# searchRetrieve: Part 3. searchRetrieve Operation: APD Binding for SRU 2.0 Version 1.0

## Committee Specification 01

## 13 April 2012

### Specification URIs

**This version:**

http://docs.oasis-open.org/search-ws/searchRetrieve/v1.0/cs01/part3-sru2.0/searchRetrieve-v1.0-cs01-part3-sru2.0.doc (Authoritative)
http://docs.oasis-open.org/search-ws/searchRetrieve/v1.0/cs01/part3-sru2.0/searchRetrieve-v1.0-cs01-part3-sru2.0.html
http://docs.oasis-open.org/search-ws/searchRetrieve/v1.0/cs01/part3-sru2.0/searchRetrieve-v1.0-cs01-part3-sru2.0.pdf

**Previous version:**

N/A

**Latest version:**

http://docs.oasis-open.org/search-ws/searchRetrieve/v1.0/searchRetrieve-v1.0-part3-sru2.0.doc (Authoritative)
http://docs.oasis-open.org/search-ws/searchRetrieve/v1.0/searchRetrieve-v1.0-part3-sru2.0.html
http://docs.oasis-open.org/search-ws/searchRetrieve/v1.0/searchRetrieve-v1.0-part3-sru2.0.pdf

**Technical Committee:**

OASIS Search Web Services TC

**Chairs:**

Ray Denenberg (rden@loc.gov), Library of Congress
Matthew Dovey (m.dovey@jisc.ac.uk), JISC Executive, University of Bristol

**Editors:**

Ray Denenberg (rden@loc.gov), Library of Congress
Larry Dixson (ldix@loc.gov), Library of Congress
Ralph Levan (levan@oclc.org), OCLC
Janifer Gatenby (Janifer.Gatenby@oclc.org), OCLC
Tony Hammond (t.hammond@nature.com), Nature Publishing Group
Matthew Dovey (m.dovey@jisc.ac.uk), JISC Executive, University of Bristol

**Additional artifacts:**

This prose specification is one component of a Work Product which also includes:

- XML schemas: http://docs.oasis-open.org/search-ws/searchRetrieve/v1.0/cs01/schemas/
- *searchRetrieve: Part 0. Overview Version 1.0*. http://docs.oasis-open.org/search-ws/searchRetrieve/v1.0/cs01/part0-overview/searchRetrieve-v1.0-cs01-part0-overview.html
- *searchRetrieve: Part 1. Abstract Protocol Definition Version 1.0*. http://docs.oasis-open.org/search-ws/searchRetrieve/v1.0/cs01/part1-apd/searchRetrieve-v1.0-cs01-part1-apd.html

- *searchRetrieve: Part 2. searchRetrieve Operation: APD Binding for SRU 1.2 Version 1.0*. http://docs.oasis-open.org/search-ws/searchRetrieve/v1.0/cs01/part2-sru1.2/searchRetrieve-v1.0-cs01-part2-sru1.2.html
- *searchRetrieve: Part 3. searchRetrieve Operation: APD Binding for SRU 2.0 Version 1.0*. (this document) http://docs.oasis-open.org/search-ws/searchRetrieve/v1.0/cs01/part3-sru2.0/searchRetrieve-v1.0-cs01-part3-sru2.0.html
- *searchRetrieve: Part 4. APD Binding for OpenSearch Version 1.0*. http://docs.oasis-open.org/search-ws/searchRetrieve/v1.0/cs01/part4-opensearch/searchRetrieve-v1.0-cs01-part4-opensearch.html
- *searchRetrieve: Part 5. CQL: The Contextual Query Language Version 1.0*. http://docs.oasis-open.org/search-ws/searchRetrieve/v1.0/cs01/part5-cql/searchRetrieve-v1.0-cs01-part5-cql.html
- *searchRetrieve: Part 6. SRU Scan Operation Version 1.0*. http://docs.oasis-open.org/search-ws/searchRetrieve/v1.0/cs01/part6-scan/searchRetrieve-v1.0-cs01-part6-scan.html
- *searchRetrieve: Part 7. SRU Explain Operation Version 1.0*. http://docs.oasis-open.org/search-ws/searchRetrieve/v1.0/cs01/part7-explain/searchRetrieve-v1.0-cs01-part7-explain.html

**Related work:**

This specification is related to:

- Search/Retrieval via URL. The Library of Congress. http://www.loc.gov/standards/sru/

**Abstract:**

This document specifies a binding of the OASIS SWS Abstract Protocol Definition to the specification of version 2.0 of the protocol SRU: Search/Retrieve via URL. This is one of a set of documents for the OASIS Search Web Services (SWS) initiative.

**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" 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).

**Citation format:**

When referencing this specification the following citation format should be used:

**[SearchRetrievePt3]**

*searchRetrieve: Part 3. searchRetrieve Operation: APD Binding for SRU 2.0 Version 1.0*. 13 April 2012. OASIS Committee Specification 01. http://docs.oasis-open.org/search-ws/searchRetrieve/v1.0/cs01/part3-sru2.0/searchRetrieve-v1.0-cs01-part3-sru2.0.html.

# 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 name "OASIS" is a trademark 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 one of a set of documents for the OASIS Search Web Services (SWS) initiative.

**This document, "*SearchRetrieve Operation: Binding for SRU 2.0*" is the specification of the protocol SRU: Search/Retrieve via URL**.

The set of documents includes the Abstract Protocol Definition (APD) for searchRetrieve operation, which presents the model for the SearchRetrieve operation and serves as a guideline for the development of *application protocol bindings* describing the capabilities and general characteristic of a server or search engine, and how it is to be accessed.

The collection of documents also includes three bindings. This document is one of the three.

Scan, a companion protocol to SRU, supports index browsing, to help a user formulate a query. The Scan specification is also one of the documents in this collection.

Finally, the Explain specification, also in this collection, describes a server's Explain file, which provides information for a client to access, query and process results from that server.

The documents in this collection of specifications are:

1. Overview
2. APD
3. SRU1.2
4. SRU2.0 (this document)
5. OpenSearch
6. CQL
7. Scan
8. Explain

## 1.1 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

## 1.2 References

All references for the set of documents in this collection are supplied in the Overview document:

*searchRetrieve: Part 0. Overview Version 1.0*

http://docs.oasis-open.org/search-ws/searchRetrieve/v1.0/csd01/part0-overview/searchRetrieve-v1.0-csd01-part0-overview.doc

## 1.3 Namespace

All XML namespaces for the set of documents in this collection are supplied in the Overview document: *searchRetrieve: Part 0. Overview Version 1.0*

http://docs.oasis-open.org/search-ws/searchRetrieve/v1.0/csd01/part0-overview/searchRetrieve-v1.0-csd01-part0-overview.doc

# 2 Model

## 2.1 Relationship to Abstract Protocol Definition

The APD defines abstract request parameters and abstract response elements. A binding lists those abstract parameters and elements applicable to that binding and indicates the corresponding actual name of the parameter or element to be transmitted in a request or response.

```
Example.
The APD defines the abstract parameter: startPosition as "The position within
the result set of the first item to be returned. "
And this specification refers to that abstract parameter and notes that its
name, as used in this specification is 'startRecord'.  Thus the request
parameter 'startRecord' in this specification represents the abstract
parameter startPosition in the APD.
```

Different bindings may use different names to represent this same abstract parameter, and its semantics may differ across those bindings as the binding models differ. It is the responsibility of the binding to explain these differences in terms of their respective models.

## 2.2 Operation Model

This specification defines the protocol **SRU: Search/Retrieve via URL**. Different bindings may define different protocols for search/retrieve. The SRU protocol defines a request message (sent from an SRU client to an SRU server) and a response message (sent from the server to the client). This transmission of an SRU request followed by an SRU response is called a SearchRetrieve *operation*.

For the SRU protocol, three operations are defined:

1. **SearchRetrieve Operation**. The SearchRetrieve operation is defined by the SRU protocol, which is this specification.
2. **Scan Operation**. Similar to SRU, the Scan protocol defines a request message and a response message. The transmission of a Scan request followed by a Scan response constitutes a Scan operation.
3. Explain Operation. See Explain Model below.

*Note: In earlier versions a searchRetrieve or scan request carried a mandatory operation parameter. In version 2.0, there is no operation parameter for either. See Interoperation with Earlier Versions.*

## 2.3 Data model

A *server* exposes a *database* for access by a remote *client* for purposes of search and retrieval. The database is a collection of units of data, each referred to as an *abstract record*. In this model there is a single database at any given server.

Associated with a database are one or more formats that the server may apply to an abstract record, resulting in an exportable structure referred to as a *response record*.

Note:
The term record is often used in place of "abstract record" or "response record" when the meaning is clear from the context or when the distinction is not important.

Such a format is referred to as a *record schema*. It represents a common understanding shared by the client and server of the information contained in the records of the database, to allow the transfer of that information. It does not represent nor does it constrain the internal representation or storage of that information at the server.

**Relationship of Data Model to Abstract Model**

The data model in the APD says that a "datastore 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 item is referred to as a record.

The APD 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:

- An item type is referred to as a record schema.

## 2.4 Protocol Model

The protocol model assumes these conceptual components:

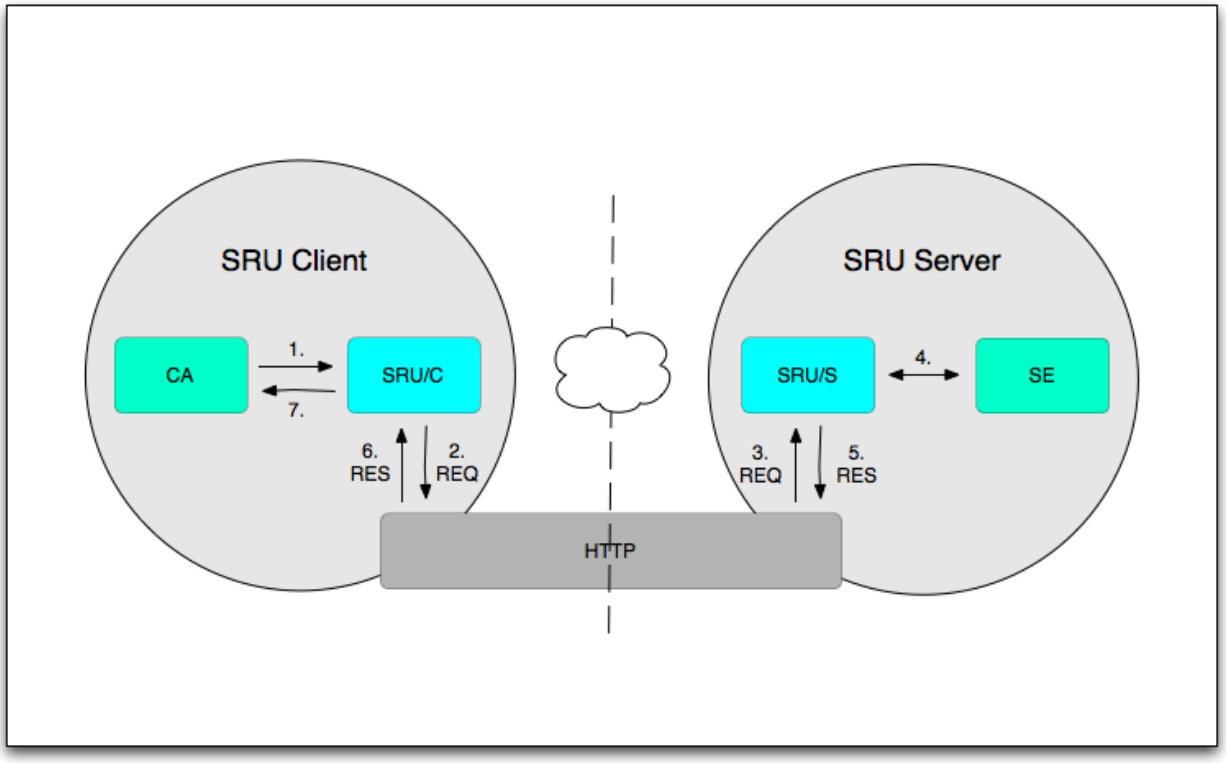
- The client application (**CA**),
- the SRU protocol module at the client (**SRU/C**),
- the lower level protocol (**HTTP**),
- the SRU protocol module at the server (**SRU/S**),
- the search engine at the server (S**E**).

For modeling purposes this standard assumes but does not prescribe bindings between the CA and SRU/C and between SRU/S and SE, as well as between SRU/C and HTTP and between SRU/S and HTTP; for examples of the latter two see Bindings to Lower Level Protocols. The conceptual model of protocol interactions is as follows:

- At the client system the SRU/C accepts a request from the CA, formulates a searchRetrieve protocol request (**REQ**) and passes it to HTTP.
- Subsequently at the server system HTTP passes the request to the SRU/S which interacts with the SE, forms a searchRetrieve protocol response (**RES**), and passes it to the HTTP.
- At the client system, HTTP passes the response to the SRU/C which presents results to the CA.

The protocol model is described diagrammatically in the following picture:




1. CA passes a request to SRU/C.
2. SRU/C formulates a REQ and passes it to HTTP.
3. HTTP passes the REQ to SRU/S.
4. SRU/S interacts with SE to form a RES.
5. The RES is passed to HTTP.
6. HTTP passes the RES to SRU/C.
7. SRU/C presents results to CA.

