Search Web Services - searchRetrieve Operation: Binding for SRU 1.2 Version 1.0

Committee Draft 01

30 June 2008

Specification URIs:
This Version:
http://docs.oasis-open.org/search-ws/june08releases/sru-1-2-V1.0-cd-01.doc (Authoritative)
http://docs.oasis-open.org/search-ws/june08releases/sru-1-2-V1.0-cd-01.pdf
http://docs.oasis-open.org/search-ws/june08releases/sru-1-2-V1.0-cd-01.html

Latest Version:
http://docs.oasis-open.org/search-ws/v1.0/sru-1-2-V1.0.doc
http://docs.oasis-open.org/search-ws/v1.0/sru-1-2-V1.0.pdf
http://docs.oasis-open.org/search-ws/v1.0/sru-1-2-V1.0.html

Technical Committee:
OASIS Search Web Services TC

Chair(s):
Ray Denenberg <rden@loc.gov>
Matthew Dovey <m.dovey@jisc.ac.uk>

Editor(s):
Ray Denenberg rden@loc.gov
Larry Dixson ldix@loc.gov
Matthew Dovey m.dovey@jisc.ac.uk
Janifer Gatenby Janifer.Gatenby@oclc.org
Ralph LeVan levan@oclc.org
Ashley Sanders a.sanders@MANCHESTER.AC.UK
Rob Sanderson azaroth@liverpool.ac.uk

Related work:
This specification is related to:

- Search Retrieve via URL (SRU)

Abstract:
This is a binding of the Search Web Services - searchRetrieve operation – Abstract Protocol Definition. This binding is the specification of SRU 1.2.

Status:
This document was last revised or approved by the OASIS Search Web Services TC on the above date. The level of approval is also listed above. Check the “Latest Version” or “Latest Approved Version” location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee’s email list. Others should send comments to the Technical Committee by using the

"Send A Comment" button on the Technical Committee's web page at http://www.oasis-open.org/committees/search-ws

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (http://www.oasis-open.org/committees/search-ws/ipr.php.

The non-normative errata page for this specification is located at http://www.oasis-open.org/committees/search-ws/.

Notices

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The names "OASIS", [insert specific trademarked names and abbreviations here] are trademarks of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see http://www.oasis-open.org/who/trademark.php for above guidance.

Table of Contents

1 Introduction

This is a binding of the OASIS SWS (Search Web Services) searchRetrieve operation – ABSTRACT PROTOCOL DEFINITION.

This binding is the specification of SRU 1.2.

1.1 Terminology

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “NOT RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in **[RFC2119]**. When these words are not capitalized in this document, they are meant in their natural language sense.

1.2 Normative References

[RFC2119] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, http://www.ietf.org/rfc/rfc2119.txt, IETF RFC 2119, March 1997.

2 Overview and Model

2.1 Data model

The data model in the Abstract Protocol Model says that a "d*atastore* is a collection of units of data. Such a unit is referred to as an *abstract item…*".

In this binding:

- A datastore is referred to as a database.
- An abstract item is referred to as an *abstract record*, or *record.*

The Abstract Protocol Model further notes that "Associated with a datastore are one or more formats that the server may apply to an abstract item, resulting in an exportable structure referred to as a *response Item*. . Such a format is referred to as a response item type or *item type*.*."*

In this Binding:

- A response item is referred to as a *response record, or record.*
  Note that "abstract record" and "response record" are referred to as "record" when the meaning is clear from the context.
- An item type is referred to as a *record schema*.

A record schema is an XML schema. All records within a response, as well as the response itself, are transferred in XML. Data is not assumed to be stored in XML; records that are not natively XML must be first transformed into XML before being transferred.

2.2 Processing Model

A client sends a searchRetrieve request to a server. The client formulates the request according to a binding that describes a particular request construction. (An example of such a binding is: *Search Web Services (SWS) Auxiliary Binding for HTTP GET* which describes the construction of an http URL to encode parameter values of the form '*key=value'*.)

