SRU

Search and Retrieve via URL


This document is based on the SRU version 1.2 specifications at http://www.loc.gov:8081/standards/sru, which are proposed as input to the TC. This is a first attempt to consolidate those specifications, which are currently spread across many web pages, into a single document which would contain the essential specifications and exclude non-normative content. The purpose of the document is to begin to determine what content should or should not be included in a committee draft and in what order. Please review this document more with that in mind than technical content.

Contents

1. The searchRetrieve operation .......................................................... 4

1.1 Parameters .................................................................................. 4

1.1.1 Request Parameters .................................................................. 4

1.1.2 Response Parameters ............................................................... 5

1.2 Records ...................................................................................... 6

1.2.1 Record Parameters .................................................................. 6

1.2.2 Record Packing ................................................................. 8

1.3 Result Sets ................................................................................. 8

1.3.1 Result Set Model ............................................................... 8

1.3.2 resultSetId ................................................................. 9
3.3.1.5 Boolean Modifiers ................................................................. 35
3.3.1.6 Proximity Units ................................................................. 36

4. Diagnostics ................................................................................. 38
  4.1 Diagnostic Schema .................................................................. 38
  4.2 Diagnostics List ...................................................................... 40
  5.1 ZeeRex for SRU........................................................................ 54

6. Scan Operation ............................................................................ 56
  6.1 Parameters ............................................................................ 56
  6.2 Terms .................................................................................... 58
  6.3 Example Scan Response .......................................................... 59
SRU - Search and Retrieve via URL

SRU is a standard XML-focused search protocol for Internet search queries, utilizing CQL (Contextual Query Language), a standard syntax for representing queries.

1. The searchRetrieve operation

The searchRetrieve operation is the main operation in SRU. It allows the client to submit a search and retrieve for matching records from the server.

1.1 Parameters

1.1.1 Request Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Occurence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>operation</td>
<td>mandatory</td>
<td>The string: 'searchRetrieve'.</td>
</tr>
<tr>
<td>version</td>
<td>mandatory</td>
<td>The version of the request, and a statement by the client that it wants the response to be less than, or preferably equal to, that version. See Version.</td>
</tr>
<tr>
<td>query</td>
<td>mandatory</td>
<td>Contains a query expressed in CQL to be processed by the server. See CQL.</td>
</tr>
<tr>
<td>startRecord</td>
<td>optional</td>
<td>The position within the sequence of matched records of the first record to be returned. The first position in the sequence is 1. The value supplied MUST be greater than 0. The default value if not supplied is 1.</td>
</tr>
<tr>
<td>maximumRecords</td>
<td>optional</td>
<td>The number of records requested to be returned. The value must be 0 or greater. Default value if not supplied is determined by the server. The server MAY return less than this number of records, for example if there are fewer matching records than requested, but MUST NOT return more than this number of records.</td>
</tr>
<tr>
<td>recordPacking</td>
<td>optional</td>
<td>A string to determine how the record should be escaped in the response. Defined values are 'string' and 'xml'. The default is 'xml'. See Records.</td>
</tr>
<tr>
<td>recordSchema</td>
<td>optional</td>
<td>The schema in which the records MUST be returned. The</td>
</tr>
</tbody>
</table>
value is the URI identifier for the schema or the short name for it published by the server. The default value if not supplied is determined by the server. See Record Schemas.

resultSetTTL optional

The number of seconds for which the client requests that the result set created should be maintained. The server MAY choose not to fulfil this request, and may respond with a different number of seconds. If resultSetTTL is not supplied then the server will determine the value. See Result Sets.

stylesheet optional

A URL for a stylesheet. The client requests that the server simply return this URL in the response. See Stylesheets.

extraRequestData optional

Provides additional information for the server to process. See Extensions.

Example:


This example is a search for the term "dinosaur", requesting that at most one record be returned, according to the 'dc' schema

1.1.2 Response Parameters

The response to an SRU searchRetrieve request is an XML document. The table below provides a summary and description of the elements provided by the XML document. The "Type" column indicates either an XML Schema type ("xsd:"), or a type defined within the schema.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Occurence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>xsd:string</td>
<td>Mandatory</td>
<td>The version of the response. This MUST be less than or equal to the version requested by the client. See Versions.</td>
</tr>
<tr>
<td>numberOfRecords</td>
<td>xsd:integer</td>
<td>Mandatory</td>
<td>The number of records matched by the query. If the query fails this MUST be 0.</td>
</tr>
<tr>
<td>resultSetId</td>
<td>xsd:string</td>
<td>Optional</td>
<td>The identifier for a result set that was</td>
</tr>
</tbody>
</table>
created through the execution of the query. See Result Sets.

The number of seconds after which the created result set will be deleted. The result set may also become unavailable before this. See Result Sets.

A sequence of records matched by the query, or surrogate diagnostics. See Records.

The next position within the result set following the final returned record. If there are no remaining records, this field MUST be omitted.

A sequence of non surrogate diagnostics generated during execution. See Diagnostics.

Additional information returned by the server. See Extensions.

The request parameters echoed back to the client in a simple XML form. See Echoing the Request.

1.2 Records

All records in SRU are transferred in XML. Records are not assumed to be stored in XML. Records which are not natively XML must be first transformed into XML before being transferred. Records in the response 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 validatable against the record schema.

The records parameter in the response is a sequence of record elements, each of which contains either a record or a surrogate diagnostic explaining why that particular record could not be transferred. If the record schema is unknown or the record cannot be rendered in that schema, then the server MUST return a diagnostic.

Each record element is structured into the following elements:

1.2.1 Record Parameters
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Occurrence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>recordSchema</td>
<td>xsd:string</td>
<td>mandatory</td>
<td>The URI identifier of the XML schema in which the record is encoded. Although the request may use the server's assigned short name, the response must always be the full URI. See Record Schemas</td>
</tr>
<tr>
<td>recordPacking</td>
<td>xsd:string</td>
<td>mandatory</td>
<td>The packing used in recordData, as requested by the client or the default. See below.</td>
</tr>
<tr>
<td>recordData</td>
<td>&lt;stringOrXmlFragment&gt;</td>
<td>mandatory</td>
<td>The record itself, either as a string or embedded XML</td>
</tr>
<tr>
<td>recordIdentifier</td>
<td>xsd:string</td>
<td>optional</td>
<td>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.</td>
</tr>
<tr>
<td>recordPosition</td>
<td>xsd:positiveInteger</td>
<td>optional</td>
<td>The position of the record within the result set. See Result Sets</td>
</tr>
<tr>
<td>extraRecordData</td>
<td>&lt;xmlFragment&gt;</td>
<td>optional</td>
<td>Any additional information to be transferred with the record. See Extensions</td>
</tr>
</tbody>
</table>

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

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

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 which forms the response. However, some clients may prefer that the records be transferred as XML in order to manipulate them directly with a stylesheet which renders the records and potentially also the user interface.