## 2.5 Processing Model

A client sends a searchRetrieve request to a server. The request includes 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, as well as the identifier of a response schema for transfer of the entire response (including all of the response records).

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.

## 2.6 Query model

Any appropriate query language may be used for SRU version 2.0. Only one in particular is required to be supported: the Contextual Query Language, CQL [4]. The following is intended as only a very cursory overview of CQL's capabilities; for details, consult the CQL specification.

A CQL query consists of a single search clause, or multiple search clauses connected by Boolean operators: AND, OR, or AND-NOT. A search clause may include an index, relation, and search term (or a search term alone where there are rules to infer the index and relation). Thus for example "title = dog" is a search clause in which "title" is the index, "=" is the relation, and "dog" is the search term. "Title = dog

AND subject = cat" is a query consisting of two search clauses linked by a Boolean operator AND, as is "dog AND cat". CQL also supports proximity and sorting. For example, "cat prox/unit=paragraph hat" is a query for records with "cat" and "hat" occurring in the same paragraph. "title = cat sortby author" requests that the results of the query be sorted by author.

## 2.7 Parameter Model

The SRU protocol defines several parameters by name. A searchRetrieve request includes one or more of these parameters and may also include one or more parameters not defined by the protocol.

One of the parameters defined by SRU is named 'query'. Each request includes a query, carried either in the 'query' parameter or collectively in those parameters *not* defined by the protocol.

One reason for modeling parameters in this manner – where parameters may occur in the request that are not defined in the protocol – is to accommodate the case where a query must be conveyed by multiple parameters and it is not feasible to attempt to predict how many parameters. An example might be a forms-based query where each component of the query is carried in a separate parameter. Another reason is to allow a developer of a query type to designate a specific parameter name for that query type. For example a developer might define a query type based on the W3C XQuery specification [7] and designate that it be carried in a parameter named XQuery.

This model aims to provide a simple syntax for well-known query types by providing a default parameter (query) while allowing more complex queries (form-based queries for example) to be supported.

See Query Parameters for details.

## 2.8 Result Set Model

This is a logical model; support of result sets is neither assumed nor required by this standard. There are applications where result sets are critical and applications where result sets are not viable.

When a query is processed, a set of matching records is selected and that set is represented by a result set maintained at the server. The result set, logically, is an ordered list of references to the records. Once created, a result set cannot be modified; any process that would somehow change a result set is viewed logically to instead create a new result set. (For example, an existing result set may be sorted. In that case, the existing result set is logically viewed to be deleted, and a new result set – the sorted set - created.) Each result set is referenced via a unique identifying string, generated by the server when the result set is created.

From the client point of view, the result set is a set of abstract records each referenced by an ordinal number, beginning with 1.The client may request a given record from a result set according to a specific format. For example the client may request record 1 in the Dublin Core format, and subsequently request record 1 in the MODS [7] format. The format in which records are supplied is not a property of the result set, nor is it a property of the abstract records as a member of the result set; the result set is simply the ordered list of abstract records. How the client references a record in the result set is unrelated to how the server may reference it.

The records in a result set are not necessarily ordered according to any specific or predictable scheme. The server determines the order of the result set, unless it has been created with a request that includes a sort specification. (In that case, only the final sorted result set is considered to exist, even if the server internally creates a temporary result set and then sorts it. The unsorted, temporary result set is not considered to have ever existed, for purposes of this model.) In any case, the order must not change. (As noted above, if a result set is created and subsequently sorted, a new result set must be created.)

Thus, suppose an abstract record is deleted or otherwise becomes unavailable while a result set which references that record still exists. This MUST not cause re-ordering. For example, if a client retrieves records 1 through 3, and subsequently record 2 becomes unavailable, if the server again requests record 3, it must be the same record 3 that was returned as record 3 in the earlier operation. ("Same record" does not necessarily mean the same content; the record's content may have changed.) If the server

requests record 2 (no longer available) the server should supply a surrogate diagnostic (see Diagnostic Model) in place of the response record for record 2.

**Relationship of Result Set Model to Abstract Model**

The result set model for SRU 2.0 is as described in the Abstract Protocol Definition, with the following exceptions:

- Addition of the preceding paragraph (beginning with "when a result set record becomes unavailable…)".
- The APD says "A server might support requests by record … or it may instead support requests by group. It may support one form only or both." That sentence has been deleted. In SRU requests are by record; groups are not supported.

## 2.9 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 results 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. See Diagnostics.

## 2.10 Explain Model

Every SRU server provides an associated Explain record. The standard requires that this record be retrievable as the response of an HTTP GET at the base URL for SRU server. The Explain record for a serve may be obtained from other sources as well. An SRU client may retrieve this record which provides information about the server's capabilities. The client may use the information in the Explain record to self-configure and provide an appropriate interface to the user.

The Explain record provides such details as query types supported, CQL context sets (and for each context set indexes supported), diagnostic sets, record schemas, sorting capabilities, specification of defaults, and other details. It also includes sample queries, and conditions of use (for example mandatory display of copyright and syndication rights).

## 2.11 Serialization Model

This specification does not restrict the serialization of the response message and response records. For modeling purposes this document assumes XML serialization. Examples and schemas are portrayed using XML. However non-XML serializations may be used, either for the response message, the response records, or both.

## 2.12 Multi-server search Model

A server might support multi-server searching: sending the query to multiple data sources and consolidating the results into a single result set. From the protocol point of view there is a single server and multi-server searching is for the most part invisible. However there are two areas where multiple data sources may be exposed: faceted search and search result analysis. See the Multi-server search Support subsections of the Faceted Search and Search Result Analysis sections.

# 3 Request Parameters (Summary)

As noted at the beginning of this document, the APD defines abstract request parameters. A binding, such as this specification, lists those abstract parameters and indicates the corresponding actual names of the parameter to be transmitted in a request. Below, the actual parameters for this binding are listed, and following that, the binding of each parameter to its corresponding abstract parameter in the APD.

## 3.1 Actual Request Parameters for this Binding

The following table provides a summary of the actual request parameters defined in this binding, and links to their descriptions.

. Table 1: *Summary of Request Parameters*

| Actual Parameter Name | Occurrence | Reference |
|---|---|---|
| **query** | optional, non-repeatable; must occur if queryType is omitted. | see queryType and query |
| **startRecord** | optional, non-repeatable | see startRecord and maximumRecords |
| **maximumRecords** | optional, non-repeatable | |
| **recordXMLEscaping** (Renamed. Was recordPacking in 1.2.) | optional, non-repeatable | see recordXMLEscaping |
| **recordSchema** | optional, non-repeatable | see recordSchema |
| **resultSetTTL** | optional, non-repeatable | see resultSetTTL |
| **Stylesheet** | optional, non-repeatable | see stylesheet and renderedBy |
| ***Extension parameters*** | optional, see note | see Extensions |
| The following parameters are new in version 2.0 | | |
| **queryType** | optional, non-repeatable; must occur if parameter 'query' is omitted | see queryType and query |
| **sortKeys** | optional, non-repeatable | see sorting |
| ***Facet Parameters*** | Optional, (individually) non-repeatable | see Facet Request Parameters |
| **RenderedBy** | optional, non-repeatable | see stylesheet and renderedBy |
| **httpAccept** | optional, non-repeatable | see httpAccept |
| **responseType** | optional, non-repeatable | see responseType |
| **recordPacking** | optional, non-repeatable | See recordPacking |

*Note: If there is more than one extension parameter they will normally have different names, and so no individual extension parameter will be repeated.*

## 3.2 Relationship of Actual Parameters to Abstract Parameters

### 3.2.1 Abstract Request Parameters

The following table summarizes the relationship of actual parameters to abstract parameters defined in the APD. In the first two columns are shown abstract parameters and their corresponding actual parameters for those abstract parameters that have corresponding actual parameters in this binding. The third column shows abstract parameters for which no corresponding actual parameters are defined for this binding. The fourth column lists new parameters defined for this binding, that is, for which there are no corresponding abstract parameters.

*Table 2: Relationship of actual parameters to abstract parameters*

| **Abstract Parameter** | **Corresponding Actual Parameter** | **Excluded Abstract Parameters** | **Additional Actual Parameters** |
|---|---|---|---|
| responseFormat | httpAccept | | |
| query | query | | |
| startPosition | startRecord | | |
| maximumItems | maximumRecords | | |
| responseItemType | recordSchema | | |
| sortOrder | sortKery | | |
| | | group | |
| | | | queryType |
| | | | recordXMLEscaping |
| | | | *Facet Parameters* |
| | | | resultSetTTL |
| | | | Stylesheet |
| | | | renderedBy |
| | | | *Extension parameters* |
| | | | httpAccept |
| | | | responseType |
| | | | recordPacking |

# 4 Response Elements (Summary)

The APD defines abstract response elements. Binding list those abstract elements and indicate the corresponding actual names of the parameter to be transmitted in a response. Below, the actual elements for this binding are listed, and following that, the binding of each elements to its corresponding abstract element in the APD.

## 4.1 Actual Response Elements for this Binding

The following table describes the top-level XML elements in the response.

*Table 3: Summary of Actual Response Elements*

| Actual Element Name | Type | Occurrence | Reference |
|---|---|---|---|
| <numberOfRecords> | xs:integer | optional, non-repeatable | see numberOfRecords |
| <resultSetId> | xs:string | optional, non-repeatable | see resultSetId |
| <records> | structured | optional, non-repeatable | see records |
| <nextRecordPosition> | xs:integer | optional, non-repeatable | see nextRecordPosition |
| <echoedSearch RetrieveRequest> | structured | optional, non-repeatable | see Echoed Request. |
| <diagnostics> | structured | optional, non-repeatable | see Diagnostics (This element applies to non-surrogate diagnostics.) |
| <extraResponseData> | structured | optional, repeatable | see Extensions |
| The following elements are new in version 2.0 | | | |
| <resultSetTTL> | xs:integer | optional, non-repeatable | See resultSetTTL |
| <resultCountPrecision> | xs:string | Optional, repeatable | see resultCountPrecision |
| <facetedResults> | structured | optional, repeatable | see facetedResults |
| <searchResultAnalysis> | structured | optional, repeatable | see searchResultAnalysis |

## 4.2 Relationship of Actual Elements to Abstract Elements

The following table summarizes the relationship of actual elements to abstract elements defined in the APD. In the first two columns are shown abstract elements and their corresponding actual elements for those abstract elements that have corresponding actual elements in this binding. The third column shows abstract elements for which no corresponding actual elements are defined for this binding. The fourth column lists new elements defined for this binding, that is, for which there are no corresponding abstract elements.

*Table 4: Relationship of actual element to abstract elements*

| Abstract Element | Corresponding Actual element | Excluded Abstract Elements | Additional Actual Elements |
|---|---|---|---|
| numberOfItems | numberOfRecords | | |
| resultSetId | resultSetId | | |
| item | record | | |
| nextPosition | nextRecordPosition | | |
| diagnostics | diagnostics | | |
| echoedRequest | echoedSearch RetrieveRequest | | |
| | | numberOfGroups | |
| | | nextGroup | |
| | | | resultCountPrecision |
| | | | facetedResults |
| | | | searchResultAnalysis |
| | | | extraResponseData |

248

# 5 Parameter and Element Descriptions - Summary

All of the parameters and elements are described in the following several sections. This section provides a summary.

- Query Parameters**:**
  - describes parameters queryType and query.
- Result Set Parameters and Elements:
  - Describes parameters startRecord, maximumRecords, resultSetTTL, resultSetIdleTime, and resultCountPrecision; and elements <nextResultPosition>, <resultSetId>, and <numberOfRecords>.
- facets:
  - describes facet request parameters and faceted results.
- Search Result Analysis:
  - describes the searchResultAnalysis element.
- Sorting:
  - describes the sortKeys parameter.
- Diagnostics:
  - describes the <diagnostics> element.
- Extensions
  - describes extension request parameters and response element <extraResponseData>:
- Response and Record Serialization Parameters and Elements:
  - describes parameters recordXMLEscaping, recordSchema, recordPacking, httpAccept (and other accept parameters), and responseType; and elements <records>, <stylesheet>, <renderedBy>
- echoedRequest:
  - describes the element <echoedSearchRetrieveRequest>

# 6 Query Parameters

## 6.1 Parameter queryType

The request parameter **queryType** is a string indicating the query type. It is optional and if omitted the default value is 'cql'. Its value (except in the case of a reserved query type) is a short name as described in Query Type Definition and Short Name.

### 6.1.1 Reserved Values for parameter queryType

The following strings 'are reserved values for queryType.

1. '**cql**' See [4]
2. '**searchTerms**'. When the value of the parameter queryType is 'searchTerms', the query may (but need not) consist of a list of term separated by space (e.g. "cat hat rat"). The server processes the query however it chooses.

### 6.1.2 Query Type Definition and Short Name

If the query type is other than a reserved type, then there must be a definition for that query type. Each SRU server lists supported query types within its explain file, and for each, supplies a URI for the query type's definition, and a short name to be used for the value of the parameter queryType in a request.

The short name need not be unique. For example one developer might define a query type based on the W3C XQuery specification [7], and another developer may also define a query type also based on XQuery, perhaps using a different profile of the XQuery specification. These two query types would have different definitions (and thus different URIs) and if both are supported on the same server would need different short names, e.g. XQuery1 and XQuery2, but if supported on separate servers they could be assigned the same short name, e.g. XQuery.