The request includes request parameters including a query to be matched against the database at the server. The server processes the query, creating a result set of records that match the query.

The request also indicates the desired number of records to be included in the response and includes the identifier of a record schema for transfer of the records in the response. The entire response (including all of the response records) is packaged in the SRU 1.2 response schema.

The response includes records from the result set, diagnostic information, and a result set identifier that the client may use in a subsequent request to retrieve additional records.

2.3 Result Set Model

The result set model for SRU 1.2 is as described in the Abstract Protocol Definition, with the following additional feature:

```
 When a result set record becomes unavailable, if a client then requests that
record, the server is expected to supply a surrogate diagnostic in place of
the record. For example suppose a result set of three records is created,
subsequently the record at position 2 is deleted, and the client then requests
records 1 through 3. The server should supply, in order: record 1, a surrogate
diagnostic for record 2, record 3.
```

2.4 Diagnostic Model

A server supplies diagnostics in the response as appropriate. A diagnostics is *fatal*' or *non-fatal*. A fatal diagnostic is generated when the execution of the request cannot proceed and no entries are available. For example, if the client supplied an invalid query there might be nothing that the server can do. A non-fatal diagnostic is one where processing may be affected but the server can continue. For example if a particular record is not available in the requested schema but others are, the server may return the ones that are available rather than failing the entire request.

Non-fatal diagnostics are further divided into two categories: 'surrogate' and 'non-surrogate'. Surrogate diagnostics take the place of a record (as described in the Result Set Model). Non-surrogate, non-fatal diagnostics are diagnostics saying that while some or all the entries are available, something may have gone wrong; for example the requested sorting algorithm might not be available. Or, it may be just a warning.

To summarize: A surrogate diagnostic replaces a record; a non-surrogate diagnostic refers to the response at large and is supplied in addition and external to the records. A non-surrogate diagnostic may be fatal or non-fatal. So three combinations are possible:

- surrogate, non-fatal diagnostic
- non-surrogate, non-fatal diagnostic
- non-surrogate, fatal diagnostic

(“Fatal, surrogate” is not a valid combination.)

3 Request Parameters

3.1 Request Parameters for this Binding

The following table describes actual request parameters defined in this binding. It combines those that are included from the Abstract Protocol Definition (table 3) and new request parameters (table 5).

. *Table 1: Summary of Request Parameters*

Actual Parameter Name	Occurence	Restrictions/Values/Description
operation	mandatory	The string: 'searchRetrieve'.
version	mandatory	The string ‘1.2’.
query	mandatory	A query expressed in CQL.
startRecord	optional	Positive integer. Default if omitted is 1.
maximumRecords	optional	See maximumRecords.
recordPacking	optional	'string' or 'xml'. Default is 'xml'. See recordPacking.
recordSchema	optional	See recordSchema.
resultSetTTL	optional	See resultSetTTL and resultSetIdleTime.
stylesheet	optional	See stylesheet.
extraRequestData	optional	See extraRequestData, extraResponseData, and extraRecordData

3.2 Relationship of Actual Parameters to Abstract Parameters

3.2.1 Abstract Request Parameters

The following table, copied from the Abstract Protocol Definition, summarizes the abstract parameters of the SWS searchResponse request. Each parameter occurs either in table 3 or table 4.

Table 2: Abstract Request Parameters

Abstract Parameter Name	Description
responseType	e.g. 'text/html', ‘application/atom+xml’ , ‘application/x-sru’
query	The search query of the request.
startPosition	The position within the result set of the first record to be returned.

maximumItems	The number of items requested to be returned.
group	The number of the result group requested to be returned.
responseItemType	e.g. ‘string’, ‘jpeg’, ‘dc’, ‘iso2709’. From list provided by server.
sortOrder	The requested order of the result set.