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

1.3 Result Sets

SRU does not assume support of persistent result sets -- that a result set created by one request may necessarily be accessed by a client in a subsequent request. SRU does expect the server to state whether or not it supports persistent result sets, and if so the result set model described below is required.

There are applications in which result sets are critical; on the other hand there are applications in which result sets are not viable. An example of the first might be scientific investigation of a database with comparison of data sets produced at different times. An example of the latter might be a very frequently used database of web pages in which persistent result sets would be an impossible burden on the infrastructure due to the frequency of use.

Even if the server does not make result sets available for public manipulation, the following model is also important to understand in order to allow a single request to both match records and then sort them.

1.3.1 Result Set Model

Processing of a query results in the selection of a set of records, represented by a result set maintained at the server; logically it is an ordered list of references to the records. Once created, a result set cannot be modified. Any operation which would somehow change a result set instead creates a new result set. Each result set is referenced via a unique identifying string, generated by the server when the result set is created.

From the client's point of view, the result set is a set of records each referenced by an ordinal number, beginning at 1. The client may request a given record from a result set according to a specific schema. For example the client may request record 1 in Dublin Core, and subsequently request record 1 in MODS. The requested schema is not a property of the result set (nor of the requested records as a member of the result set); the result set is simply the ordered list of records.
A record might be deleted or otherwise become unavailable while a result sets which references that record still exists. If a client then requests that record, the server is expected to supply a surrogate diagnostic in place of the record. For example, if the record at position 2 in a result set is deleted and then a client requests records 1 through 3, the server should supply, in order: record 1, a surrogate diagnostic for record 2, record 3.

The records in a result set are not necessarily ordered according to any specific or predictable scheme, unless it has been created with a request that contains a sort specification as part of the query. See sorting in CQL for more information regarding the specifics of sorting. If search and sort specifications are supplied on the same request then only the final sorted result set is considered to exist, even if the server internally creates a result set and then sorts it.

1.3.2 resultSetId

If the server supports result sets, it may include a resultSetId in the searchRetrieve response, along with an idle time described below. If another query is submitted then the server will again supply a result set id. If the result of the query would modify an existing result set (for example, a request to sort an existing result set), then the server must supply a new id for this new set. The server should maintain unique names for each result set created, even if the result sets no longer exist, such that clients do not mistakenly request records from the new set when meaning to refer to the previous set with the same identifier.

1.3.3 resultSetIdleTime

The server may supply an idle time along with a result set. The server is making a good-faith estimate that the result set will remain available and unchanged (both in content and order) until a timeout (a period of inactivity exceeding the idle time). The idle time is an integer representing seconds; it must be a positive integer, and should not be so small that a client cannot realistically reference the result set again. If the server does not intend that the result set be referenced, it should omit the result set identifier in the response.

1.4 Version

In any actively developed protocol or piece of software, there is a concern about interoperability between different versions. In SRU, there is an explicit interoperability mechanism, with precisely defined semantics. The mechanism defined allows for clients and servers using different versions to interact without protocol level errors. Versions of SRU will always be recorded as strings of the format 'major.minor' where major and minor are independent integers. Operations All SRU operations have a version parameter, with the exception of the parameterless form of the explain request. For example:

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

The version parameter on a request both indicates the version of the request and is a statement by the client that it wants the response to be less than, or preferably equal to, that version. The version parameter in the
response message is the version of the response. If the server cannot supply a response in that version or lower, then it must return a diagnostic. If possible this diagnostic would be in the version requested or lower, but that is not a requirement. Here are some examples of how this works in practice. If a 2.0 client asks a 1.1 server for a 2.0 response, then the server is able to respond with a 1.1 response as it is lower than version 2.0. If a 1.1 client asks a 2.0 server for a 1.1 response then the server is able to reduce its response version to accomodate the client. If a 1.1 client asks a 1.1 server for a 1.1 response, then there is no version mismatch and the server is able to accomodate the request. Version 1.0 Version 1.0 was an experiment, and has been officially deprecated. The version 1.0 implementation trial was very useful informing the development of version 1.1, the first non-experimental version. Version 1.0 does not have a version parameter in any of the requests or responses and hence cannot be considered to be part of this version interoperability system. If a client requests version 1.0, then the server may return a 1.0 response but is under no obligation to do so. Version Documentation and Changes Documentation for all versions of SRU will be maintained such that server and client authors are able to track changes between versions. A summary document of changes between versions will also be maintained. This version interoperability solution will only work so long as there are no additional mandatory parameters added to a request, and as such the SRU editorial board will endeavor to only ever add optional elements. If there is a requirement to add a mandatory parameter to a request, then this will be announced with as much prior warning as possible. While only the documentation for the most recent version of the protocol is maintained, the changes between versions are listed below.

1.5 Stylesheets

In order to render the response, "thin" clients may provide a stylesheet 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. The echoedRequest parameter on the response enables a client to use this stylesheet to also have the request it just made available without any client side logic. Operations All operations, other than the parameterless explain request, have the stylesheet parameter. The value of the parameter is the URL of the stylesheet to be included in the response. This URL is to be included 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. If the server cannot fulfill this request it must supply a diagnostic. This parameter may not be used with SRU via SOAP. It is a SOAP error to return a stylesheet, and hence an error to request one. If this parameter is not supplied, then the server can, at its discretion, include a default stylesheet. The default stylesheet URL may be included in the explain document. For example, upon receiving the request ...