## 6.2 Parameter query

The parameter '**query**' contains the query when the query type is a reserved query type, or when the query type definition specifies that 'query' is the parameter to contain the query. It must occur if parameter queryType is omitted, in which case it contains a CQL query.

## 6.3 Parameters that Carry the Query

Each query type definition list one or more parameters to carry the query. If more than one is listed, then they collectively carry the query, and the query definition describes how the query is to be assembled from these parameters.

These are parameters that are not defined in the SRU specification - with one possible exception: the definition may state that the query is to be carried in the 'query' parameter (which *is* defined by SRU).

# 7 Result Set Parameters and Elements

## 7.1 startRecord and maximumRecords

The client requests that the server include a range of result set records in the response, beginning with **startRecord** and the number of records supplied is not to exceed **maximumRecords**.

startRecord is a positive integer, optional, and its default if omitted is 1. maximumRecords is a non-negative integer, optional, and if omitted, the server may choose any value.

The server may return less than the number of records specified by maximumRecords, for example if there are fewer matching records than requested, but MUST NOT return more.

## 7.2 numberOfRecords

The server reports the size of the result set via the response element <**numberOfRecords**>. If the query fails, its value MUST be zero.

## 7.3 nextRecordPosition

When the last record in the response is not the last result set record, the response includes the element <**nextRecordPosition>** whose value is the ordinal position of the next result set record following the final returned record. If there are no remaining records, this element MUST be omitted.

## 7.4 resultSetId

The server may supply the identifier of the result set created by the current operation via the response element <**resultSetId**>. Its purpose is to allow the result set to be referenced in a subsequent request.

Note that the SRU protocol does not directly support the ability to reference a result set in a request. Thus In order for an SRU request to reference the result set, that reference must occur within the query, and thus the query language must support this feature. CQL, for example, does support this feature.

## 7.5 resultSetTTL

In the request the client may supply the request parameters r**esultSetTTL,** the result set time to live, specified in seconds. If supplied, the client is suggesting that the result set need exist no longer than the specified time.

In the response the server may supply the response elements <r**esultSetTTL>.** It may be supplied or omitted whether or not the corresponding parameter had been supplied in the request. The value in the response need not agree with the value in the request (if supplied).

<resultSetTTL> if supplied is a good-faith estimate by the server of the result set's time to live (measured from the time that the response is transmitted). The server projects, but does not guarantee the value.

For example suppose the server says in the search response that resultSetTTL is two weeks (1209600). The server projects that the result set will probably disappear after two weeks, although it could disappear anytime before two weeks, or stay available indefinitely.

## 7.6 resultCountPrecision

The response element<**resultCountPrecision**> allows the server to indicate or estimate the accuracy of the result count as reported by <numberOfRecords>. The value is a URI, identifying a term from a controlled vocabulary.

There may be many such vocabularies maintained from which the value of this element may be chosen. There will be one such vocabulary maintained in conjunction with this standard. The following values are

included in the standard vocabulary at the time of publication of this standard and additional values may be added:

- **exact**
  - The server guarantees that the value as reported in <numberOfRecords> is accurate.
- **unknown**
  - The server has no idea what the result count is, and does not want to venture an estimate.
- **estimate**
  - The server does not know the result set count, but offers an estimate.
- **maximum**
  - The value supplied is an estimate of the maximum possible count that the result set will attain.
- **minimum**
  - The server does not know the result count but guarantees that it is at least this large.
- **current**
  - The value supplied is an estimate of the count at the time the response was sent, however the result set may continue to grow.

### 7.6.1 Extensibility

In general the value for parameter resultCountPrecision is a URI. Any URI indentifying a term appropriate for use as a value of this parameter may be used.

### 7.6.2 'info' URI Representation of Value

The values listed above ( 'exact', 'unknown', etc.) are represented by the following URIs:

- info:srw/vocabulary/resultCountPrecision/1/exact
- info:srw/vocabulary/resultCountPrecision/1/unknown
- info:srw/vocabulary/resultCountPrecision/1/estimate
- info:srw/vocabulary/resultCountPrecision/1/maximum
- info:srw/vocabulary/resultCountPrecision/1/minimum
- info:srw/vocabulary/resultCountPrecision/1/current

For these values, the actual parameter value used may be the URI or it may be the term itself. The rule is that whenever the parameter value does not take the form of a URI, then it is assumed to be prefixed by the string 'info:srw/vocabulary/resultCountPrecision/1/'.

In these URIs, the path component '1' is the authority component; '1' refers to the SRU Maintenance Agency. Other authorities will be registered upon request. See http://www.loc.gov/standards/sru/resources/infoURI.html for details. In this manner additional values may be defined for the parameter resultCountPrecision.

The 'info' URI mechanism is not intended to preclude use of other types of URIs to represent values of this parameter.

# 8 Facets

The client may request, and/or the server supply, faceted results for a query: analysis of how the search results are distributed over various categories (or "facets"). For example the analysis may reveal how the results are distributed by author. The user might then refine the query to one particular author among those listed.

## 8.1 Facet Request Parameters

These parameters are used to request (or suppress) the reporting of facet counts.

### 8.1.1 facetLimit Parameter

The maximum number of counts that should be reported per facet field.

The facetLimit parameter can specify a limit on a per field basis, and/or a global limit applying to all fields.

Examples:

1. **facetLimit:=100**
   sets the limit to 100 for any field.
2. **facetLimit:=100:dc.subject|**
   means that the limit is 100 for dc.subject; do not supply facets for any other field.
3. The combination: **facetLimit=10,100:dc.subject**
   means that the limit is 100 for dc.subject and 10 for all other fields.
4. The combination: **facetLimit=10,100:dc.subject,200:dc.title**
   means that the limit is 100 for dc.subject , 200 for dc.title, and 10 for all other fields.
5. The combination: **facetLimit=100:dc.subject,200:dc.title**
   means that the limit is 100 for dc.subject , 200 for dc.title; do not supply facets for any other fields.
6. The combination:**facetLimit= -1,100: dc.subject**
   means that the limit is 100 for dc.subject and is unlimited for all other fields.
7. **facetLimit:=0**
   means do not supply any facets.

The parameter may have zero or one unqualified limit and zero or more qualified limits. An unqualified limit applies to all fields and a qualified limit applies to a specified field. Limits are separated by comma, their values are integers, and a qualified limit is followed by a colon followed by the name of an index.

The value of each limit is an integer whose meaning is as follows:

- If positive, the server may (but is not obligated to) supply facet counts not to exceed the limit.
- If zero, no facet counts are to be supplied.
- A negative value means that unlimited counts may be supplied.

If the facetLimit parameter is omitted entirely, then the server is free to supply facets as it sees fit, and the effect would be the same as if 'facetLimit=-1" had been specified.

**Restriction on index names that may be used in the facetLimit parameter**:
Index names which contain either or both of the characters comma (,) or colon (:) should not be used within the facetLimit expression.
These two characters have been introduced as delimiters within the syntax but both are legal characters in index names. Thus theoretically there could be an index with the name

**'dc.subject,200:dc.title'.**

In that case, the parameter assignment

**facetLimit=100:dc.subject,200:dc.title**

would be ambiguous.
It is recommended that special characters, and in particular the comma and colon, not be used when creating index names.

### 8.1.2 facetStart

An offset into the list of counts, to allow paging. It is 1-based and the default value is 1 (meaning start with the first count). This parameter can be specified on a per field basis.

- **facetStart=10**
  means begin with the 10th count.
- **facetStart:dc.subject=10**
  means begin with the 10th count for dc.subject.

### 8.1.3 facetSort Parameter

The facetSort parameter is a sort specification for the facet results. It is non-repeatable, and has the following components.

- sortBy. One of the following:
  - 'recordCount' (the number of records matching the facet value)
  - 'alphanumeric'
- order. Optional, one of:
  - 'ascending' (default for sortby=alphanumeric)
  - 'descending' (default for sortby=recordCount)
- caseSensitivity. Optional, and meaningful only for 'alphanumeric'. One of:
  - 'caseSensitive'
  - 'caseInsensitive' (default)

**Serialization**

The value of the parameter is a sequence of these components, occurring in the order given, separated by commas. sortBy must occur; either of the other two may be omitted. If caseSensitivity only is omitted, then sortBy and order are separated by a single comma. If order only is omitted, then sortBy and caseSensitivity are separated by two consecutive commas. In any case there is no trailing comma after the last component.

**Examples**

- facetSort=recordCount
- facetSort=alphanumeric,descending
- facetSort=alphanumeric,,caseSensitive
- facetSort=alphanumeric,descending,caseSensitive

If the server is unable to sort according to the request, then it MUST supply an appropriate diagnostic.

### 8.1.4 facetCount Parameter

This parameter may be used to request the facet count for a specific term, instead of the matching facet values. The parameter may be repeated, but should not be used in conjunction with any other facet parameter.

Example:

**facetCount:dc.subject=history**

## 8.2 facetedResults

The server supplies faceted results in the response, via the response element <facetedResults>; see the example below, and see XML for Faceted Results. Results should correspond to the facet parameters supplied:

- If there were no facet parameters in the request, the server may supply whatever faceted results it chooses to, or none at all.
- If the sole facet parameter in the request was “facetLimit=0” then the server MUST NOT supply any faceted results.
- Otherwise, the server SHOULD attempt to supply faceted results according to the facet parameters in the request, as described above.

### 8.2.1 Multi-server search Support for Faceted Search

A server might support multi-server searching, that is, sending the query to multiple data sources and then consolidating the results into a single result set. This is for the most part invisible to the protocol, that is, from the protocol point of view there is a single server. However, for faceted results, multiple data sources may be exposed, as illustrated in the following example. (Multiple data sources may also be exposed for search result analysis.)

### 8.2.2 Faceted results example

For this example there are two data sources: Library of Congress Catalog, and MELVYL. An XML instance and schema are provided in Appendix B.

## Example

In this example, there are two data sources,

- Library of Congress Catalog
- MELVYL

The server supports facet types:

- subject
- author

A request includes the following parameters:

query=nuthatch
facetLimit=10

Which says: for the query "nuthatch" report facet counts for all facet types, not to exceed 10 counts per facet type.

The response includes the following facet counts:

### source

- Library of Congress Catalog
  - Facet counts for subject
    - birds – 15 records
    - nuthatches – 12 records
  - Facet counts for author
    - Davies, Melvyn - 1 record
    - Pravosudov, Vladimir V - 1 record
- MELVYL
  - Facet counts for subject
    - nuthatches – 18 records
    - Sitta carolinensis – 4 records
  - Facet counts for author
    - Deignan, H. G - 2 records
    - Dunbar , Catherine - 1 record
    - Audubon, John James - 1 record

---

# 9 Search Result Analysis

The response may provide analysis of the search results, via element <**searchResultAnalysis**>

## 9.1 Example

Consider for example the query:

```
title=cat and subject=dog or author=frog
```

Sub queries could be:

- title=cat
- subject=dog
- author=frog
- title=cat and subject=dog
- title=cat or author=frog
- subject=dog or author=frog

Sub query analysis would provide information for some or all of these sub queries, including the result count and a request URL. The choice of which sub queries to evaluate is determined by the server and typically based on intermediate results derived during the processing of the query. In this example, one could expect all the single terms to be returned as they needed to be evaluated to process the query. In normal left-to-right processing, the next sub query evaluated would be “title=cat and subject=dog”. Finally, “author=frog” would be ORd into that sub query and returned as the final result.

The above query might contain the following information.

- Subquery: title=cat
  - Results: 9003
  - Request URL: http://www.example.com/sru?query=“title=cat”
- Subquery: subject=dog
  - Results: 2007
  - Request URL: http://www.example.com/sru?query=“subject=dog”
- Subquery: author=frog
  - Results: 100
  - Request URL: http://www.example.com/sru?query=“author=frog”
- Subquery: title=cat and subject =dog
  - Results: 1863
  - Request URL: http://www.example.com/sru?query=“title=cat and subject=dog”

If the example had been “title=cat and (subject=dog or author=frog)” ; and if “title=cat” had resulted in zero documents, some search engines might have stopped processing on the query as no possible values in the remainder of the query would have affected the result of the overall query. In that case, the only sub query returned would have been “title=cat”.

## 9.2 Multi-server search Support for Search Result Analysis

As noted above in Facets, a server might support multi-server searching, invisible to the protocol except in the case of facet results and search result analysis. The above example does not illustrate multiple data sources; however this feature is illustrated in the XML schema and example instance in Annex B (XML for Search Result Analysis).

# 10 Sorting

If the **sortKeys** request parameter is included, it is a request for the server to sort the result set. The sortKeys parameter consists of one or more sort keys, each with sub-parameters described below.

*Note: While sorting is a function of SRU, sorting may or may not also be a function of the query language. So it is possible for a request to include a sort specification at both the protocol level and at the query level. (For example, sorting is a function of CQL. Thus an SRU request may include a CQL query, where both the SRU request and the query include a sort specification.) The server decides how such a request is to be handled. Diagnostic 94 , 95, or 96 may apply.*

## 10.1 Sort Key Sub-parameters

Each sort key has one or more of the following sub-parameters.

- **Path**

  Mandatory. An XPath [6] expression for a tagpath to be used in the sort.

  **xsortSchema**
  Optional. A short name for a URI identifying an XML schema to which the XPath expression applies. (The short name to URI mapping is included in the server's Explain file.) This is a utility schema into which records can be transformed in order to sort them in a particular way.