3.2.2 Abstract/Actual Request Parameters

The following table lists those abstract request parameters that are realized as actual parameters in this binding. See table 1 for their descriptions

Table 3: Abstract/Actual Request Parameters

Abstract Parameter Name	Actual Parameter Name
query	query
startPosition	startRecord
maximumItems	maximumRecords
responseItemType	recordSchema

3.2.3 Abstract/Excluded Request Parameters

The following table summarizes the abstract request parameters that are excluded from this binding, and why they are excluded.

Table 4: Abstract/Excluded Request Parameters

Abstract Parameter Name	Why Excluded
responseType	The response schema for SRU 1.2 is fixed.
group	request by group is not a feature of SRU 1.2.
sortOrder	The sort order is specified within the query.

3.2.4 New Request Parameters

The following table lists actual request parameters defined in this binding that are not defined as abstract parameters. See table 1 for their descriptions.

Table 5: New Request Parameters

Actual Parameter Name
operation
version
recordPacking

resultSetTTL
stylesheet
extraRequestData

100 # 4 Response Elements

101 ### 4.1.1 Actual Response Elements for this Binding

102 The following table describes the top-level XML elements in the response. These correspond to the actual
103 response elements (from tables 8 and 9).

104 *Table 6: Summary of Actual Response Elements*

Actual Element Name	Type	Occurrence	Restrictions/Values/ Description
<version>	*xsd:string*	Mandatory, non-repeatable	` The string ‘1.2’.
<numberOfRecords>	*xsd:integer*	Mandatory, non-repeatable	If the query fails this MUST be 0.
<resultSetId>	*xsd:string*	Optional, non-repeatable	
<resultSetIdleTime>	*xsd:integer*	Optional, non-repeatable	see resultSetTTL and resultSetIdleTime.
<record>	*<record>*	Optional, repeatable	See records
<nextRecordPosition>	*xsd:integer*	Optional, non-repeatable	If there are no remaining records, this field MUST be omitted.
<diagnostics>	<diagnostics> (non-surrogate)	Optional, non-repeatable	see diagnostics
<extraResponseData>	*structured*	Optional, non-repeatable	See extraRequestData, extraResponseData, and extraRecordData
<echoedSearch RetrieveRequest>	*structured*	Optional, non-repeatable	see echoedSearch RetrieveRequest

105 ## 4.2 Relationship of Actual Elements to Abstract Elements

106 ### 4.2.1 Abstract Response Elements

107 The following table, copied from the Abstract Protocol Definition, summarizes the abstract elements of the
108 SWS searchResponse response.

109 *Table 7: Abstract Response Elements*

Abstract Element Name	Description/Reference
numberOfItems	The number of items matched by the query.
numberOfGroups	The number of result groups in the result set.

resultSetId	The identifier for a result set created through the execution of the query.
item	An individual response item (one of possibly many).
nextPosition	The next position within the result set following the final returned item.
nextGroup	The next result group following the group being returned.
diagnostics	Error message and/or diagnostics.
echoedRequest	The server may echo the request back to the client.

4.2.2 Abstract/Actual Response Elements

The following table lists those abstract response elements that are realized as actual elements in this binding. See table 6 for their descriptions.

Table 8: Abstract/Actual Response Elements

Abstract Element Name	Actual Element Name
numberOfItems	numberOfRecords
resultSetId	resultSetId
item`	record
nextPosition	nextRecordPosition
diagnostics	diagnostics
echoedSearchRetrieveRequest	echoedSearchRetrieveRequest

4.2.3 Abstract/Excluded Response Elements

The following table summarizes the abstract response elements that are excluded from this binding, and why they are excluded.

Table 4: Abstract/Excluded Response Elements

Abstract Element Name	Why Excluded
numberOfGroups	request by group is not a feature of SRU 1.2.
nextGroup	request by group is not a feature of SRU 1.2.