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

...the server must include the following as beginning of the response:

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

Very thin clients, such as a web browser with a stylesheet as above, may not have the facility to record the query that generated the response it has just received. In order to prevent clients having to maintain this information, the server may echo the request back to the client along with the response. OperationsThere are no request elements associated with this functionality. There is one response element per operation in which the request is echoed. The name of this is the name of the response element, prefixed by echoed. The parameters are rendered into XML, as per the SRU over SOAP request.xQuery xQuery is an additional parameter for searchRetrieve and scan, which has the query rendered in XCQL. This has two benefits:

1. The client can use XSLT or other XML manipulation to modify the query without having a CQL query parser.
2. The server can return extra information specific to the clauses within the query. See the next section on extensions for more information.

1.7 baseUrl

A server can include is own base URL in the echoed request. This allows the client to easily 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. For example:

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

1.8 Extensions

Messages in all of the operations, both in the request and in the response, have a field in which additional information may be provided. This is a built in extension mechanism where profiles may specify a schema for what to include in this section without requiring the developers to change the basic messages and thus render their implementation uninteroperable with other servers and clients. It is expected that if there is sufficient demand for a particular piece of additional information, that piece of information will be migrated...
into the protocol in a later version. In this way, only implemented and useful features will be added in future versions, rather than features which just seem like a good idea. OperationsRequest For SRU via GET or POST, the name for an extension parameter must begin with 'x-': lower case x followed by hyphen. The SRU protocol will never include an official parameter with a name beginning with 'x-', and hence this will never clash with a mainstream parameter name. It is recommended that the parameter name be 'x-' followed by an identifier for the namespace for the extension, again followed by a hyphen, followed by the name of the element within the namespace. For example

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

Note that this convention does not guarantee uniqueness since the parameter 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 parameter "x-infoNNN-XXX" where NNN is the 'info:srw' authority string, and XXX is the name of the parameter. Extension names MUST never be assigned with this form except by the proper authority for the given 'info' namespace. Response Every response has an extraResponseData section. This section can include any well-formed XML, and hence servers can include namespaced XML fragments within it in order to convey information back to the client. The extension MUST supply a namespace and the element names with which to do this, if feedback to the client is necessary. For example:

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

1.8.1 SRU Extension Parameters via SOAP

For SRU via SOAP, the extension parameters are XML structures. The request parameters are identified by their full namespace, and the name of the parameter is the name of the XML element. Even if there is only one piece of additional information supplied, it must be within a namespaced XML element. This is in order to ensure that servers can distinguish a parameter from one extension from another. For example:

```xml
<extraRequestData>
  <theo:onSearchFail xmlns:theo="info:srw/extension/4/searchextensions">scan</theo:onSearchFail>
</extraRequestData>
```

**Semantics:** If the server does not understand a piece of information in an extension parameter, it may silently ignore it. This is unlike many other request parameters, where if the server does not implement that particular feature it MUST respond with a diagnostic. If the particular request requires some confirmation
that it has been carried out rather than ignored, then the profile designer should include a field in the response. The semantics of parameters in the request may not be modified by extensions. For example, a \textit{x-qt-queryType} parameter could not change query to be an SQL query, as a server that does not understand the extension would expect the query to be in CQL and thus be unable to parse it. Instead, the extension should create a new parameter for the SQL query. The semantics of parts of the response may be modified by extensions. It would be possible (though very strange), for example, to replace the recordSchema field in the record structure with your favorite chocolate cookie recipe when the client sends a \textit{x-ct-cookieType} request parameter. The response semantics may be changed in this way only if the client specifically requests the change. Clients should also expect to receive the regular semantics, as servers are at liberty to ignore extensions, and hence it is recommended that this not be done. ExtraResponseData may be sent that is not directly associated with the request. For example it may contain cost information regarding the query or information on the server or database supplying the results. This data must, however, have been requested. As the request may be echoed, the server must be able to transform the parameters into their XML form. If it encounters an unrecognised parameter, the server may either make its best guess as to how to transform the parameter, or simply not return it at all. It should not, however, add an undefined namespace to the element as this would invalidate the response. If the contents of the parameter is an XML structure, then the extension designer should also specify how to encode this structure in a URL. This may simply be to escape all of the special characters, but the designer could also create a string encoding form with rules as to how to generate the XML in much the same fashion as the relationship between CQL and XCQL. Record, Term and Query ExtensionsIn addition to extra information being included at the top level of the response, there are three further elements in which information can be returned. These are:

1. **extraRecordData**: Extra information relating to the record in the explain or searchRetrieve responses.

2. **extraTermData**: Extra information relating to the term in the scan response.

3. **extraOperandData**: Extra information relation to the query clause echoed back to the client in XCQL.