  It is not (necessarily) the same schema used to supply records in the response, there is a separate parameter, recordSchema, for that. However If sortSchema is omitted, then recordSchema applies.

  For example, if the record has a geographical location in it, it may be desirable to sort the locations in the records from north to south and east to west. This may require transformation into a schema that allows sorting by a convenient coordinate system, rather than lexically on the place name, and this schema may not be available for retrieving the records. So to sort by title, one might specify the xpath of "/record/title" within the Dublin Core schema

- **Ascending**

  Optional. Boolean, default 'true'.

- **caseSensitive**

  Optional. Boolean, default 'false'.

- **missingValue**
  Optional. (Default is 'highValue'). If the supplied XPath is not present within the record (for example if the server is instructed to sort by author, and a record has no author), it will behave in accordance with this value. One of:
  - 'abort'

    Supply a diagnostic saying that the sort could not be performed.
  - 'highValue'

    The server should sort this as if it were the highest possible value.
  - 'lowValue'

    The server should sort this as if it were the lowest possible value.

- 'omit'.
  The server should remove this record from the results.
- A fixed value.
  The server should sort the record as if this value were supplied.

## 10.2 Serialization

The value of the parameter sortKeys is represented as one or more keys, serialized as follows:

- Each key except the last is followed by a space.
- Each individual key is a sequence of key-parameters, occurring in the order given above, separated by commas.
- Key-parameters beyond the first may be supplied with no value (represented by consecutive commas), in which case the server will use the default, except for the last parameter supplied, which must have a value (i.e. the key may not end in a comma).
- The path and schema must be quoted if they contain quotes, commas or spaces. Internal quotes must be escaped with a backslash.
- Boolean parameters are expressed as 1 (true) or 0 (false).

An example of the sortKeys parameter in an SRU URL might be:

&sortKeys=title,onix date,onix,,0

This example asks to sort as follows:

- Primarily by 'title', from the 'onix' schema. ‘ascending’,'caseSensitive' and 'missingValue' are not given, therefore the defaults apply, namely ‘ascending’, 'insensitive' and 'highValue'.
- Secondarily by 'date', also from the 'onix' schema, ascending (as the default), insensitive. 'missingValue' is not given, therefore the default applies, namely 'highValue'.

## 10.3 Failure to Sort

If the server is unable to create a sorted result set according to the request, then it must supply an appropriate diagnostic.

# 11 Diagnostics

Diagnostics are provided in SRU responses both in the response element <**diagnostics**>, and in the response element <records>.

A diagnostics is *fatal* or *non-fatal.* Non-fatal diagnostics are further divided into two categories: *surrogate* and *non-surrogate*. See the diagnostic model; 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 (in element <records>)
- non-surrogate, non-fatal diagnostic (in element <diagnostics>)
- non-surrogate, fatal diagnostic (in element <diagnostics>)

("Fatal, surrogate" is not a valid combination.)

## 11.1 Diagnostic List

See Diagnostics for use with SRU 2.0 . 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 2.0 need not be limited to this list, nor need this list be used exclusively for SRU 2.0.

## 11.2 Diagnostic Format

The format described in this section is based on the XML Schema for Content Type Application/sru+xml and is the default diagnostic schema for SRU 2.0.

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

The 'uri' field is required. Its value 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> | xs:anyURI | Mandatory | The diagnostic's identifying URI. |
| <details> | xs:string | Optional | Any supplementary information available, often in a format specified by the diagnostic |
| <message> | xs: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. |

## 11.3 Examples

These examples are based on the format described above.

### 11.3.1 Non-Surrogate Example

```
Non-surrogate, fatal diagnostic:
<diagnostics>
     <diagnostic xmlns=”info:srw/xmlns/1/sru-2-0-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>
```

### 11.3.2 Surrogate Example

```
Surrogate, non-fatal diagnostic:
<records>

      <record>
         <!– a real record here  -->
     </record>

      <!– surrogate diagnostic record:  -->
   <record>
         <recordSchema> info:srw/schema/1/diagnostics-v1.1</recordSchema>
         <recordData>
              <diagnostic xmlns=" info:srw/xmlns/1/sru-2-0-diagnostic">
                     <uri>info:srw/diagnostic/1/65</uri>
                     <message>Record deleted by another user.</message>
              </diagnostic>
         </recordData>
 </record>

      <record>
         <!–a real record here -->
     </record>

……

<records>
```

# 12 Extensions

Both in the request and in the response, additional information may be provided - in the request by an extension parameter (whose name is constructed as described next) and in the response by the <**extraResponseData**> element.

## 12.1 Extension Request Parameter

An extension parameter takes on the name of the extension. It must begin with 'x-' : lower case x followed by hyphen. (SRU will never define a parameter with a name beginning with 'x-').

The extension definition MUST supply a namespace. It is recommended that the extension name be 'x-' followed by an identifier for the namespace, 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-infoNNN-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.

## 12.2 Extension Response Element: extraResponseData

An extension definition may (but need not) define a response, to be carried via the extraResponseData element. The extension definition indicates the element names, from the extension's namespace, which will carry the response information.

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

## 12.3 Behavior

The response may include extraResponseData for a given extension only if the request included the extension parameter for that extension, and the extension definition prescribes a response. Thus, an SRU response may never include unsolicited extraResponseData. For example the response may contain cost information regarding the query or information on the server or database supplying the results. This data must, however, have been requested.

If the server does not recognize an extension supplied in an extension parameter, it may simply ignore it. (For that matter, even if the server does recognize the extension, it may choose to ignore it.) If the particular request requires some confirmation that it has been carried out rather than ignored, then the extension designer should define a response. There may even be an element defined in the response for the server to indicate that it did recognize the request but did not carry it out (and even an indication why). However, the server is never obliged to include a response. Thus though a response may be included in the definition of an extension, it may never be designated as mandatory.

693 Thus, the semantics of parameters in the request may not be modified by extensions, because the client
694 cannot be assured that the server recognizes the extension. On the other hand, the semantics of parts of
695 the response may be modified by extensions, because the client will be aware that the extension has
696 been invoked, because extensions are always invoked by the client: the response semantics may be
697 changed by an extension only if the client specifically requests the change. Even when a client does
698 request a change in response semantics, it should be prepared to receive regular semantics since
699 servers are at liberty to ignore extensions.

## 700 12.4 Echoing the Extension Request

701 If the server chooses to echo the request ( see echoedRequest) it must be able to transform the
702 extension parameter into XML, properly namespaced (the extension parameter name will not transform to
703 a valid element in the SRU namespace). If it encounters an unrecognized element and cannot determine
704 the namespace, the server may either make its best guess as to how to transform the element, or simply
705 not return it at all. It should not, however, add an undefined namespace to the element as this would
706 invalidate the response.

# 13 Response and Record Serialization Parameters and Elements

## 13.1 recordXMLEscaping

For those responses where records are transferred in XML, 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 request parameter **recordXMLEscaping.** This parameter is defined for requests only in cases when the response records are to be transferred in XML. Its value is 'string' or 'xml' (default is 'xml' if omitted). If the value of the parameter is 'string', then the server should perform the conversion (escape the relevant characters) before transferring records. If the value is 'xml', then it should embed the XML directly into the response. If the server cannot comply with this request, then it MUST return a diagnostic.

## 13.2 recordPacking

*Note: the recordPacking parameter in earlier versions had entirely different semantics. That parameter has been replaced by the above parameter, recordXMLEscaping.*

Data in a record might be viewed as roughly consisting of structure and payload, where the latter is the actual application data. In some cases the data may be served according to a strict schema and in other cases the server may be prepared to offer variations, allowing easier access to the application data without forcing the client to go through the syntactic infrastructure.

This distinction is made via the request parameter **recordPacking.** Its value is 'packed' or 'unpacked' (default is 'packed' if omitted). If the value of the parameter is 'packed', then the client requests that the server should supply records strictly according to the requested schema. If the value is 'unpacked', then the server is free to allow the location of application data to vary within the record.

## 13.3 recordSchema

The request parameter **recordSchema** identifies the 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 (note: schema identifiers are not restricted to those in this table) and the short name might (but need not) be 'mods'. 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, for example info: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, for example: info:srw/diagnostic/1/67: "Record not available in this schema".

## 13.4 httpAccept

The request parameter **httpAccept** may be supplied to indicate the preferred format of the response. The value is an internet media type. For example if the client wants the response to be supplied in the ATOM format, the value of the parameter is 'application/atom+xml'.

The default value for the response type is 'application/sru+xml'.

The intent of the httpAccept parameter can be accomplished with an HTTP Accept header. Servers SHOULD support either mechanism. In either case (via the httpAccept parameter or HTTP Accept header), if the server does not support the requested media type then the server MUST respond with a 406 status code and SHOULD return an HTML message with pointers to that resource in supported media types.

If an SRU server supports multiple media types and uses content negotiation to determine the media type of the response, then the server SHOULD provide a URL in the Content-Location header of the response pointing directly to the response in that mime-type.

For instance, if the client had sent the URL

http://example.org/sru?query=dog

with an Accept header of 'application/rss+xml',

then the server SHOULD return a Content-Location value of http://example.org/sru?query=dog&httpAccept=application/rss+xml. This Content-Location header is returned along with the content itself, presumably in the application/rss+xml format. (It would also be acceptable to return a redirect to that URL instead, but that behavior is not encouraged as it is inefficient.)

The default response type is application/sru+xml. That is, if there is neither an Accept header (or if there is an Accept header of "*") nor an httpAccept parameter, the response should be of media type application/sru+xml, and a corresponding Content-Location header should be returned with the response. For example if the request is

http://example.org/sru?query=dog

a Content-Location header of

http://example.org/sru?query=dog&httpAccept=application/sru+xml

should be returned.

## 13.5 responseType

The request parameter '**responseType**', paired with the internet media type specified for the response (via either the httpAccept parameter or http accept header) determines the schema for the response.

The value of the parameter is a string. It should be bound to a URI via Explain.

This parameter is optional, and the internet media type (either httpAccept or accept header) is also optional, and any combination is valid – media type plus responseType, media type with no responseType, responseType with no media type, or neither.

The parameter need not be supplied in the case where the requested internet media type is a fully specified SRU response format, for example application/sru+xml.

Suppose, however, that the requested media type is, for example, 'application/atom+xml'. This is not a fully specified SRU format; however there is an ATOM extension which is. Within the definition of that extension, a URI is declared ("info:srw/1/response-type/SRU-ATOM-Response-1") for use as the value of this parameter, to indicate that this particular extension is desired as the response format.

When there is a mismatch:

- Media type that is not a fully specified SRU response format and no responseType; or
- responseType but no media type supplied and the media type cannot be determined from the responseType; or
- both the responseType and media type are specified but the responseType is not valid for the given media type; or

- any other incompatibility between responseType and media type;

then the server SHOULD return an http 406 error, "Not Acceptable".

## 13.6 records

The response element <**records**> contains the records and/or surrogate diagnostics, and may be represented differently for different response schemas. For the SRU default response schema, it is a sequence of <record> elements, where each contains either a record, or a surrogate diagnostic explaining why that record could not be transferred.

## 13.7 stylesheet and renderedBy

The request parameter '**stylesheet**' is a URL for a stylesheet, to be used for the display of the response to the user. The value of parameter '**renderedBy**' determines whether the stylesheet is to be rendered by the client or server; its value is 'client' or 'server', respectively. If omitted, the default is 'client'.

### 13.7.1 Client Rendering

If the value of 'renderedBy' is 'client', 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?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 ...
```

### 13.7.2 Server Rendering

If the value of 'renderedBy' is 'server', the client requests that the server format the response according to the specified stylesheet, assuming the default SRU response schema as input to the stylesheet. Typically this would make sense only if the client is requesting (either via the httpAccept parameter or HTTP Accept header) HTML as the format for the response.

# 14 Echoed Request

Very thin clients, such as a web browser with a stylesheet, may not have the facility to have recorded the query that generated the response it has just received. The server may thus echo the request back to the client via the response element <**echoedSearchRetrieveRequest**>. There are no request elements associated with this functionality, the server may choose to include it or not within a response.

<echoedSearchRetrieveRequest> includes subelements corresponding to request parameters, using the same name.

**Echoed Request Example**