4.2.4 New Response Elements

The following table summarizes actual response elements defined in this binding that are not defined as abstract elements. See table 9 for their descriptions.

Table 9: New Response Elements

Actual Element Name
version

resultSetIdleTime
extraResponseData

5 Parameter and Element Descriptions

5.1 maximumRecords

The request parameter maximumRecords is a non-negative integer. It may be omitted; if so the server determines the default value. The server may return less than this number of records, for example if there are fewer matching records than requested, but MUST NOT return more.

5.2 recordPacking

In order that records which are not well formed do not break the entire message, it is possible to request that they be transferred as a single string with the <, > and & characters escaped to their entity forms. Moreover some toolkits may not be able to distinguish record XML from the XML that forms the response. However, some clients may prefer that the records be transferred as XML in order to manipulate them directly with a stylesheet that renders the records and potentially also the user interface.

This distinction is made via the recordPacking parameter in the request. If the value of the parameter is 'string', then the server should escape the record before transferring it. If the value is 'xml', then it should embed the XML directly into the response. Either way, the data is transferred within the 'recordData' field. If the server cannot comply with this packing request, then it MUST return a diagnostic.

5.3 recordSchema

The requestParameter recordSchema is the XML schema of the records to be supplied in the response. The value of the parameter is the short name that the server assigns to the identifier for the schema, as listed in the server's explain file. The default value if not supplied is determined by the server.

For example, for the MODS Schema Version 3.3 the identifier is info:srw/schema/1/mods-v3.3, as shown in the table at http://www.loc.gov/standards/sru/resources/schemas.html and the short name might (but need not) be 'mods'. (Note: schema identifiers are not restricted to those in this table.)

The server MUST supply records in the requested schema only. If the schema is unknown or a record cannot be rendered in that schema, then the server MUST return a diagnostic:

- If the schema is unknown, the server should supply a non-surrogate (fatal) diagnostic: i*nfo:srw/diagnostic/1/66*: "Unknown schema for retrieval".
- If an individual record cannot be rendered in the requested schema, the server should supply a surrogate (non-fatal) diagnostic in place of the record: i*nfo:srw/diagnostic/1/67*: "Record not available in this schema".

5.4 resultSetTTL and resultSetIdleTime

The request parameter resultSetTTL is the time (in seconds) that the client requests that the result set created should be maintained. The server MAY choose not to fulfill this request, and may respond with a different value, via the response element resultSetIdleTime.

resultSetIdleTime is a good-faith estimate by the server of the idle time, in seconds. That is, the server projects (but does not guarantee) that the result set will remain available and unchanged (both in content and order) until there is a period of inactivity exceeding this idle time. The idle time must be a positive integer, and should not be so small that a client cannot realistically reference the result set again. If the server does not intend that the result set be referenced, it should omit the result set identifier in the response.

The response element resultSetIdleTime may be less-than, equal-to, or greater-than the request parameter resultSetTTL, and may be supplied or omitted regardless of whether resultSetTTL is supplied or omitted. Thus the two (from a protocol point of view) are independent.

5.5 Stylesheet

The request parameter ‘stylesheet’ is a URL for a stylesheet. The client requests that the server simply return this URL in the response, in the href attribute of the xml-stylesheet processing instruction before the response xml. (It is likely that the type will be XSL, but not necessarily so.) If the server cannot fulfill this request it must supply a non-surrogate diagnostic .

The purpose is to allow a thin client to turn the response XML into a natively renderable format, often HTML or XHTML. This allows a web browser, or other application capable of rendering stylesheets, to act as a dedicated client without requiring any further application logic.

