, SitesTbl.SiteID, SitesTbl.Latitude, SitesTbl.Longitude from ontologymapping

INNER JOIN

(SELECT SCIID, VariableID, SiteID, NetworkID, BeginDateTime, EndDateTime

FROM SeriesIndexTable

WHERE SeriesIndexTable.BeginDateTime > '1/1/1911 1:00:00 AM' AND SeriesIndexTable.EndDateTime < '10/5/2011 4:09:42 PM' ) SCITbl

ON OntologyMapping.omSeriesID = SCITbl.SCIID

INNER JOIN

(SELECT Sites.Latitude, Sites.SiteID, Sites.Longitude

FROM Sites

WHERE (Sites.Latitude BETWEEN 32.5333633489217 AND 42.0007699651706 AND Sites.Longitude BETWEEN -124.371653612142 AND -114.125018073364)) SitesTbl

ON SitesTbl.SiteID = SCITbl.SiteID

) SeriesList ON SeriesList.omSeriesID = OntologyMapping.omSeriesID

INNER JOIN CalcConcepts

ON CalcConcepts.CConceptID = OntologyMapping.omConceptID

LEFT JOIN Synonyms

ON Synonyms.ConceptID = CalcConcepts.CConceptID

# Appendix B: GetAllTypedOntologyElements()

**Request:**

<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">

<s:Header>

<Action s:mustUnderstand="1" xmlns="http://schemas.microsoft.com/ws/2005/05/addressing/none">http://tempuri.org/MultiFacetedHISSvc/GetAllTypedOntologyElements</Action>

</s:Header>

<s:Body>

<GetAllTypedOntologyElements xmlns="http://tempuri.org/" />

</s:Body>

</s:Envelope>

**Response (subset of 5366 OntologyElement entities returned):**

<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">

<s:Header />

<s:Body>

<GetAllTypedOntologyElementsResponse xmlns="http://tempuri.org/">

<GetAllTypedOntologyElementsResult xmlns:a="http://schemas.datacontract.org/2004/07/MultiFacetedHIS" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">

<a:OntologyElement>

<a:IsLeaf>false</a:IsLeaf>

<a:IsRoot>false</a:IsRoot>

<a:IsSearchable>false</a:IsSearchable>

<a:Layer>1</a:Layer>

<a:Querystructuretbl>5</a:Querystructuretbl>

<a:Synonym />

<a:cConceptID>1</a:cConceptID>

<a:cConceptName>Hydrosphere</a:cConceptName>

<a:cParentID>6030</a:cParentID>

<a:itsJoinProperty />

<a:latitude>0</a:latitude>

<a:longitude>0</a:longitude>

<a:myJoinProperty />

<a:mySearchProperty />

<a:seriesCatalogDistance>-1</a:seriesCatalogDistance>

<a:tblJoinsTo>-1</a:tblJoinsTo>

<a:tblName />

</a:OntologyElement>

<a:OntologyElement>

<a:IsLeaf>false</a:IsLeaf>

<a:IsRoot>false</a:IsRoot>

<a:IsSearchable>true</a:IsSearchable>

<a:Layer>1</a:Layer>

<a:Querystructuretbl>5</a:Querystructuretbl>

<a:Synonym />

<a:cConceptID>2</a:cConceptID>

<a:cConceptName>Physical</a:cConceptName>

<a:cParentID>1</a:cParentID>

<a:itsJoinProperty />

<a:latitude>0</a:latitude>

<a:longitude>0</a:longitude>

<a:myJoinProperty />

<a:mySearchProperty />

<a:seriesCatalogDistance>-1</a:seriesCatalogDistance>

<a:tblJoinsTo>-1</a:tblJoinsTo>

<a:tblName />

</a:OntologyElement>

…

</GetAllTypedOntologyElementsResult>

</GetAllTypedOntologyElementsResponse>

</s:Body>

</s:Envelope>

# Appendix C: Workaround for Developing a Plugin with Web Services

Add the following XML to your HydroDesktop.exe.config file in the same directory as your HydroDesktop.exe application:

<system.serviceModel>

<bindings>

<basicHttpBinding>

<binding name="BasicHttpBinding\_MultiFacetedHISSvc" closeTimeout="00:05:00"

openTimeout="00:05:00" receiveTimeout="00:10:00" sendTimeout="00:05:00"

allowCookies="false" bypassProxyOnLocal="false" hostNameComparisonMode="StrongWildcard"

maxBufferSize="2147483647" maxBufferPoolSize="2147483647"

maxReceivedMessageSize="2147483647" messageEncoding="Text"

textEncoding="utf-8" transferMode="Buffered" useDefaultWebProxy="true">

<readerQuotas maxDepth="32" maxStringContentLength="2147483647"

maxArrayLength="2147483647" maxBytesPerRead="2147483647" maxNameTableCharCount="2147483647" />

<security mode="None">

<transport clientCredentialType="None" proxyCredentialType="None"

realm="" />

<message clientCredentialType="UserName" algorithmSuite="Default" />

</security>

</binding>

</basicHttpBinding>

</bindings>

<client>

<endpoint address="http://localhost/FacetedSearch/MultiFacetedHISSvc.svc"

binding="basicHttpBinding" bindingConfiguration="BasicHttpBinding\_MultiFacetedHISSvc"

contract="CUAHSIFacetedSearch.MultiFacetedHISSvc" name="BasicHttpBinding\_MultiFacetedHISSvc" />

</client>

</system.serviceModel>