```
<echoedSearchRetrieveRequest>
        <query>dc.title = dinosaur</query>
        <recordSchema>mods</recordSchema>
       <xQuery>
              <searchClause xmlns="info:srw/xmlns/1/xcql-2-0-v1”>
                    <index>dc.title</index>
                    <relation>
                         <value>=</value>
                   </relation>
                   <term>dinosaur</term>
             </searchClause>
       </xQuery>
      <baseUrl>http://z3950.loc.gov:7090/voyager</baseUrl>
</echoedSearchRetrieveRequest>
```

In addition to the echoed parameters, note the sub-elements <xQuery> and <baseUrl>.

<xQuery> represents an XCQL rendering of the query. (See XCQL Annex of CQL specification.)

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.

# 15 Conformance

An SRU 2.0 client or server conforms to this standard if it meets the conditions specified in Client Conformance or Server Conformance respectively.

## 15.1 Client Conformance

### 15.1.1 Protocol

The client must implement the protocol model. It must support at least one LLP.
The SRU/C must be able to:

1. Accept a request from the CA.
2. Assign values to parameters and form Search/Retrieve requests according to the procedures described in the standard.
3. Compose an REQ and pass it to HTTP.
4. Accept an RES from HTTP.
5. Decompose the RES and present information from it to the CA.

### 15.1.2 Query

The client must be capable of sending a CQL query. At minimum, level 0 must be supported.

### 15.1.3 Response Format

The client must support the 'application/sru+xml' media type for the response.

### 15.1.4 Diagnostics

The client must support the diagnostic schema and be able to present diagnostics received in an RES to the CA.

### 15.1.5 Explain

The client must be able to retrieve the Explain record.

## 15.2 Server Conformance

### 15.2.1 Protocol

The server must implement the protocol model; it must support at least one LLP.
The SRU/S must be able to:

1. Accept an REQ from HTTP.
2. Decompose the REQ to determine parameter values and interact with the SE as necessary in order to process the request.
3. Assign values to elements and compose an REQ according to the procedures described in the standard.
4. Pass the response to HTTP.

### 15.2.2 Query

The server must support CQL queries. At minimum, level 0 must be supported.

### 15.2.3 Response Format

The server must support Application/sru+xml for the response.

### 15.2.4 Diagnostics

The server must support the diagnostic schema and be able to present diagnostic information received from the SE.

### 15.2.5 Explain

The Explain record describing the server must be available at the base URL.

# Appendix A. Acknowledgements

Acknowlegements are supplied in the Overview document:

*searchRetrieve: Part 0. Overview Version 1.0*

http://docs.oasis-open.org/search-ws/searchRetrieve/v1.0/csd01/part0-overview/searchRetrieve-v1.0-csd01-part0-overview.doc

# Appendix B. SRU 2.0 Bindings to Lower Level Protocol (Normative)

## B.1 Binding to HTTP GET

This annex describes the construction of an SRU 2.0 http: URL to encode parameter values of the form 'key=value'. Support for Unicode characters is described.

### B.1.1 Syntax

The client sends a request via the HTTP GET method. The request is a URI as described in RFC 3986. Specifically it is an HTTP URL of the form:

```
<base URL>?<searchpart>
```

using the standard &-separated key=value encoding for parameters in <searchpart>.

Example

```
Assume:
 -   The base URL is 'z3950.loc.gov:7090'.
 -   The value of parameter 'query' is "dinosaur".

Then the URL would be:
http://z3950.loc.gov:7090/voyager?query=dinosaur

And over the wire goes:
GET /voyager?query=dinosaur HTTP/1.1
Host: z3950.loc.gov:7090
```

### B.1.2 Encoding (Client Procedure)

The following encoding procedure is recommended, in particular, to accommodate Unicode characters (characters from the Universal Character Set, ISO 10646) beyond U+007F, which are not valid in a URI.

1. Convert the value to UTF-8.
2. Percent-encode characters as necessary within the value. See RFC 3986 section 2.1.
3. Construct a URI from the parameter names and encoded values.

Note: In step 2, it is recommended to percent-encode every character in a value that is not in the URI unreserved set, that is, all except alphabetic characters, decimal digits, and the following four special characters: dash (-), period (.), underscore (_), tilde (~). By this procedure some characters may be percent-encoded that do not need to be -- For example '?' occurring in a value does not need to be percent encoded, but it is safe to do so.

### B.1.3 Decoding (Server Procedure)

1. Parse received request based on '?', '&', and '=' into component parts: the base URL, and parameter names and values.
2. For each parameter:
    a. Decode all %-escapes.
    b. Treat the result as a UTF-8 string.

### B.1.4 Example

Consider the following parameter:

query=dc.title =/word kirkegård

The name of the parameter is "query" and the value is "dc.title =/word kirkegård"

Note that the first '=' (following "query") must not be percent encoded as it is used as a URI delimiter; it is not part of a parameter name or value. The second '=' (preceding the '/') must be percent encoded as it is part of a value.

The following characters must be percent encoded:

- the second '=', percent encoded as %3D
- the '/', percent encoded as %2F
- the spaces, percent encoded as %20
- the 'å'. Its UTF-8 representation is C3A5, two octets, and correspondingly it is represented in a URI as two characters percent encoded as %C3%A5.

The resulting parameter to be sent to the server would then be:

query=dc.title%20%3D%2Fword%20kirkeg%C3%A5rd

## B.2 Binding to HTTP POST

Rather than construct a URL, the parameters may be sent via POST.

The Content-type header MUST be set to

**application/x-www-form-urlencoded'**

POST has several benefits over GET. Primarily, the issues with character encoding in URLs are removed, and an explicit character set can be submitted in the Content-type HTTP header. Secondly, very long queries might generate a URL for HTTP GET that is not acceptable by some web servers or client. This length restriction can be avoided by using POST.

The response for SRU via POST is identical to that of SRU via GET.

An example of what might be passed over the wire in the request:

```
POST /voyager HTTP/1.1
Host: z3850.loc.gov:7090
Content-type: application/x-www-form-urlencoded; charset=iso-8859-1
Content-length: 14
query=dinosaur
```

## B.3 Binding to HTTP SOAP

SRU via SOAP is a binding to the SOAP recommendation of the W3C. The benefits of SOAP are the ease of structured extensions, web service facilities such as proxying and request routing, and the potential for better authentication systems.

In this transport, the request is encoded in XML and wrapped in some additional SOAP specific elements. The response is the same XML as SRU via GET or POST, but wrapped in additional SOAP specific elements.

### B.3.1 SOAP Requirements

The specification adheres to the Web Services Interoperability recommendations.

- SOAP version 1.1 is required. Version 1.2 or higher may be supported.
- The service style is 'document/literal'.
- Messages MUST be inline with no multirefs.
- The SOAPAction HTTP header may be present, but should not be required. If present its value MUST be the empty string. It MUST be expressed as:

**SOAPAction: ""**

- As specified by SOAP, for version 1.1 the Content-type header MUST be 'text/xml'. For version 1.2 the header value MUST be 'application/soap+xml'. (End points supporting both versions of SOAP as well as SRU via POST thus have three content-type headers to consider.)

### B.3.2 Parameter Differences

SRU parameters that cannot be transported via the SOAP binding:

- The 'stylesheet' request parameter MUST NOT be sent. SOAP prevents the use of stylesheets to render the response.

### B.3.3 Example SOAP Request

```
<SOAP:Envelope
xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP:Body>
   <SRW:searchRetrieveRequest xmlns:SRW="info:srw/xmlns/1/sru ">
    <SRW:query>dinosaur</SRW:query>
    <SRW:startRecord>1</SRW:startRecord>
    <SRW:maximumRecords>1</SRW:maximumRecords>
    <SRW:recordSchema>info:srw/schema/1/mods-
v3.0</SRW:recordsSchema>
   </SRW:searchRetrieveRequest>
  </SOAP:Body>
</SOAP:Envelope>
```

For WSDL for SOAP support see the Schema Annex.

# Appendix C. Content Type application/sru+xml (Normative)

This Annex describes the media type application/sru+xml, which is the default SRU response format. See http://tools.ietf.org/html/rfc6207

## C.1 Example searchRetrieve Response

The following is an example of a searchRetrieve response supplied in the content type application/sru+xml:

```
<searchRetrieveResponse>
    <numberOfRecords>10</numberOfRecords>
    <resultSetId>resultA</resultSetId>
    <resultSetTTL>180</resultSetTTL>
    <records>
            <record>
              record 1 ….
            </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>
```

## C.2 Structure of the <Record> Element

The response element <records> is a sequence of <record> elements as shown below. 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.

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> | xs: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 use the full URI. |
| <recordXMLEscaping> | xs:string | mandatory | 'string' or 'xml'. |
| <recordData> | <stringOrXmlFragment> | mandatory | The actual record. |
| <recordIdentifier> | xs: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> | xs: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 <records> element with three records:

```
<records>
    <record>
        <recordSchema>info:srw/schema/1/dc-v1.1</recordSchema>
        <recordXMLEscaping>xml</recordXMLEscaping>
        <recordData>
            <srw_dc:dc xsi:schemaLocation="info:srw/schema/1/dc-schema
                http://www.loc.gov/standards/sru/resources/dc-schema.xsd">
                <title>Fay Vincent Oral History Project collection [videorecording] /</title>
                <creator>Vincent, Fay, interviewer.</creator>
                <type>Oral histories. aat</type>
                <language>eng</language>
                <subject>African American baseball players--Interviews.</subject>
            </srw_dc:dc>
        </recordData>
        <recordPosition>1</recordPosition>
    </record>


    <record>
        <recordSchema>info:srw/schema/1/dc-v1.1</recordSchema>
        <recordXMLEscaping>xml</recordXMLEscaping>
        <recordData>
            <srw_dc:dc xsi:schemaLocation="info:srw/schema/1/dc-schema
            http://www.loc.gov/standards/sru/resources/dc-schema.xsd">
                <title>Whitey Ford : a biography /</title>
                <creator>Coverdale, Miles.</creator>
                <type>text</type>
                <publisher>Jefferson, N.C. : McFarland and Co.,</publisher>
                <date>c2006.</date>
                <language>eng</language>
                <description>Includes bibliographical references (p. 233) and index.</description>
                <subject>Ford, Whitey, 1928-</subject>
                <identifier>http://www.loc.gov/catdir/toc/ecip0610/2006009578.html</identifier>
                <identifier>URN:ISBN:0786425148 (pbk. : alk. paper)</identifier>
```

```
                        </srw_dc:dc>
                    </recordData>
                    <recordPosition>2</recordPosition>
                </record>