Example

```
http://z3950.loc.gov:7090/voyager?version=1.2&operation=searchRetrieve
 &stylesheet=/master.xsl&query=dinosaur
```

This requests the server to include the following as beginning of the response:

```
<?xml version="1.0"?>
 <?xml-stylesheet type="text/xsl" href="/master.xsl"?>
 <sru:searchRetrieveResponse ...
```

5.6 records

The response element ‘records’ is a sequence of <record> elements. See Structure of the <Record> Element. Each contains either a record, or a surrogate diagnostic explaining why that record could not be transferred. All records are transferred in XML. Records may be expressed as a single string, or as embedded XML. If a record is transferred as embedded XML, it must be well formed and should be validatible against the record schema.

5.7 diagnostics

The response element ‘diagnostics’ includes one or more non-surrogate diagnostics.

Note: See Diagnostic Model. Non-surrogate diagnostics are distinguished from surrogate diagnostics. The latter occur in the 'records' element of the response (they take the place of the record for which they are a surrogate). Non-surrogate diagnostics, both fatal and non-fatal, occur in the 'diagnostics' element.

5.8 extraRequestData, extraResponseData, and extraRecordData

The request parameter extraRequestData’; response element <extraResponseData>, shown in Response Schema for SRU 1.2; and the <extraRecordData> element listed in Structure of the Record Element, are fields in which additional information may be provided. The mechanism is described in Extensions.

5.9 echoedSearchRetrieveRequest

The response element <echoedSearchRetrieveRequest> is as shown in the example below. Note the two sub-elements <xQuery> and <baseUrl>.

<xQuery> represents an XCQL [reference] rendering of the query.

Note: This has two benefits.

- The client can use XSLT or other XML manipulation to modify the query without having a CQL query parser.
- The server can return extra information specific to the clauses within the query.

<baseURL> allows the client to reconstruct queries by simple concatenation, or retrieve the explain document to fetch additional information such as the title and description to include in the results presented to the user.

```
Echoed Request Example

<echoedSearchRetrieveRequest>
        <version>1.2</version>
        <query>dc.title = dinosaur</query>
        <recordSchema>mods</recordSchema>
       <xQuery>
             <searchClause xmlns="http://www.loc.gov/zing/cql/xcql/">
                   <index>dc.title</index>
                   <relation>
                        <value>=</value>
                  </relation>
                  <term>dinosaur</term>
             </searchClause>
       </xQuery>
      <baseUrl>http://z3950.loc.gov:7090/voyager</baseUrl>
</echoedSearchRetrieveRequest>
```

6 Response Schema for SRU 1.2

An example of a searchRetrieve response:

```
</searchRetrieveResponse>
    <numberOfRecords>10</numberOfRecords>
    <resultSetId>resultA</resultSetId>
    <resultSetIdleTime>180</resultSetIdleTime>
    <records>
            <record>
              record 1 …. See Example Record
            </record>
            <record>
             record 2 …..
            </record>
    </records>
    <nextRecordPosition>3</nextRecordPosition>
    <echoedSearchRetrieveRequest>

         …. see Echoed Request Example

    </echoedSearchRetrieveRequest>

    <diagnostics>
         <diagnostic>

        first non-surrogate diagnostic … see Non Surrogate Diagnostic Example

         </diagnostic>
         <diagnostic>

        second non-surrogate diagnostic

         </diagnostic>
    </diagnostics>

    <extraResponseData>
              see Extension Example
    </extraResponseData>
</searchRetrieveResponse>
```

6.1 Structure of the <Record> Element

Each <record> element is structured into the elements shown in the following table.

Table 3: Structure of the <Record> Element

Element	Type	Occurence	Description
<recordSchema>	*xsd:string*	mandatory	The URI identifier of the XML schema in which the record is encoded. Although the request may use the server's assigned short name, the response must always be the full URI.

<recordPacking>	*xsd:string*	mandatory	'string' or 'xml'.
<recordData>	*<stringOrXmlFragment>*	mandatory	The actual record.
<recordIdentifier>	*xsd:string*	optional	An identifier for the record by which it can unambiguously be retrieved in a subsequent operation. For example via the 'rec.identifier' index in CQL.
<recordPosition>	*xsd:positiveInteger*	optional	The position of the record within the result set.
<extraRecordData>	*,<xmlFragment>*	optional	Any additional information to be transferred with the record.