Examples:

```xml
<record>
  <recordSchema>info:srw/schema/1/dc-v1.1</recordSchema>
  <recordPacking>xml</recordPacking>
  <recordData>...</recordData>
  <recordPosition>1</recordPosition>
  <extraRecordData>
    0.965</rel:score>
  </extraRecordData>
</record>

<rel:score xmlns:rel="info:srw/extension/2/relevancy-1.0">
  <term>
1.8.2 Authentication Tokens

Authentication tokens are an illustration of the usage of extensions in SRU.

SRU does not define any method to distinguish among concurrent users, but some business models may impose requirements, for example, to ensure that one user does not modify another's resultsets, to allow a server to restrict a user to a pre-determined number of searches before charges are imposed, or to limit the number of concurrent searches for a user or number within a certain time frame. Or, on the other hand, if it can be demonstrated that a search has led directly to a sale, then the user may receive a commission. Another example is to enable the service to track how different users use the system, possibly to enforce acceptable usage policies.

This section aims to discuss the various methods in which different users may be authenticated in an interoperable manner. In a stateless environment, or one where the ability to track individual users is not important, this can be ignored without peril.

There are several technical methods by which distinct users may be identified, from IP address to additional header information to SSL. The different methods create additional requirements and function at various levels of success.

IP Address

Users may be differentiated by the IP address from which they are connecting to the SRU server. Unfortunately this is unreliable at best due to the increasing use of web proxy systems -- there may be many
users all of which appear to be coming from the same IP address due to a proxy. The advantage is that it is completely transparent to the client and hence the user, so for a small service may be appropriate.

**Basic Authentication**

Basic Authentication is the fairly simple method used in many web servers to authenticate users against a list or database -- the client is required to send a username and password. This is a very easy-to-configure method to authenticate users, however it does not allow for users that are not authenticated -- every request must have a valid user and password sent or it will be rejected. This model is appropriate for a paid-for service or one which is used only by a set of known individuals, but is less appropriate for a service which may be used by anyone.

**Secure Sockets**

SSL is an encrypted version of HTTP (https) and hence is more secure than basic authentication alone as the traffic cannot be easily intercepted. For financial transations using SRU then this is certainly appropriate as the user is already known in advance and every care for the data must be taken. However for every day services that may be used by anyone, it is a very complex solution.

**Additional Message Data**

The preferred method for identifying users while still allowing non-authenticated access is by the inclusion of an additional field in the extraRequestData and extraResponseData fields. This method allows the server to chose when authentication is required (for example only if a resultset is needed) and when it can continue to act in a stateless fashion. This may be appropriate for any sort of transaction with the exception of cases when the data should be conveyed in an encrypted fashion, in which case SSL should be used as well.

The recommended name for this field is **authenticationToken**, and hence **x-authenticationToken** when it is passed on the URL for SRU. If the server sends back one of these tokens with a response, then the client should return it in the same fashion in any subsequent request to allow the server to know that the requests should be considered to be from the same user.

Further business logic may be required to manipulate these tokens. For example a separate SOAP service may be required to distribute the tokens on request, to delete tokens when they've finished being used or to enable the sharing of such tokens between users to allow shared access to result sets. No best practice guidelines are currently available for such models, but any interested parties are asked to bring up their requirements on the SRU mailing list so that some may be drafted.

The URI for the namespace for this extension is **info:srw/extension/2/auth-1.0**
2. Protocol Transport

2.1 SRU Via HTTP GET

The client may send an SRU request via the HTTP GET method. A URL is constructed and sent to the server with fixed parameter names with fixed meanings. When unicode characters need to be encoded, there are some additional constraints, discussed below.

The response must be XML conforming to the response schema of the operation. SRU via HTTP GET can thus be described as the simplest case of XML over HTTP.

An example of what might pass over the wire:

GET /voyager?version=1.2&operation=searchRetrieve&query=dinosaur HTTP/1.1
Host: z3950.loc.gov:7090

2.1.1 Syntax

An SRU request (when transported via HTTP GET) is a URI as described in RFC 3986 (See Note 1). Specifically it is an HTTP URL (as described in section 3.3 of RFC 1738; however there are some further notes about character encoding below, and uses the standard & separated key=value encoding for parameters in the query part of the URI.

The parameters for the query section of the URL (the information following the question mark) of the various operations are described in their own sections.

2.1.2 Encoding Issues

The following encoding procedure is recommended, in particular, to accomodate Unicode characters (characters from the Universal Character Set, ISO 10646) beyond U+007F, which are not valid in a URI. This is normally relevant only to the query parameter of the searchRetrieve operation and the scanClause parameter of the scan operation.

1. Convert the value to UTF-8.

2. Percent-encode characters as necessary within the value. See http://rfc.net/rfc3986.html#s2.1.

3. Construct the 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. If in doubt, percent-encode.

**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 delimeter, 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

**2.1.3 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.
Notes:

1. RFC 1738 is obsoleted by RFC 3986. However, RFC 1738 describes the 'http:' URI scheme; RFC 3986 does not, instead indicating that a separate document will be written to do so, but it has not yet been written. So currently there is no valid, normative reference for the 'http:' URI scheme, and so the obsolete RFC 1738 is referenced. When there is a valid, normative reference, it will be listed here.

2.2 SRU Via HTTP POST

Instead of constructing a URL, the parameters may be sent via POST to the server. The Content-type header MUST be set to 'application/x-www-form-urlencoded'. Compare to 'text/xml' for SRU via SOAP below, which can be used to distinguish the two transports at the same end point.

POST has several benefits over GET for transferring the request to the server. 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 clients. This length restriction can be avoided by using POST.

The response for SRU via POST is identical to that of SRU via GET, an xml document.

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: 51
version=1.1&operation=searchRetrieve&query=dinosaur

2.3 SRU Via SOAP

SRU via SOAP is a binding to the SOAP recommendation of the W3C. 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.

The incremental benefits of SRU via SOAP are the ease of structured extensions, web service facilities such as proxying and request routing, and the potential for better authentication systems.

2.3.1 SOAP Requirements

Clients and servers MUST support SOAP version 1.1, and MAY support version 1.2 or higher. This
requirement is allow as much flexibility in implementation as possible.

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 and SRU via POST thus have three content-type headers to consider.

The specification tries to adhere to the Web Services Interoperability recommendtions.

2.3.2 SOAP Parameter Differences

There are some differences regarding the parameters that can be transported via the SOAP binding.

The 'operation' request parameter MUST NOT be sent. The operation is determined by the XML constructions employed.

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

Example SOAP request:

```xml
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP:Body>
    <SRW:searchRetrieveRequest xmlns:SRW="http://www.loc.gov/zing/srw/">
      <SRW:version>1.1</SRW:version>
      <SRW:query>dinosaur</SRW:query>
      <SRW:startRecord>1</SRW:startRecord>
      <SRW:maximumRecords>1</SRW:maximumRecords>
    </SRW:searchRetrieveRequest>
  </SOAP:Body>
</SOAP:Envelope>
```
3. CQL: Contextual Query Language

CQL, the Contextual Query Language, is a formal language for representing queries to information retrieval systems such as web indexes, bibliographic catalogs and museum collection information. The design objective is that queries be human readable and writable, and that the language be intuitive while maintaining the expressiveness of more complex languages.

Traditionally, query languages have fallen into two camps: Powerful, expressive languages, not easily readable nor writable by non-experts (e.g. SQL, PQF, and XQuery); or simple and intuitive languages not powerful enough to express complex concepts (e.g. CCL and google). CQL tries to combine simplicity and intuitiveness of expression for simple, every day queries, with the richness of more expressive languages to accomodate complex concepts when necessary.

3.1 Query Syntax

1. CQL Query
   A CQL query consists of either a single search clause [example 1], or multiple search clauses connected by boolean operators [example 2]. It may have a sort specification at the end, following the 'sortBy' keyword [example 3]. In addition it may include prefix assignments which assign short names to context set identifiers [example 4].

   Examples:
   
   a. dc.title any fish
   
   b. dc.title any fish or dc.creator any sanderson
   
   c. dc.title any fish sortBy dc.date/sort.ascending
   
   d. > dc = "info:srw/context-sets/1/dc-v1.1" dc.title any fish

2. Search Clause
   A search clause consists of either an index, relation and a search term [example 1], or a search term by itself [example 2]. If the clause consists of just a term, then the index is treated as 'cql.serverChoice', and the relation is treated as '=' [example 3].

   Examples:
   
   a. dc.title any fish
   
   b. fish
c. cql.serverChoice = fish

3. Search Term
Search terms MAY be enclosed in double quotes [example 1], though need not be [example 2]. Search terms MUST be enclosed in double quotes if they contain any of the following characters: < > = / ( ) and whitespace [example 3]. The search term may be an empty string [example 4], but must be present in a search clause. The empty search term has no defined semantics.

Examples:
   a."fish"
   b.fish
   c."squirrels fish"
   d.""

4. Index Name
An index name always includes a base name [example 1] and may also include a prefix [example 2], which determines the context set of which the index is a part. The base name and the prefix are separated by a dot character ('). If multiple ' ' characters are present, then the first should be treated as the prefix/base name delimiter. If the prefix is not supplied, it is determined by the server. Examples:
   a.title any fish
   b.dc.title any fish

5. Relation
The relation in a search clause specifies the relationship between the index and search term. It also always includes a base name [example 1] and may also include a prefix providing a context for the relation [example 2]. If a relation does not have a prefix, the context set is 'cql'. If no relation is supplied in a search clause, then = is assumed, which means that the relation is determined by the server.

Examples:
   a.dc.title any fish
   b.dc.title cql.any fish

6. Relation Modifiers
Relations may be modified by one or more relation modifiers. Relation modifiers always include a base
name, and may include a prefix for a context set as above [example 1]. If a prefix is not supplied, the context set is 'cql'. Relation modifiers are separated from each other and from the relation by forward slash characters('/'). Whitespace may be present on either side of a '/' character, but the relation plus modifiers group may not end in a '/' [example 2]. Relation modifiers may also have a comparison symbol and a value. The comparison symbol is any of '=' '<=' '>=' '<>'. The value must obey the same rules for quoting as search terms, above [example 3].

Examples:

    a. dc.title any/relevant fish
    b. dc.title any/ relevant /cql.string fish
    c. dc.title any/rel.algorithm=cori fish

7. Boolean Operators
Search clauses may be linked by boolean operators. These are: **and**, **or**, **not** and **prox** [example 1]. Note that **not** is 'and-not' and must not be used as a unary operator. Boolean operators all have the same precedence; they are evaluated left-to-right. Parentheses may be used to override left-to-right evaluation [example 2].

Examples:

    a. dc.title any fish or dc.creator any sanderson
    b. dc.title any fish or (dc.creator any sanderson and dc.identifier = "id:1234567")

8. Boolean Modifiers
Booleans may be modified by one or more boolean modifiers, separated as per relation modifiers with '/' characters. Again, boolean modifiers consist of a base name and may include a prefix determining the modifier's context set [example 1]. If not supplied, then the context set is 'cql'. As per relation modifiers, they may also have a comparison symbol and a value [example 2].

Examples:

    a. dc.title any fish or/rel.combine=sum dc.creator any sanderson
    b. dc.title any fish prox/unit=word/distance>3 dc.title any squirrel

Proximity Modifiers
Basic proximity modifiers are defined in the **CQL context set**. Proximity units 'word', 'sentence', 'paragraph', and 'element' are defined in the CQL context set, and may also be defined in other context sets.
Within the CQL set they are explicitly undefined. When defined in another context set they may be assigned specific meaning.

Thus compare "prox/unit=word" with "prox/xyz.unit=word". In the first, 'unit' is a prox modifier from the CQL set, and as such its values are undefined, so 'word' is subject to interpretation by the server. In the second, 'unit' is a prox modifier defined by the xyz context set, which may assign the unit 'word' a specific meaning.

The context set xyz may define additional units, for example, 'street':

prox/xyz.unit="street"

Note that this approach, 'prox/xyz.unit="street"', is preferable to 'Prox/unit=xyz.street'. In the first case, 'unit' is a modifier define in the xyz context set, and 'street' is a value defined for that modifier. In the second, 'unit' is a modifier from the cql context set, with a value defined in a different set. so its value would have to be one that is defined in the cql context set. Pairing a modifier from one set with a value from another is not a good practice.

9. Sorting

Queries may include explicit information on how to sort the result set generated by the search. The sort specification is included at the end, and is separated by a 'sortBy' keyword. The specification consists of an ordered list of indexes, potentially with modifiers, to use as keys on which to sort the result set. If multiple keys are given, then the second and subsequent keys should be used to determine the order of items that would otherwise sort together. Each index used as a sort key has the same semantics as when it is used to search.

Modifiers may be attached to the index in the same way as to booleans and relations in the main part of the query. These modifiers may be part of any context set, but the CQL context set and the Sort context set are especially important. If a modifier may be used in this way should be stated in the description of its semantics, and it is the only time at which modifiers may be attached to indexes. As many types of search also require specification of term order (for example the <, > and within relations), these modifiers are often specified as relation modifiers.

Examples:

a."cat" sortBy dc.title

b."dinosaur" sortBy dc.date/sort.descending dc.title/sort.ascending

10. Prefix Assignment

Warning: The use of Prefix Maps is very uncommon.

A Prefix Map may be used to assign context set names to specific identifiers in order to be sure that the server maps them in a desired fashion. It may occur at any place in the query and applies to anything below
the map in the query tree. A prefix assignment is specified by: '>' shortname '=' identifier [example 1]. The shortname and '=' sign may be omitted, in which case it sets a default context set for indexes [example 2].

**Examples:**

```
a.> dc = "http://deepcustard.org/" dc.custardDepth > 10
b.> "http://deepcustard.org/" custardDepth > 10
```

11. **Case Insensitive**

All parts of CQL are case insensitive apart from user supplied search terms, values for modifiers and prefix map identifiers, which may or may not be case sensitive. If any case insensitive part of CQL is specified with both upper and lower case, it is for aesthetic purposes only.

**Examples:**