<record>
                    <recordSchema>info:srw/schema/1/dc-v1.1</recordSchema>
                    <recordXMLEscaping>xml</recordXMLEscaping>
                    <recordData>
                        <srw_dc:dc xsi:schemaLocation="info:srw/schema/1/dc-schema
                         htp://www.loc.gov/standards/sru/resources/dc-schema.xsd">
                            <title>Whitey Ford sings the blues [sound recording] /</title>
                            <creator>Everlast (Musician) prf</creator>
                            <type>sound recording</type>
                            <publisher>New York, NY : Tommy Boy,</publisher>
                            <date>p1998.</date>
                            <language>eng</language>
                            <description>Rap and rock music.</description>
                            <description>Everlast (vocals, guitars, keyboard, scratches); with assisting musicians </description>
                            <description>"Parental advisory, explicit lyrics"--Container.</description>
                            <description>Compact disc.</description>
                            <description>The white boy is back </description>
                            <subject>Rap (Music)</subject>
                            <subject>Rock music--1991-2000.</subject>
                        </srw_dc:dc>
                    </recordData>
                    <recordPosition>3</recordPosition>
                </record>
</records>
```

# Appendix D. Diagnostics for use with SRU 2.0 (Normative)

The diagnostics below are defined for use with the following SRU diagnostic namespace: info:srw/diagnostic/1. The number in the first column identifies the specific diagnostic within that namespace (e.g., diagnostic 2 below is identified by the URI: info:srw/diagnostic/1/2). The “details format” column specifies what should be returned in the details field. If this column is blank, the format is undefined and the server may return whatever it feels appropriate, including nothing.

| General Diagnostics | | | |
|---|---|---|---|
| **Number** | **Description** | | **Details Format** |
| 1 | General system error | note | Debugging information (traceback) |
| 2 | System temporarily unavailable | note | |
| 3 | Authentication error | note | |
| 4 | Unsupported operation | note | |
| 5 | Unsupported version | note | Highest version supported |
| 6 | Unsupported parameter value | note | Name of parameter |
| 7 | Mandatory parameter not supplied | note | Name of missing parameter |
| 8 | Unsupported parameter | note | Name of the unsupported parameter |
| 9 | Unsupported combination of parameters | note | |
| **CQL Diagnostics** | | | |
| **Number** | **Description** | | **Details Format** |
| 10 | Query syntax error | note | |
| 11 | Not used. | | |
| 12 | Too many characters in query | note | Maximum supported |
| 13 | Invalid or unsupported use of parentheses | Note | Character offset to error |

| | | | |
|---|---|---|---|
| 14 | Invalid or unsupported use of quotes | Note | Character offset to error |
| 15 | Unsupported context set | Note | URI or short name of context set |
| 16 | Unsupported index | Note | Name of index |
| 17 | Not used. | | |
| 18 | Unsupported combination of indexes | Note | Space delimited index names |
| 19 | Unsupported relation | Note | Relation |
| 20 | Unsupported relation modifier | Note | Value |
| 21 | Unsupported combination of relation modifers | note | Slash separated relation modifier |
| 22 | Unsupported combination of relation and index | note | Space separated index and relation |
| 23 | Too many characters in term | note | Length of longest term |
| 24 | Unsupported combination of relation and term | note | |
| 25 | Not used. | | |
| 26 | Non special character escaped in term | Note | Character incorrectly escaped |
| 27 | Empty term unsupported | Note | |
| 28 | Masking character not supported | Note | |
| 29 | Masked words too short | Note | Minimum word length |
| 30 | Too many masking characters in term | Note | Maximum number supported |
| 31 | Anchoring character not supported | Note | |
| 32 | Anchoring character in unsupported position | Note | Character offset |
| 33 | Combination of proximity/adjacency and masking characters not supported | Note | |
| 34 | Combination of proximity/adjacency and anchoring characters not supported | Note | |
| 35 | Term contains only stopwords | Note | Value |

| | | | |
|---|---|---|---|
| 36 | Term in invalid format for index or relation | Note | |
| 37 | Unsupported boolean operator | Note | Value |
| 38 | Too many boolean operators in query | Note | Maximum number supported |
| 39 | Proximity not supported | note | |
| 40 | Unsupported proximity relation | Note | Value |
| 41 | Unsupported proximity distance | Note | Value |
| 42 | Unsupported proximity unit | Note | Value |
| 43 | Unsupported proximity ordering | Note | Value |
| 44 | Unsupported combination of proximity modifiers | Note | Slash separated values |
| 45 | Not used. | | |
| 46 | Unsupported boolean modifier | Note | Value |
| 47 | Cannot process query; reason unknown | Note | |
| 48 | Query feature unsupported | Note | Feature |
| 49 | Masking character in unsupported position | note | the rejected term |

## Diagnostics Relating to Result Sets

| Number | Description | | Details Format |
|---|---|---|---|
| 50 | Result sets not supported | note | |
| 51 | Result set does not exist | note | Result set identifier |
| 52 | Result set temporarily unavailable | note | Result set identifier |
| 53 | Result sets only supported for retrieval | note | |
| 54 | Not used. | | |
| 55 | Combination of result sets with search terms not supported | note | |
| 56 | Not used. | | |

| | | | |
|---|---|---|---|
| 57 | Not used. | | |
| 58 | Result set created with unpredictable partial results available | note | |
| 59 | Result set created with valid partial results available | note | |
| 60 | Result set not created: too many matching records | note | Maximum number |
| **Diagnostics Relating to Records** | | | |
| **Number** | **Description** | | **Details Format** |
| 61 | First record position out of range | note | |
| 62 | Not used. | | |
| 63 | Not used. | | |
| 64 | Record temporarily unavailable | note | |
| 65 | Record does not exist | note | |
| 66 | Unknown schema for retrieval | note | Schema URI or short name |
| 67 | Record not available in this schema | note | Schema URI or short name |
| 68 | Not authorized to send record | note | |
| 69 | Not authorized to send record in this schema | note | |
| 70 | Record too large to send | note | Maximum record size |
| 71 | Unsupported recordXMLEscaping value | note | |
| 72 | XPath retrieval unsupported | note | |
| 73 | XPath expression contains unsupported feature | note | Feature |
| 74 | Unable to evaluate XPath expression | note | |
| **Diagnostics Relating to Sorting** | | | |
| **Number** | **Description** | | **Details Format** |
| 80 | Sort not supported | note | |

| | | | |
|---|---|---|---|
| 81 | Not used. | | |
| 82 | Unsupported sort sequence | note | Sequence |
| 83 | Too many records to sort | note | Maximum number supported |
| 84 | Too many sort keys to sort | note | Maximum number supported |
| 85 | Not used. | | |
| 86 | Cannot sort: incompatible record formats | note | |
| 87 | Unsupported schema for sort | note | URI or short name of schema given |
| 88 | Unsupported path for sort | note | XPath |
| 89 | Path unsupported for schema | note | XPath |
| 90 | Unsupported direction | note | Value |
| 91 | Unsupported case | note | Value |
| 92 | Unsupported missing value action | note | Value |
| 93 | Sort ended due to missing value | note | |
| 94 | Sort spec included both in query and protocol: query prevails | | |
| 95 | Sort spec included both in query and protocol: protocol prevails | | |
| 96 | Sort spec included both in query and protocol: error | | |
| **Diagnostics Relating to Explain** | | | |
| **Number** | **Description** | | **Details Format** |
| 100 | Not used. | | |
| 101 | Not used. | | |
| 102 | Not used. | | |
| **Diagnostics relating to Stylesheets** | | | |
| **Number** | **Description** | | **Details Format** |

| | | | |
|---|---|---|---|
| 110 | Stylesheets not supported | note | |
| 111 | Unsupported stylesheet | note | URL of stylesheet |
| **Diagnostics 120-121 reserved for Scan** | | | |

## D.1 Notes

| No. | Cat. | Description | Notes/Examples |
|---|---|---|---|
| 1 | General | General system error | The server returns this error when it is unable to supply a more specific diagnostic. The sever may also optionally supply debugging information. |
| 2 | General | System temporarily unavailable | The server cannot respond right now, perhaps because it's in a maintenance cycle, but will be able to in the future. |
| 3 | General | Authentication error | The request could not be processed due to lack of authentication. |
| 4 | General | Unsupported operation | Currently three operations are defined -- searchRetrieve, explain, and scan. searchRetrieve and explain are mandatory, so this diagnostic would apply only to scan, or in SRU where an undefined operation is sent. |
| 5 | General | Unsupported version | Currently only version 1.1 is defined and so this diagnostic has no meaning. In the future, when another version is defined, for example version 1.2, this diagnostic may be returned when the server receives a request where the version parameter indicates 1.2, and the server doesn't support version 1.2. |
| 6 | general | Unsupported parameter value | This diagnostic might be returned for a searchRetrieve request which includes the recordXMLEscaping parameter with a value of 'xml', when the server does not support that value. The diagnostic might supply the name of parameter, in this case 'recordXMLEscaping'. |
| 7 | General | Mandatory parameter not supplied | This diagnostic might be returned for a searchRetrieve request which omits the query parameter. The diagnostic might supply the name of missing parameter, in this case 'query'. |
| 8 | General | Unsupported Parameter | This diagnostic might be returned for a searchRetrieve request which includes the recordXPath parameter when the server does not support that parameter. The diagnostic might supply the name of unsupported parameter, in this case 'recordXPath'. |
| 9 | General | Unsupported combination of parameter | One of the two parameters, query and queryType, must be included in a request. This diagnostic might be supplied when neither is included. |
| 10 | Query | Query syntax error | The query was invalid, but no information is given for exactly what was wrong with it. Eg. dc.title fox fish (The reason is that fox isn't a valid relation in the default context set, but the |

| | | | |
|---|---|---|---|
| | | | server isn't telling you this for some reason) |
| 12 | Query | Too many characters in query | The length (number of characters) of the query exceeds the maximum length supported by the server. |
| 13 | Query | Invalid or unsupported use of parentheses | The query couldn't be processed due to the use of parentheses. Typically either that they are mismatched, or in the wrong place. Eg. (((fish) or (sword and (b or ) c) |
| 14 | Query | Invalid or unsupported use of quotes | The query couldn't be processed due to the use of quotes. Typically that they are mismatched Eg. "fish' |
| 15 | Query | Unsupported context set | A context set given in the query isn't known to the server. Eg. dc.title any dog |
| 16 | Query | Unsupported index | The index isn't known, possibly within a context set. Eg. dc.author any leVan (dc has a creator index, not author) |
| 18 | Query | Unsupported combination of indexes | The particular use of indexes in a boolean query can't be processed. Eg. The server may not be able to do title queries merged with description queries. |
| 19 | Query | Unsupported relation | A relation in the query is unknown or unsupported. Eg. The server can't handle 'within' searches for dates, but can handle equality searches. |
| 20 | Query | Unsupported relation modifier | A relation modifier in the query is unknown or unsupported by the server. Eg. 'dc.title any/fuzzy starfish' when fuzzy isn't supported. |
| 21 | Query | Unsupported combination of relation modifers | Two (or more) relation modifiers can't be used together. Eg. dc.title any/cql.word/cql.string "star fish" |
| 22 | Query | Unsupported combination of relation and index | While the index and relation are supported, they can't be used together. Eg. dc.author within "1 5" |
| 23 | Query | Too many characters in term | The term is too long. Eg. The server may simply refuse to process a term longer than a given length. |
| 24 | Query | Unsupported combination of relation and term | The relation cannot be used to process the term. Eg dc.title within "dixson" |
| 26 | Query | Non special character escaped in term | Characters may be escaped incorrectly Eg "\a\r\n\s" |
| 27 | Query | Empty term | Some servers do not support the use of an empty term for |

| | | | |
|---|---|---|---|
| | | unsupported | search or for scan. Eg: dc.title > "" |
| 28 | Query | Masking character not supported | A masking character given in the query is not supported. Eg. The server may not support * or ? or both |
| 29 | Query | Masked words too short | The masked words are too short, so the server won't process them as they would likely match too many terms. Eg. dc.title any * |
| 30 | Query | Too many masking characters in term | The query has too many masking characters, so the server won't process them. Eg. dc.title any "???a*f??b* *a?" |
| 31 | Query | Anchoring character not supported | The server doesn't support the anchoring character (^) Eg dc.title = "^jaws" |
| 32 | Query | Anchoring character in unsupported position | The anchoring character appears in an invalid part of the term, typically the middle of a word. Eg dc.title any "fi^sh" |
| 33 | Query | Combination of proximity/adjacency and masking characters not supported | The server cannot handle both adjacency (= relation for words) or proximity (the boolean) in combination with masking characters. Eg. dc.title = "this is a titl* fo? a b*k" |
| 34 | Query | Combination of proximity/adjacency and anchoring characters not supported | Similarly, the server cannot handle anchoring characters. |
| 35 | Query | Term contains only stopwords | If the server does not index words such as 'the' or 'a', and the term consists only of these words, then while there may be records that match, the server cannot find any. Eg. dc.title any "the" |
| 36 | Query | Term in invalid format for index or relation | This might happen when the index is of dates or numbers, but the term given is a word. Eg dc.date > "fish" |
| 37 | Query | Unsupported boolean operator | For cases when the server does not support all of the boolean operators defined by CQL. The most commonly unsupported is Proximity, but could be used for NOT, OR or AND. |
| 38 | Query | Too many boolean operators in query | There were too many search clauses given for the server to process. |
| 39 | Query | Proximity not supported | Proximity is not supported at all. |
| 40 | Query | Unsupported proximity relation | The relation given for the proximity is unsupported. Eg the server can only process = and > was given. |

| | | | |
|---|---|---|---|
| 41 | Query | Unsupported proximity distance | The distance was too big or too small for the server to handle, or didn't make sense. Eg 0 characters or less than 100000 words |
| 42 | Query | Unsupported proximity unit | The unit of proximity is unsupported, possibly because it is not defined. |
| 43 | Query | Unsupported proximity ordering | The server cannot process the requested order or lack thereof for the proximity boolean |
| 44 | Query | Unsupported combination of proximity modifiers | While all of the modifiers are supported individually, this particular combination is not. |
| 46 | Query | Unsupported boolean modifier | A boolean modifier on the request isn't supported. |
| 47 | Query | Cannot process query; reason unknown | The server can't tell (or isn't telling) you why it can't execute the query, maybe it's a bad query or maybe it requests an unsupported capability. |
| 48 | Query | Query feature unsupported | the server is able (contrast with 47) to tell you that something you asked for is not supported. |
| 49 | Query | Masking character in unsupported position | Eg, a server that can handle xyz* but not *xyz or x*yz |
| 50 | result set | Result sets not supported | The server cannot create a persistent result set. |
| 51 | result set | Result set does not exist | The client asked for a result set in the query which does not exist, either because it never did or because it had expired. |
| 52 | result set | Result set temporarily unavailable | The result set exists, it cannot be accessed, but will be able to be accessed again in the future. |
| 53 | result set | Result sets only supported for retrieval | Other operations on results apart from retrieval, such as sorting them or combining them, are not supported. |
| 55 | result set | Combination of result sets with search terms not supported | Existing result sets cannot be combined with new terms to create new result sets. eg cql.resultsetid = foo not dc.title any fish |
| 58 | result set | Result set created with unpredictable partial results available | The result set is not complete, possibly due to the processing being interupted mid way through. Some of the results may not even be matches. |
| 59 | result set | Result set created with valid partial results available | All of the records in the result set are matches, but not all records that should be there are. |

| | | | |
|---|---|---|---|
| 60 | result set | Result set not created: too many matching records | There were too many records to create a persistent result set. |
| 61 | Records | First record position out of range | For example, if the request matches 10 records, but the start position is greater than 10. |
| 64 | Records | Record temporarily unavailable | The record requested cannot be accessed currently, but will be able to be in the future. |
| 65 | Records | Record does not exist | The record does not exist, either because it never did, or because it has subsequently been deleted. |
| 66 | Records | Unknown schema for retrieval | The record schema requested is unknown. Eg. the client asked for MODS when the server can only return simple Dublin Core |
| 67 | records | Record not available in this schema | The record schema is known, but this particular record cannot be transformed into it. |
| 68 | Records | Not authorized to send record | This particular record requires additional authorisation in order to receive it. |
| 69 | Records | Not authorized to send record in this schema | The record can be retrieved in other schemas, but the one requested requires futher authorisation. |
| 70 | Records | Record too large to send | The record is too large to send. |
| 71 | Records | Unsupported recordXMLEscaping value | The server supports only one of string or xml, or the client requested a recordXMLEscaping which is unknown. |
| 72 | Records | XPath retrieval unsupported | The server does not support the retrieval of nodes from within the record. |
| 73 | Records | XPath expression contains unsupported feature | Some aspect of the XPath expression is unsupported. For example, the server might be able to process element nodes, but not functions. |
| 74 | Records | Unable to evaluate XPath expression | The server could not evaluate the expression, either because it was invalid or it lacks some capability. |
| 80 | Sort | Sort not supported | the server cannot perform any sort; that is the server only returns data in the default sequence. |
| 82 | Sort | Unsupported sort sequence | The particular sequence of sort keys is not supported, but the keys may be supported individually. |
| 83 | Sort | Too many records to sort | used when the server will only sort result sets under a certain size and the request returned a set larger than that limit. |
| 84 | Sort | Too many sort keys to sort | the server can accept a sort statement within a request but cannot deliver as requested, e.g. the server can sort by a maximum of 2 keys only such as "title" and "date" but was requested to sort by "title", "author" and "date". |

| | | | |
|---|---|---|---|
| 86 | Sort | Cannot sort: incompatible record formats | The result set includes records in different schemas and there is insufficient commonality among the schemas to enable a sort. |
| 87 | Sort | Unsupported schema for sort | the server does not support sort for records in a particular schema, e.g. it supports sort for records in the DC schema but not in the ONIX schema. |
| 88 | Sort | Unsupported path for sort | the server can accept a sort statement within a request but cannot deliver as requested, e.g. the server can deliver in title or date sequence but subject was requested. |
| 89 | Sort | Path unsupported for schema | The path given cannot be generated for the schema requested. For example asking for /record/fulltext within the simple Dublin Core schema |
| 90 | Sort | Unsupported direction | the server can accept a sort statement within a request but cannot deliver as requested, e.g. the server can deliver in ascending only but descending was requested. |
| 91 | Sort | Unsupported case | the server can accept a sort statement within a request but cannot deliver as requested, e.g. the server's index is single case so sorting case sensitive is unsupported |
| 92 | Sort | Unsupported missing value action | the server can accept a sort statement within a request but cannot deliver as requested. For example, the request includes a constant that the server should use where a record being sorted lacks the data field but the server cannot use the constant to override its normal behavior, e.g. sorting as a high value. |
| 93 | Sort | Sort ended due to missing value | missingValue of 'abort' |
| 110 | Stylesheet | Stylesheets not supported | The SRU server does not support stylesheets, or a stylesheet was requested from an SRW server. |
| 111 | Stylesheet | Unsupported stylesheet | This particular stylesheet is not supported, but others may be. |

1133

# Appendix E. Extensions for Alternative Response Formats (Non Normative)

This Annex supplies examples of SRU response elements using responses with content types other than 'application/sru+xml'. While an SRU response could, in principle, be layered on top of any arbitrary host format based on a given content type it is most likely to be encountered in the familiar syndication formats used in applications such as OpenSearch. This Annex will show how SRU response elements may be mapped onto the following host formats: ATOM, JSON (both JSON and JSONP), and RSS (both 1.0 and 2.0).

It is worth noting that in contrast to the standard SRU response which is constrained by a W3C XML Schema the syndication formats tend to be "open" and are more loosely specified in terms of element ordering. While this has the advantage of making them ideal carrier, or host, formats, it does mean that there is no single canonical way to map SRU responses. What is shown here is a general indication of how SRU response elements (and in particular the SRU record data structures) can be embedded within the host format.

The standard SRU response shown below in outline is used for the examples, where the individual SRU response elements are shown in red, the SRU record data in purple, and the SRU records are shown in bold. The same coloring and bolding is applied to the SRU extension formats to show how an SRU response may be mapped onto those host formats.