Example

An example record, in the simple Dublin Core schema, packed as XML:

```
    <record>
           <recordSchema>info:srw/schema/1/dc-v1.1</recordSchema>
           <recordPacking>xml</recordPacking>
           <recordData>
                  <srw_dc:dc xmlns:srw_dc="info:srw/schema/1/dc-v1.1">
                       <dc:title>This is a Sample Record</dc:title>
                  </srw_dc:dc>
         </recordData>
        <recordPosition>1</recordPosition>
        <extraRecordData>
            <rel:score xmlns:rel="info:srw/extensions/2/rel-1.0"> 0.965
</rel:score>
        </extraRecordData>
   </record>
```

7 Diagnostics

7.1 Diagnostic List

Diagnostics for use with SRU 1.2 are listed at http://www.loc.gov/standards/sru/resources/diagnostics-list.html. This diagnostic list has the namespace: info:srw/diagnostic/1. For example, the URI info:srw/diagnostic/1/10 identifies the diagnostic “Query syntax error”.

Diagnostics used in SRU 1.2 need not be limited to this list, nor need this list be used exclusively for SRU 1.2.

7.2 Diagnostic Schema

The diagnostic schema for SRU 1.2 has three elements, 'uri', 'details' and 'message'.

The required 'uri' field is a URI, identifying the particular diagnostic. The 'details' part contains information specific to the diagnostic. The 'message' field contains a human readable message to be displayed. Only the uri field is required, the other two are optional.

The identifier for the diagnostic schema is: info:srw/schema/1/diagnostics-v1.1

Table 3: Elements of the Diagnostic Schema

Element	***Type***	***Occurence***	***Description***
<uri>	*xsd:anyURI*	*Mandatory*	*The diagnostic's identifying URI.*
<details>	*xsd:string*	*Optional*	*Any supplementary information available, often in a format specified by the diagnostic*
<message>	*xsd:string*	*Optional*	*A human readable message to display to the end user. The language and style of this message is determined by the server, and clients should not rely on this text being appropriate for all situations.*

7.3 Diagnostic Examples

7.3.1 Non-Surrogate Example

```
Non-surrogate, fatal diagnostic:
<diagnostics>
     <diagnostic xmlns="http://www.loc.gov/zing/srw/diagnostic/">
              <uri>info:srw/diagnostic/1/38</uri>
             <details>10</details>
            <message>Too many boolean operators, the maximum is 10.
                                Please try a less complex query.</message>
    </diagnostic>
 </diagnostics>
```

7.3.2 Surrogate Example

```
Surrogate, non-fatal diagnostic:
<records>
      <record>
         <recordSchema> info:srw/schema/1/diagnostics-v1.1</recordSchema>
         <recordData>
```

```
            <diagnostic xmlns="http://www.loc.gov/zing/srw/diagnostic/">
                  <uri>info:srw/diagnostic/1/65</uri>
                  <message>Record deleted by another user.</message>
            </diagnostic>
        </recordData>
    </record> ...
</records>
```

8 Extensions

The name for an extension must begin with 'x-' (lower case x followed by hyphen. SRU 1.2 and future versions will never include an official extension with a name beginning with 'x-', so this will never clash with a mainstream extension name.) It is recommended that the extension name be 'x-' followed by an identifier for the namespace for the extension, again followed by a hyphen, followed by the name of the element within the namespace.

```
 example
http://z3950.loc.gov:7090/voyager?...&x-info4-onSearchFail=scan
```

Note that this convention does not guarantee uniqueness since the extension name will not include a full URI. The extension owner should try to make the name as unique as possible. If the namespace is identified by an 'info:srw' URI , then the recommended convention is to name the extension "x-info*NNN*-*XXX*" where *NNN* is the 'info:srw' authority string, and XXX is the name of the extension. Extension names MUST never be assigned with this form except by the proper authority for the given 'info' namespace.