```
<sru:searchRetrieveResponse xmlns:srw="http://www.loc.gov/zing/srw/">
  ...
  <sru:numberOfRecords>2</sru:numberOfRecords>
  <sru:records>
    <sru:record>
      <sru:recordSchema>info:srw/schema/1/dc-v1.1</sru:recordSchema>
      <sru:recordXMLEscaping>xml</sru:recordXMLEscaping>
      <sru:recordData>
        <srw_dc:dc ...>
        ...
        </srw_dc:dc>
      </sru:recordData>
    <sru:recordPosition>1</sru:recordPosition>
  </sru:record>
  <sru:record>
    <sru:recordSchema>info:srw/schema/1/dc-v1.1</sru:recordSchema>
    <sru:recordXMLEscaping>xml</sru:recordXMLEscaping>
    <sru:recordData>
      <srw_dc:dc ...>
      ...
      </srw_dc:dc>
      </sru:recordData>
      <sru:recordPosition>2</sru:recordPosition>
    </sru:record>
  </sru:records>
</sru:searchRetrieveResponse>
```

## E.1 ATOM Extension

When the SRU request indicates a preference for SRU response elements within an ATOM response using the mime type 'application/atom+xml' (either through an HTTP 'Accept' header or via an '**httpAccept**' parameter) it may also designate a particular ATOM extension format through the parameter '**responseType**'.

The default extension format for an ATOM response as outlined in the example below is designated by the URI:

- info:srw/1/response-type/atom

Note that other response types may be possible and may be designated with their own 'response-type' URIs. If no '**responseType**' parameter is present then the default response type shown here will be assumed to be requested.

| | |
|---|---|
| Mime type ('Accept' header or '**httpAccept**' param) | application/atom+xml |
| Response type ('**responseType**' param) | info:srw/1/response-type/atom |

**Example - ATOM Response (with SRU Response Elements)**

```
<feed xmlns="http://www.w3.org/2005/Atom" xmlns:sru="info:srw/xmlns/1/sru-2-0-v1">
  <title>SRU Records</title>
  <author>
    <name>Clint Courtney</name>
  </author>
  <id>urn:uuid:b636f3b1-dd07-4b9a-aeb8-d05a1997876e</id>
  <link rel="self"
href="http://z3950.loc.gov:7090/voyager?version=1.1&amp;operation=searchRetrieve&amp;query%3D%22hitchiker's%20guide%22&amp;startRecord%3D1&amp;maximumRecords%3D5&amp;recordSchema%3Ddc"/>
  <updated>2009-12-12T12:00:00Z</updated>
  <sru:numberOfRecords>2</sru:numberOfRecords>
  <entry>
    <id>urn:isbn:1840235012</id>
    <title>Don't panic : Douglas Adams and the hitchiker's guide to the galaxy </title>
    <link href="http://www.example.com/xyz" />
    <updated>2009-12-12T12:00:00Z</updated>
    <sru:recordSchema>info:srw/schema/1/dc-v1.1</sru:recordSchema>
    <sru:recordXMLEscaping>xml</sru:recordXMLEscaping>
    <sru:recordData>
      <srw_dc:dc xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:srw_dc="info:srw/schema/1/dc-schema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="info:srw/schema/1/dc-schema http://www.loc.gov/standards/sru/resources/dc-schema.xsd">
        <dc:title> Don't panic : Douglas Adams and the hitchiker's guide to the galaxy </dc:title>
        <dc:creator>Gaiman, Neil.</dc:creator>
        <dc:type>text</dc:type>
        <dc:publisher>London : Titan,</dc:publisher>
        <dc:date>2002</dc:date>
```

```
        <dc:language>eng</dc:language>
        <dc:subject>Adams, Douglas, 1952-2001.</dc:subject>
        <dc:subject>Adams, Douglas, 1952-2001. Hitch-hiker's guide to the galaxy.</dc:subject>
        <dc:subject>Science fiction, English--History and criticism.</dc:subject>
        <dc:subject>Novelists, English--20th century--Biography.</dc:subject>
        <dc:subject>Prefect, Ford (Fictitious character)</dc:subject>
        <dc:subject>Dent, Arthur (Fictitious character)</dc:subject>
        <dc:identifier>urn:isbn:1840235012</dc:identifier>
      </srw_dc:dc>
    </sru:recordData>
    <sru:recordPosition>1</sru:recordPosition>
  </entry>
  <entry>
    <id>urn:isbn:1868720721</id>
    <title>The hitchiker's guide to the internet : an African handbook /</title>
    <link href="http://www.example.com/abc" />
     <updated>2009-12-12T12:00:00Z</updated>
    <sru:recordSchema>info:srw/schema/1/dc-v1.1</sru:recordSchema>
    <sru:recordXMLEscaping>xml</sru:recordXMLEscaping>
    <sru:recordData>
      <srw_dc:dc xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:srw_dc="info:srw/schema/1/dc-schema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="info:srw/schema/1/dc-
schema http://www.loc.gov/standards/sru/resources/dc-schema.xsd">
        <dc:title>The hitchiker's guide to the internet : an African handbook /</dc:title>
        <dc:creator>Goldstuck, A.</dc:creator>
        <dc:type>text</dc:type>
        <dc:publisher>Johannesburg : [S.N.]</dc:publisher>
        <dc:date>1998</dc:date>
        <dc:language>eng</dc:language>
        <dc:identifier>urn:isbn:1868720721</dc:identifier>
        </srw_dc:dc>
      </sru:recordData>
    <sru:recordPosition>2</sru:recordPosition>
  </entry>
</feed>
```

## E.2 JSON Extension

When the SRU request indicates a preference for SRU response elements within a JSON response using the mime type ‘application/json’ (either through an HTTP ‘Accept’ header or via an ‘**httpAccept**’ parameter) it may also designate a particular JSON extension format through the parameter ‘**responseType**’.

The default extension format for a JSON reponse as outlined in the example below is designated by the URI:

- info:srw/1/response-type/json

Note that other response types may be possible and may be designated with their own 'response-type' URIs. If no '**responseType**' parameter is present then the default response type shown here will be assumed to be requested.

| | |
|---|---|
| Mime type ('Accept' header or '**httpAccept**' param) | application/json |
| Response type ('**responseType**' param) | info:srw/1/response-type/json |

**Example - JSON Response (with SRU Response Elements)**


```
{
   "feed": {
      "title": "SRU Records",
      "author": {
                "name": "Clint Courtney"
      },
      "updated": "2009-12-12T12:00:00Z",
      "id": "urn:uuid:a6852153-dc12-4cd9-b3e0-f9ff2ed7f0b3",
      "link":
"http://z3950.loc.gov:7090/voyager?version=1.1&operation=searchRetrieve&query%3D%22hitchiker's%20guide%22&startRecord%3D1&maximumRecords%3D5&recordSchema%3Ddc",
      "sru:numberOfRecords": 2,
      "entry": [
        {
          "id": "urn:isbn:1840235012",
          "title": "Don't panic : Douglas Adams and the hitchiker's guide to the galaxy",
          "link": "http://www.example.com/xyz",
           "updated": "2009-12-12T12:00:00Z",
          "sru:recordSchema": "info:srw/schema/1/dc-v1.1",
          "sru:recordXMLEscaping": "xml",
          "sru:recordData": {
            "srw_dc:dc": {
              "dc:title": "Don't panic : Douglas Adams and the hitchiker's guide to the galaxy",
              "dc:creator": [
                 "Gaiman, Neil"
              ],
              "dc:type": "text",
              "dc:publisher": "London : Titan",
              "dc:date": "2002",
              "dc:language": "eng",
              "dc:subject": [
                 "Adams, Douglas, 1952-2001.",
                 "Adams, Douglas, 1952-2001. Hitch-hiker's guide to the galaxy",
```

```
                    "Science fiction, English--History and criticism",
                    "Novelists, English--20th century--Biography",
                    "Prefect, Ford (Fictitious character)",
                    "Dent, Arthur (Fictitious character)",
                ],
                "dc:identifier": "urn:isbn:1840235012"
            }
        },
        "sru:recordPosition": 1
    },
    {
        "id": "urn:isbn:1868720721",
        "title": "The hitchiker's guide to the internet : an African handbook /",
        "link": "http://www.example.com/abc",
        "updated": "2009-12-12T12:00:00Z",
        "sru:recordSchema": "info:srw/schema/1/dc-v1.1",
        "sru:recordXMLEscaping": "xml",
        "sru:recordData": {
            "srw_dc:dc": {
                "dc:title": "The hitchiker's guide to the internet : an African handbook /",
                "dc:creator": [
                    "Goldstuck, A."
                ],
                "dc:type": "text",
                "dc:publisher": "Johannesburg : [S.N.]",
                "dc:date": "1998",
                "dc:language": "eng",
                "dc:identifier": "urn:isbn:1868720721"
            }
        },
        "sru:recordPosition": 2
    }
  ]
 }
}
```


## E.3 JSONP

When the SRU request indicates a preference for SRU response elements within a JSONP response using the mime type ‘text/javascript’ (either through an HTTP ‘Accept’ header or via an ‘**httpAccept**’ parameter) it may also designate a particular JSONP extension format through the parameter ‘**responseType**’.

The default extension format for a JSONP reponse as outlined in the example below is designated by the URI:

- info:srw/1/response-type/jsonp

Note that other response types may be possible and may be designated with their own 'response-type' URIs. If no '**responseType**' parameter is present then the default response type shown here will be assumed to be requested.

| | |
|---|---|
| Mime type ('Accept' header or '**httpAccept**' param) | text/javascript |
| Response type ('**responseType**' param) | info:srw/1/response-type/jsonp |

**Example - JSONP Response (with SRU Response Elements)**

```
callback({
   "feed": {
   ...
   }
})
```

Note that the actual JSON data string is the same as in the previous example.

Web clients such as browsers impose a "same origin" security policy on any executable code which is retrieved over the network. This applies also to data structures such as JSON - or JavaScript Object Notation. One workaround to this rule which allows for the JSON to be fetched and executed in a so-called "cross-site request" is to wrap the JSON in a function call - a fomat which is known as JSONP (JSON with padding). The JSONP mechanism operates by employing a script injection technique which fectches the JSON data structure wrapped within a function call (the JSONP) and immediately executes it as an argument to the function call supplied. A server may thus need to be able to serve up a JSONP response in order to satisfy cross-site requests.

A JSONP reponse is nothing more than the text wrapper elements "callback(" and ")" placed around the actual JSON string, e.g.

```
callback({
   "feed": {
   ...
   }
})
```

Note that the actual function name (here "callback") may be statically or dynamically allocated.

Note also that the mime type changes from 'application/json' to 'text/javascript' as JSONP is technically a JavaScript text rather than a JSON object.

## E.4 RSS Extension

When the SRU request indicates a preference for SRU response elements within an RSS response using the mime type 'appliaction/rss+xml' (either through an HTTP 'Accept' header or via an '**httpAccept**' parameter) it may also designate a particular RSS extension format through the parameter '**responseType**'.

The possible extension formats for an RSS reponse as outlined in the examples below are designated by the URIs:

- info:srw/1/response-type/rss1.0 (RSS 1.0 - default)
- info:srw/1/response-type/rss2.0 (RSS 2.0)

Note that other response types may be possible and may be designated with their own 'response-type' URIs. If no '**responseType**' parameter is present then the default response type (RSS 1.0) shown in the first example will be assumed to be requested.

| | |
|---|---|
| Mime type ('Accept' header or '**httpAccept**' param) | application/rss+xml |
| Response type ('**responseType**' param) | info:srw/1/response-type/rss1.0 (default) |
| | info:srw/1/response-type/rss2.0 |

### E.4.1 RSS 1.0 Extension

This RSS extension type can be used if an RDF-based RSS response is required.

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:sru="info:srw/xmlns/1/sru-2-0-v1" xmlns:atom="http://www.w3.org/2005/Atom"
        xmlns:dc="http://purl.org/dc/elements/1.1/"
        xmlns="http://purl.org/rss/1.0/"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <channel rdf:about=" urn:uuid:a6852153-dc12-4cd9-b3e0-f9ff2ed7f0b3">
    <title>SRU Records</title>
<link>http://z3950.loc.gov:7090/voyager?version=1.1&amp;operation=searchRetrieve&amp;query%3D%
22hitchiker's%20guide%22&amp;startRecord%3D1&amp;maximumRecords%3D5&amp;recordSchema%
3Ddc</link>
    <sru:numberOfRecords>1509</sru:numberOfRecords>
    <items>
      <rdf:Seq>
        <rdf:li rdf:resource="http://www.example.com/xyz"/>
        <rdf:li rdf:resource="http://www.example.com/abc"/>
      </rdf:Seq>
    </items>
  </channel>
  <item rdf:about="http://www.example.com/xyz">
   <title>Don't panic : Douglas Adams and the hitchiker's guide to the galaxy </title>
   <link>http://www.example.com/xyz</link>
   <updated>2009-12-12T12:00:00Z</updated>
   <sru:recordSchema>info:srw/schema/1/dc-v1.1</sru:recordSchema>
   <sru:recordXMLEscaping>xml</sru:recordXMLEscaping>
   <sru:recordData>
    <srw_dc:dc xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:srw_dc="info:srw/schema/1/dc-schema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="info:srw/schema/1/dc-
schema http://www.loc.gov/standards/sru/resources/dc-schema.xsd">
      <dc:title> Don't panic : Douglas Adams and the hitchiker's guide to the galaxy </dc:title>
      <dc:creator>Gaiman, Neil.</dc:creator>
      <dc:type>text</type>
```

```
        <dc:publisher>London : Titan,</dc:publisher>
        <dc:date>2002</dc:date>
        <dc:language>eng</dc:language>
        <dc:subject>Adams, Douglas, 1952-2001.</dc:subject>
        <dc:subject>Adams, Douglas, 1952-2001. Hitch-hiker's guide to the galaxy.</dc:subject>
        <dc:subject>Science fiction, English--History and criticism.</dc:subject>
        <dc:subject>Novelists, English--20th century--Biography.</dc:subject>
        <dc:subject>Prefect, Ford (Fictitious character)</dc:subject>
        <dc:subject>Dent, Arthur (Fictitious character)</dc:subject>
        <dc:identifier>urn:isbn:1840235012</dc:identifier>
      </srw_dc:dc>
    </sru:recordData>
    <sru:recordPosition>1</sru:recordPosition>
  </item>
  <item rdf:about="http://www.example.com/abc">
    <title>The hitchiker's guide to the internet : an African handbook /</title>
    <link>http://www.example.com/abc</link>
    <updated>2009-12-12T12:00:00Z</updated>
    <sru:recordSchema>info:srw/schema/1/dc-v1.1</sru:recordSchema>
    <sru:recordXMLEscaping>xml</sru:recordXMLEscaping>
    <sru:recordData>
      <srw_dc:dc xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:srw_dc="info:srw/schema/1/dc-schema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="info:srw/schema/1/dc-
schema http://www.loc.gov/standards/sru/resources/dc-schema.xsd">
        <dc:title>The hitchiker's guide to the internet : an African handbook /</dc:title>
        <dc:creator>Goldstuck, A.</dc:creator>
        <dc:type>text</dc:type>
        <dc:publisher>Johannesburg : [S.N.],</dc:publisher>
        <dc:date>1998</dc:date>
        <dc:language>eng</dc:language>
        <dc:identifier>urn:isbn:1868720721</dc:identifier>
        </srw_dc:dc>
      </sru:recordData>
    <sru:recordPosition>2</sru:recordPosition>
  </item>
</rdf:RDF>
```

### E.4.2 RSS 2.0 Extension

This RSS extension type can be used if a generic RSS response is required.

```
<?xml version="1.0" encoding="UTF-8"?>
<rss version="2.0"
    xmlns:sru="info:srw/xmlns/1/sru-2-0-v1"
    xmlns:atom="http://www.w3.org/2005/Atom"
```

```
    xmlns:dc="http://purl.org/dc/elements/1.1/"
>
  <channel>
    <title>SRU Records</title>
    <description/>
<link>http://z3950.loc.gov:7090/voyager?version=1.1&amp;operation=searchRetrieve&amp;query%3D%
22hitchiker's%20guide%22&amp;startRecord%3D1&amp;maximumRecords%3D5&amp;recordSchema%
3Ddc</link>
<atom:link rel="self"
href="http://z3950.loc.gov:7090/voyager?version=1.1&amp;operation=searchRetrieve&amp;query%3D%2
2hitchiker's%20guide%22&amp;startRecord%3D1&amp;maximumRecords%3D5&amp;recordSchema%3
Ddc"/>
    <pubDate>Sat, 12 Dec 2009 12:00:00 GMT</pubDate>
    <sru:numberOfRecords>1509</sru:numberOfRecords>
    <item>
      <title>Don't panic : Douglas Adams and the hitchiker's guide to the galaxy </title>
      <description/>
      <link>http://www.example.com/xyz</link>
      <guid>urn:isbn:1840235012</guid>
      <pubDate>Sat, 12 Dec 2009 12:00:00 GMT</pubDate>
      <sru:recordSchema>info:srw/schema/1/dc-v1.1</sru:recordSchema>
      <sru:recordXMLEscaping>xml</sru:recordXMLEscaping>
      <sru:recordData>
        <srw_dc:dc xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:srw_dc="info:srw/schema/1/dc-
schema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="info:srw/schema/1/dc-schema http://www.loc.gov/standards/sru/resources/dc-
schema.xsd">
          <dc:title> Don't panic : Douglas Adams and the hitchiker's guide to the galaxy </dc:title>
          <dc:creator>Gaiman, Neil.</dc:creator>
          <dc:type>text</dc:type>
          <dc:publisher>London : Titan,</dc:publisher>
          <dc:date>2002</dc:date>
          <dc:language>eng</dc:language>
          <dc:subject>Adams, Douglas, 1952-2001.</dc:subject>
          <dc:subject>Adams, Douglas, 1952-2001. Hitch-hiker's guide to the galaxy.</dc:subject>
          <dc:subject>Science fiction, English--History and criticism.</dc:subject>
          <dc:subject>Novelists, English--20th century--Biography.</dc:subject>
          <dc:subject>Prefect, Ford (Fictitious character)</dc:subject>
          <dc:subject>Dent, Arthur (Fictitious character)</dc:subject>
          <dc:identifier>urn:isbn:1840235012</dc:identifier>
        </srw_dc:dc>
      </sru:recordData>
      <sru:recordPosition>1</sru:recordPosition>
    </item>
    <item>
```

```
1516        <title>The hitchiker's guide to the internet : an African handbook /</title>
1517        <description/>
1518        <link>http://www.example.com/abc</link>
1519        <guid>urn:isbn:1868720721</guid>
1520        <pubDate>Sat, 12 Dec 2009 12:00:00 GMT</pubDate>
1521        <sru:recordSchema>info:srw/schema/1/dc-v1.1</sru:recordSchema>
1522        <sru:recordXMLEscaping>xml</sru:recordXMLEscaping>
1523        <sru:recordData>
1524          <srw_dc:dc xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:srw_dc="info:srw/schema/1/dc-
1525    schema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1526    xsi:schemaLocation="info:srw/schema/1/dc-schema http://www.loc.gov/standards/sru/resources/dc-
1527    schema.xsd">
1528            <dc:title>The hitchiker's guide to the internet : an African handbook /</dc:title>
1529            <dc:creator>Goldstuck, A.</dc:creator>
1530            <dc:type>text</dc:type>
1531            <dc:publisher>Johannesburg : [S.N.],</dc:publisher>
1532            <dc:date>1998</dc:date>
1533            <dc:language>eng</dc:language>
1534            <dc:identifier>urn:isbn:1868720721</dc:identifier>
1535          </srw_dc:dc>
1536        </sru:recordData>
1537        <sru:recordPosition>2</sru:recordPosition>
1538      </item>
1539    </channel>
1540  </rss>
1541
```

# Appendix F. Interoperation with Earlier Versions (non-normative)

## F.1 Operation and Version

Earlier versions of the protocol (versions 1.1 and 1.2) included request parameters ‘operation’ and ‘version’, and response element <version>. These are removed from version 2.0. This section is included to describe (1) differences imposed by their removal; and (2) how version 2.0 servers may interoperate with clients running earlier versions that include them.

### F.1.1 Differences Imposed by their Removal

#### F.1.1.1 Operation – Request Parameter

Earlier versions as well as this version of SRU express the concept of an operation: a searchRetrieve operation, a scan operation, and an Explain operation are defined. A searchRetrieve or scan request carries a mandatory operation parameter whose value is 'searchRetrieve' or 'scan' respectively, allowing these operations to be distinguished, so that they can both be supported at a single network endpoint.

This specification defines the searchRetrieve operation and there is also a scan operation (a separate specification [5]). However there is no operation parameter for either. As for earlier versions, searchRetrieve and scan may be supported at a single network endpoint because it is heuristically possible to distinguish these operations: A request is a scan request if and only if it includes a scanClause parameter; it is a searchRetrieve request if and only it contains EITHER a query parameter OR a queryType parameter.

However, if a new operation were to be defined, or a new version of searchRetrieve or scan, then it may no longer be possible to heuristically distinguish these operations and then it may be necessary to define the operation parameter for one or more operations.

#### F.1.1.2 Version – Request Parameter and Response Element

In earlier versions a version request parameter and response element were defined because it was assumed that multiple versions might be supported at a single endpoint. With version 2.0, the version request parameter and response element are removed and it is EXPLICITY ASSUMED that there will be different endpoints for different versions.

### F.1.2 Interoperation

Following are guidelines for interoperation with implementations of earlier versions where the operation or version parameter or element is used.

- A client operating under version 2.0 of SRU SHOULD include NEITHER of the request parameters ‘operation’ or ‘version’.
- **If a server is operating under version 2.0 and the version parameter is included in a received request, and the server supports the requested version**: It may, if it chooses, process the request under that version. However, details of how these parameters are treated are beyond the scope of this standard.
- **If the version parameter is included in a received request, and the server does not supports the requested version**: the request should be rejected and a fatal diagnostic included. (The server may be able to supply the response according to the requested version – even though it does not in general support that version. If the server is willing and able to issue the failure response according to the requested version, then it should do so. However it is not required to do so, and, unfortunately, failure to do will probably mean that the response cannot be interpreted by the client.)

- **If the operation parameter is included in a received request:**
  - If the value of the operation parameter is searchRetrieve, the server may ignore it.
  - If the value of the operation parameter is other than searchRetrieve, the server may reject the request (with a fatal diagnostic) or may, if it chooses, process the request; however details of how the request should be processed are out of scope.

## F.2 Replacement of ResultSetIdleTime with ResultSetTTL

Earlier version of SRU include the response element <resultSetIdleTime>. In version 2.0 this element is removed and the response element <resultSetTTL> is defined.

Implementors of SRU 2.0 servers who had previous familiarity with earlier SRU versions are cautioned that <resultSetIdleTime> is no longer a valid response element. SRU 2.0 clients should be prepared to accept <resultSetTTL> in the response.

The earlier versions included the request parameter resultSetTTL and the response element <resultSetIdleTime>, with different semantics. During development of version 2.0 it was concluded that this mismatch in the request parameter and (apparently) corresponding response element was a flaw, it had caused considerable confusion, and it should not be perpetuated in version 2.0. It was believed to be serious enough a flaw that it should be corrected even though this would cause some incompatibility between versions.

Thus version 2.0 defines the request parameter resultSetTTL and corresponding response element <resultSetTTL>

## F.3 recordPacking and recordXMLEscaping

In version 2.0, the recordPacking request parameter, as well as the <recordPacking> subelement of <record> in the response, that had been in earlier versions, are renamed recordXMLEscaping. In addition, a new recordPacking request parameter is introduced with an entirely different meaning.

Servers should be prepared to accept the new recordXMLEscaping parameter, and also to recognize the new meaning of the recordPacking parameter. Clients should be prepared to accept the new <recordXMLEscaping> element in the response.