Response

The response may include extraResponseData with any well-formed XML, and hence servers can include namespaced XML fragments within it in order to convey information back to the client. The extension MUST supply a namespace and the element names with which to do this, if feedback to the client is necessary.

```
 example:
<sru:extraResponseData>
 <auth:token xmlns:auth="info:srw/extension/2/auth-1.0">
 277c6d19-3e5d-4f2d-9659-86a77fb2b7c8
 </auth:token>
 </sru:extraResponseData>
```

Semantics: If the server does not understand a piece of information in an extension parameter, it may silently ignore it. This is unlike many other request parameters, where if the server does not implement that particular feature it MUST respond with a diagnostic. If the particular request requires some confirmation that it has been carried out rather than ignored, then the extension designer should include a field in the response. The semantics of parameters in the request may not be modified by extensions, however the semantics of parts of the response may be modified by extensions. The response semantics may be changed in this way only if the client specifically requests the change. Clients should in any case be prepared to receive the regular semantics, as servers are at liberty to ignore extensions.

ExtraResponseData may be sent that is not directly associated with the request. For example it may contain cost information regarding the query or information on the server or database supplying the results. This data must, however, have been requested. As the request may be echoed, the server must be able to transform the elements into their XML form. If it encounters an unrecognized element, the server may either make its best guess as to how to transform the element, or simply not return it at all. It should not, however, add an undefined namespace to the element as this would invalidate the response. If the content of the element is an XML structure, then the extension designer should also specify how to encode this structure in a URL. This may simply be to escape all of the special characters, but the designer could also create a string encoding form with rules as to how to generate the XML in much the same fashion as the relationship between CQL and XCQL.

Acknowledgements

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

Participants:

Kerry Blinco, Australian Department of Education, Employment and Workplace Relations

Ray Denenberg, Library of Congress
Larry Dixson, Library of Congress
Matthew Dovey, JISC
Janifer Gatenby, OCLC/PICS
Ralph LeVan, OCLC
Ashley Sanders, University of Manchester
Rob Sanderson, University of Liverpool

A. SRU/CQL 2.0 Features

Non-normative Annex

A.1 Background

This specification of the SRU 1.2 Binding is a straightforward rendering of and is equivalent to the existing SRU 1.2 specification. It was developed as a proof of concept for the Abstract Protocol Definition (as is the Open Search binding). The CQL Committee Draft is similarly a straightforward rendering of the CQL 1.2 specification.

Now the OASIS SWS Technical Committee turns its attention to substantive changes to SRU and CQL, one of the reasons for the creation of the committee. This annex describes some of the changes that have been contemplated. The list that follows is neither a complete list, nor have any of the proposals in this list been approved at any level. The purpose of this list of proposals is to generate discussion on these and other proposals.

A.2 Proposed Changes for SRU 2.0 and CQL 2.0

A.2.1 Allow Non-XML Record Representations

Although SRU does not require the database to store records as XML, it does require it to transfer the data as XML. Many formats do not map easily into XML, for example multimedia, images and even complex text formats. This proposal is to allow non-xml serialized data in the response, signaled by additional values for the recordPacking parameter. Two possible values are:

- **By Value: recordPacking="base64"**
  In order to include arbitrary data into an XML document, it can be base-64 encoded. To return base-64 encoded data, the recordPacking value should be "base64", and the recordData element should include the base-64 encoded data.
- **By Reference: recordPacking="uri"**
  Instead of including the data directly into the response, the server could simply provide a link to it. In this case, the recordPacking value should be "uri", and the recordData element should include only the URI of the record.

A.2.2 Proximity as Boolean Modifier

The proposal is to deprecate the PROX BOOLEAN operator and instead have a BOOLEAN prox *modifier*, with the same modifiers: 'unit', 'distance' and 'ordered'. The rationale for doing this is twofold.

- Proximity is not very frequently implemented other than as the special case of adjacency (or phrase searching) so it does not warrant a keyword slot when it could instead be a modifier.
- More importantly, it is not currently possible to express negative proximity – the case when two terms are not proximal.

Proximity would then look like:

- `fish and/prox/distance=2/unit=word/ordered chips`
  (fish followed by chips 2 words away)
- `fish not/prox/distance=2/unit=word/ordered chips`
  (fish not followed by chips 2 words away).

A.2.3 Proximity as Relation

Alternatively proximity can be represented as a searchClause:. add a new relation to CQL called ‘window’ allowing full proximity, rather than just adjacency as is currently possible. CQL.’window’ would take the full range of proximity modifiers: ‘distance’, ‘unit’ and ‘ordered’. For example:

- `dc.title window/distance=2/unit=word/ordered "fish chips"`
  (fish followed by chips 2 words away)

This proposal also allows for more than 2 terms to be part of a proximity query (however only as relates to a single index):

- `dc.title window/distance<5/unit=word "fish chips salt"`
  (fish, chips and salt all within a span of 5 words)
- `dc.title window/distance>5/unit=word "fish chips salt"`
  (fish, chips and salt all with at least 5 words between any combination)

A.2.4 Parenthesized Terms

The above window relation does not allow choice. Window is the equivalent of “all” in that regard. For example:

- `dc.title window/distance<5/unit=word ((fish and chips) and (salt or vinegar))`
  (fish and chips and one of salt or vinegar, in a 5 word window)

This is the only way this query can be expressed. And there are other applications for this style of query.

A.2.5 Faceted Searching

Faceted searching is commonly supported by search engines; for example, one might wish to do a search on a database for books about a particular topic and then see how many records there are in different time periods.

The proposed change is to add a new, optional response element for faceted results. In addition there would be a request parameter to request faceted results.

A.2.6 Result Set Size

The proposal is to add a parameter allowing the client to indicate how much effort the server should take to determine or estimate the number of records in the result set. Similarly, the response might include a parameter indicating how accurate is the result-set-size reported.

The server may be able to determine the exact number of records, or provide a realistic estimate, but it may be an expensive process. The server might prefer not go through that process unless the client requests that it do so. Or the client might want to explicitly request that the server go through or not go through that process.

For example, the client might want the first 10 records regardless of how many there are. In that case if the server goes through the process of determining how many records there are, it may go through an expensive process for nothing.

There is also the (special) case where the server cannot determine or estimate the number of records in the result set. In that case it might be useful to have a special value or some way to indicate this condition.

A.2.7 Multiple Query Types

The proposal is to support multiple query types. CQL would be one query type but there could be other query types as well, for example, Parameterized Query and XQuery.

There are a number of possible ways to integrate support for multiple queries into the protocol.

- **Specify the query type via the parameter name**. For example "cql= ....." would be a cql query, "pq=...." would be a Parameterized Query, etc. The list of possible queries could be a controlled list in the standard, or the query types could be registered.
- **A 'queryType' parameter.** The actual query would be contained within a fixed parameter, for example "query". E.g. "queryType=cql&query=……".
- **Via the Explain record.** Let Explain specify the name of the parameter used for each query type. So one server could say that that a CQL query is specified with the "cql" parameter and another server could say that a CQL query is specified with the "query" parameter.

A.2.8 Eliminate the Version and Operation Parameters

The version parameter in SRU is based on the assumption that the same base URL is to be used for multiple versions of the protocol, and SRU prescribes a mechanism to allow different versions to attempt to interoperate. Instead, different base URLs could be used for different versions, base URLs and corresponding versions exposed via explain, and then a client would know the version of a server, so there would never be a mismatch.

Similarly, the operation parameter could be eliminated on the same basis.