```
a.dC.tiTlE any fish
b.dc.TitlE Any/rEl.algOrIThm=cori fish soRtbY Dc.TitlE
```

### 3.2 BNF

Following is the Backus Naur Form (BNF) definition for CQL. ["::=" represents "is defined as"]

```plaintext
sortedQuery ::= prefixAssignment sortedQuery 
  | scopedClause ['sortby' sortSpec]

sortSpec ::= sortSpec singleSpec | singleSpec

singleSpec ::= index [modifierList]

cqlQuery ::= prefixAssignment cqlQuery 
  | scopedClause

prefixAssignment ::= '>' prefix '=' uri 
  | '>' uri

scopedClause ::= scopedClause booleanGroup searchClause 
  | searchClause
```
booleanGroup ::= boolean [modifierList]

boolean ::= 'and' | 'or' | 'not' | 'prox'

searchClause ::= (' cqlQuery ')
          | index relation searchTerm
          | searchTerm

relation ::= comparitor [modifierList]

comparitor ::= comparitorSymbol | namedComparitor

comparitorSymbol ::= '=' | '>' | '<' | '>=' | '<=' | '<>' | '=='

namedComparitor ::= '!=' | '>' | '<' | '>= | '<=' | '<>' | '=='

modifierList ::= modifierList modifier | modifier

modifier ::= '/\' modifierName [comparitorSymbol modifierValue]

prefix, uri, modifierName, modifierValue, searchTerm, index

expression ::= identifier | 'and' | 'or' | 'not' | 'prox' | 'sortby'

identifier ::= charString1 | charString2

charString1 ::= Any sequence of characters that does not include any of the following:

whitespace
( (open parenthesis )
) (close parenthesis)
= < >
"" (double quote)
/

If the final sequence is a reserved word, that token is returned instead. Note that '.', (period) may be included, and a sequence of digits is also permitted. Reserved words are 'and', 'or', 'not', and 'prox' (case insensitive). When a reserved word is used in a search term, case is preserved.
charString2 := Double quotes enclosing a sequence of any characters except double quote (unless preceded by backslash (\)). Backslash escapes the character following it. The resultant value includes all backslash characters except those releasing a double quote (this allows other systems to interpret the backslash character). The surrounding double quotes are not included.

3.3 Context Sets

CQL is so-named ("Contextual Query Language") because it is founded on the concept of searching by semantics or context, rather than by syntax. The same search may be performed in a different way on very different underlying data structures in different servers, but the important thing is that both servers understand the intent behind the query. In order for multiple communities to define their own semantics, CQL uses Context Sets in order to ensure cross-domain interoperability.

Context sets permit CQL users to create their own indexes, relations, relation modifiers and boolean modifiers without fear of choosing the same name as someone else and thereby having an ambiguous query. All of these four aspects of CQL must come from a context set, however there are rules for determining the prevailing default if one is not supplied. Context sets allow CQL to be used by communities in ways which the designers could not have foreseen, while still maintaining the same rules for parsing which allow interoperability.

When defining a new context set, it is necessary to provide a description of the semantics of each item within it. While context sets may contain indexes, relations, relation modifiers and boolean modifiers, there is no requirement that all should be present; in fact it is expected that most context sets will only define indexes.

Each context set has a unique identifier, a URI. When sending the context set in a query, a short form is used. These short names may be sent as a mapping within the query itself, or be published by the recipient of the query in some protocol dependent fashion. The prefix 'cql' is reserved for the base CQL context set, but authors may wish to recommend a short name for use with their set.

An index, relation, or modifier qualified by a context is represented in the form prefix.value, where prefix is a short name for a unique context set identifier.

3.3.1 The CQL Context Set

The CQL context set defines a set of indexes, relations and relation modifiers. The indexes supplied are 'utility' indexes which are generally useful across all applications of the language. These utility indexes are for instances when CQL is required to express a concept not directly related to the records, or for indexes applicable in practically every context.
The reserved name for this context set is: cql

The identifier for this context set is: info:srw/cql-context-set/1/cql-v1.2

### 3.3.1.1 Indexes

**resultSetId**

A search clause may be a result set id. This is a special case, where the index and relation are expressed as "cql.resultSetId =" and the term is the result set id returned by the server in the 'resultSetId' parameter of the SRU response. It may also be used by itself in a query to refer to an existing result set from which records are desired. It may also be used in conjunction with other resultSetId clauses or other indexes, combined by boolean operators. The semantics of resultSetId with relations other than "=" is undefined. The semantics of resultSetId with scan is also undefined.

Examples:

```
cql.resultSetId = "5940824f-a2ae-41d0-99af-9a20bc4047b1"
```

*Match the result set with the given identifier.*

**allRecords**

A special index which matches every record available. Every record is matched no matter what values are provided for the relation and term, but the recommended syntax is: cql.allRecords = 1. The semantics for scanning allRecords is not defined.

Examples:

```
cql.allRecords = 1 NOT dc.title = fish
```

*Search for all records that do not match 'fish' as a word in title.*

**allIndexes**

Alias: anywhere

The 'allIndexes' index will result in a search equivalent to searching all of the indexes (in all of the context sets) that the server has access to. The semantics for scanning allIndexes is not defined.

Examples:

```
cql.allIndexes = fish
```

*If the server had three indexes title, creator and date, then this would be the same as title = fish or creator = fish or date = fish*

**anyIndexes**

Alias: serverChoice

The 'anyIndexes' index allows the server to determine how to search for the given term. The server
may choose one or more indexes in which to search, which may or may not be generally available via
CQL. It may choose a different index to search every time, based on the term for example, and hence
may not produce consistent results via scan.

This is the default when the index and relation is omitted from a search clause. The relation used when the
index is omitted is ‘=’.

Examples:

```
" cql.anyIndexes = fish
Search in any one or more indexes for the term fish
```

keywords

The keywords index is an index of terms from the record, determined by the server as being
generally descriptive or meaningful to search on. It might include the full text of a document,
descriptive metadata fields, or anything else generally useful to search as an initial entry point to the
data. Exactly which fields make up this index is determined by the server, however the choice must
be consistent, unlike anyIndexes above, when the choice can be different for different searches.

Examples:

```
" cql.keywords any/relevant "code computer calculator programming"
Search in descriptive locations for the given term
```

3.3.1.2 Relations

Implicit Relations

These relations are defined as such in the grammar of CQL. The cql context set only defines their meaning,
rather than their existence.

`=`

This is the default relation, and the server can choose any appropriate relation or means of
comparing the query term with the terms from the data being searched. If the term is numeric, the
most commonly chosen relation is '=='. For a string term, either 'adj' or '==' as appropriate for the
index and term.

Examples:

```
" animal.numberOfLegs = 4
Recommended to use '=='
```

" dc.identifier = "gb 141 staff a-m"
  Recommended to use '=='

" dc.title = "lord of the rings"
  Recommended to use 'adj'

" dc.date = "2004 2006"
  Recommended to use 'within'

==
This relation is used for exact equality matching. The term in the data is exactly equal to the term in
the search.

Examples:

" dc.identifier == "gb 141 staff a-m"
  Search for the string 'gb 141 staff a-m' in the identifier index.

" dc.date == "2006-09-01 12:00:00"
  Search for the given datestamp.

" animal.numberOfLegs == 4
  Search for animals with exactly 4 legs.

<>
This relation means 'not equal to' and matches anything which is not exactly equal to the search
term.

Examples:

" dc.date <> 2004-01-01
  Search for any date except the first of January, 2004

" dc.identifier <> ""
  Search for any identifier which is not the empty string.

<, >, <, =, >=
These relations retain their regular meanings as pertaining to ordered terms (less than, greater
than, less than or equal to, greater than or equal to).

Examples:

" dc.date > 2006-09-01
Search for dates after the 1st of September, 2006

animal.numberOfLegs < 4
Search for animals with less than 4 legs.

Defined Relations
These relations are defined as being widely useful as part of a default context set.

adj
This relation is used for phrase searches. All of the words in the search term must appear, and must be adjacent to each other in the record in the order of the search term. The query could also be expressed using the PROX boolean operator.

Examples:

" dc.title adj "lord of the rings"
Search for the phrase 'lord of the rings' somewhere in the title.

" dc.description adj "blue shirt"
Search for 'blue' followed by 'shirt' in the description.

all, any
These relations may be used when the term contains multiple items to indicate "all of these items" or "any of these items". These queries could be expressed using boolean AND and OR respectively. These relations have an implicit relation modifier of 'cql.word', which may be changed by use of alternative relation modifiers.

Examples:

" dc.title all "lord rings"
Search for both lord and rings in the title.

" dc.description any "computer calculator"
Search for either computer or calculator in the description.

within
Within may be used with a search term that has multiple dimensions. It matches if the database's term falls completely within the range, area or volume described by the search term, inclusive of the extents given.

Examples:

" dc.date within "2002 2003"
Search for dates between 2002 and 2003 inclusive.

animal.numberOfLegs within "2 5"
Search for animals that have 2, 3, 4 or 5 legs.

encloses
Conversely, encloses is used when the index's data has multiple dimensions. It matches if the database's term fully encloses the search term.

Examples:

foo.dateRange encloses 2002
Search for ranges of dates that include the year 2002.

geo.area encloses "45.3, 19.0"
Search for any area that encloses the point 45.3, 19.0

3.3.1.3 Relation Modifiers

Functional Modifiers

stem
The server should apply a stemming algorithm to the words within the term. For example such that computing and computer both match the stem of 'compute'.

relevant
The server should use a relevancy algorithm for determining matches and the order of the result set.

phonetic
The server should use a phonetic algorithm for determining words which sound like the term.

fuzzy
The server should be liberal in what it counts as a match. The exact details of this are left up to the server, but might include permutations of character order, off-by-one for numerical terms and so forth.

partial
When used with within or encloses, there may be some section which extends outside of the term. This permits for the database term to be partially enclosed, or fall partially within the search term.

ignoreCase, respectCase
The server is instructed to either ignore or respect the case of the search term, rather than its default behaviour (which is unspecified). This modifier may be used in sort keys to ensure that terms
with the same letters in different cases are sorted together or separately, respectively.

\begin{itemize}
\item \textbf{ignoreAccents, respectAccents}
\begin{itemize}
\item The server is instructed to either ignore or respect diacritics in terms, rather than its default behaviour (which is unspecified, but respectAccents is recommended). This modifier may be used in sort keys, to ensure that characters with diacritics are sorted together or separately from those without them.
\end{itemize}
\item \textbf{locale=value}
\begin{itemize}
\item The term should be treated as being from the specified locale. Locales will in general include specifications for whether sort order is case-sensitive or insensitive, how it treats accents, and so forth. The default locale is determined by the server. The value is usually of the form C, french, \texttt{fr\_CH}, \texttt{fr\_CH.iso88591} or similar. This modifier may be used in sort keys.
\end{itemize}
\end{itemize}

\textbf{Examples:}

\begin{itemize}
\item \texttt{dc.title any/stem "computing disestablishmentarianism"}
\begin{itemize}
\item Find the local stemmed form of 'computing' and 'disestablishmentarianism', and search for those stems in the stemmed forms of the terms in titles.
\end{itemize}
\item \texttt{person.phoneNumber =/fuzzy "0151 795-4252"}
\begin{itemize}
\item Search for a phone number which is something similar to '0151 795-4252' but not necessarily exactly that
\end{itemize}
\item \texttt{"fish" sortBy dc.title/ignoreCase}
\begin{itemize}
\item Search for 'fish', and then sort the results by title, case insenstively.
\end{itemize}
\item \texttt{dc.title within/locale=fr "l m"}
\begin{itemize}
\item Find all titles between l and m, ensure that the locale is 'fr' for determining the order for what is between l and m.
\end{itemize}
\end{itemize}

\textbf{Term-format Modifiers}

These modifiers specify the format of the search term to ensure that the correct comparison is performed by the server. These modifiers may all be used in sort keys.

\begin{itemize}
\item \textbf{word}
\begin{itemize}
\item The term should be broken into words, according to the server's definition of a 'word'
\end{itemize}
\item \textbf{string}
\begin{itemize}
\item The term is a single item, and should not be broken up.
\end{itemize}
\end{itemize}
Each item within the term conforms to the ISO 8601 specification for expressing dates.

Number
Each item within the term is a number.

URI
Each item within the term is a URI.

Object identifier
Each item within the term is an ISO object identifier, dot-separated format.

Examples:

dc.title =/string Jaws
Search in title for the string 'Jaws', rather than Jaws as a word. (Equivalent to the use of == as the relation)

zeerex.set ==/oid "1.2.840.10003.3.1"
Search for the given OID as an attribute set.

squirrel sortby numberOfLegs/number
Search for squirrel, and sort by the numberOfLegs index ensuring that it is treated as a number, not a string. (eg '2' would sort after '10' as a string, but before it as a number)

Masking

The following masking rules and special characters apply for search terms, unless overridden in a profile via a relation modifier. To explicitly request this functionality, add 'cql.masked' as a relation modifier.

A single asterisk (*) is used to mask zero or more characters.

A single question mark (?) is used to mask a single character, thus N consecutive question-marks means mask N characters.

Carat/hat (^) is used as an anchor character for terms that are word lists, that is, where the relation is 'all' or 'any', or 'adj'. It may not be used to anchor a string, that is, when the relation is '==' (string matches are, by default, anchored). It may occur at the beginning or end of a word (with no intervening space) to mean right or left anchored."^" has no special meaning when it occurs within a word (not at the beginning or end) or string but must be escaped nevertheless.
Backslash (\) is used to escape '*', '?', quote ('"') and '^', as well as itself. Backslash not followed immediately by one of these characters is an error.

Examples:

```
"dc.title = c*t
Matches words that start with c and end in t
"

"dc.title adj "*fish food*"
Matches a word that ends in fish, followed by a word that starts with food
"

"dc.title = c?t
Matches a three letter word that starts with c and ends in t.
"

"dc.title adj "^cat in the hat"
Matches 'cat in the hat' where it is at the beginning of the field
"

"dc.title any "^cat ^dog rat^"
Matches cat at the beginning, dog at the beginning or rat at the end
"

"dc.title == ""Of Couse", she said"
Escape internal double quotes within the term.
"

! unmasked
Do not apply masking rules, all characters are literal.
"

! substring
The 'substring' modifier may be used to specify a range of characters (first and last character) indicating the desired substring within the field to be searched. The modifier takes a value, of the form "start:end" where start and end obey the following rules:

"Positive integers count forwards through the string, starting at 1. The first character is 1, the
tenth character is 10.
"

"Negative integers count backwards through the string, with -1 being the last character.
"

"Both start and end are inclusive of that character.
"

"If omitted, start defaults to 1 and end defaults to -1.
"

Examples:

```
"marc.008 =/substring="1:6" 920102"
**regexp**

The term should be treated as a regular expression. Any features beyond those found in modern POSIX regular expressions are considered to be server dependent. This modifier overrides the default 'masked' modifier, above. It may be used in either a string or word context.

Examples:

```
" dc.title adj/regexp "(lord|king|ruler) of th[ea] r.*s"
Match lord or king or ruler, followed by of, followed by the or tha, followed by r plus zero or more characters plus s
```

### 3.3.1.4 Booleans

The CQL context set does not define booleans, as these can only be defined by the CQL grammar. It gives the semantics of the booleans defined.

**AND**

The combination of two sets of records with AND will result in the set of records that appear in both of the sets.

**OR**

The combination of two sets of records with OR will result in the set of records that appear in either or both of the sets. It is therefore inclusive OR, not exclusive OR.

**NOT**

The combination of two sets of records with NOT will result in the set of records that appear in the left set, but not in the right hand set. It cannot be used as a unary operator.

**PROX**

The prox (short for proximity) boolean operator allows for the relative locations of the terms to be used in order to determine the resulting set of records. The semantics of when a match occurs is defined by the modifiers or defaults for those modifiers, as described below.

### 3.3.1.5 Boolean Modifiers

The CQL context set defines four boolean modifiers, which are only used with the prox boolean operator.
distance symbol value
The distance that the two terms should be separated by.

Symbol is one of: < > <= >= = <>
If the modifier is not supplied, it defaults to <=.

Value is a non-negative integer.
If the modifier is not supplied, it defaults to 1 when unit=word, or 0 for all other units.

unit=value
The type of unit for the distance.
Value is one of: 'paragraph', 'sentence', 'word' and 'element', and defaults to 'word'. These values are explicitly undefined. They are subject to interpretation by the server. See Proximity Units.

unordered
The order of the two terms is unimportant. This is the default.

ordered
The order of the two terms must be as per the query.

Examples:

cat prox/unit=word/distance>2/ordered hat
Find 'cat' where it appears more than two words before 'hat'

cat prox/unit=paragraph hat
Find cat and hat appearing in the same paragraph (distance defaulting to 0) in either order (unordered default)

zeerex.set = cql prox/unit=element/distance=0 zeerex.index = resultSetId
Find the cql context set in the same element as the index name resultSetId. E.g. search for cql.resultSetIds

3.3.1.6 Proximity Units

As noted above proximity units 'paragraph', 'sentence', 'word' and 'element' are explicitly undefined, that is, they are undefined when used by the CQL context set. Other context sets may assign them specific values.

Thus compare "prox/unit=word" with "prox/xyz.unit=word". In the first, 'unit' is a prox modifier from the CQL set, and as such its values are undefined, so 'word' is subject to interpretation by the server. In the second, 'unit' is a prox modifier defined by the xyz context set, which may assign the unit 'word' a specific
meaning.

Other context sets may define additional units, for example, 'street':

prox/xyz.unit="street"

Note that this approach, 'prox/xyz.unit="street"', is preferable to 'Prox/unit=xyz.street'. In the first case, 'unit' is a modifier defined in the xyz context set, and 'street' is a value defined for that modifier. In the second, 'unit' is a modifier from the cql context set, with a value defined in a different set. so its value would have to be one that is defined in the cql context set. Pairing a modifier from one set with a value from another is not a good practice.
4. Diagnostics

Sometimes things go wrong. In these cases the server is obliged to report that something went wrong, by sending a diagnostic record explaining what happened. A list of currently supported diagnostics is available and more may be identified in the future.

Diagnostics fall into two categories, 'fatal' and 'non-fatal'. A fatal diagnostic is one in which the execution of the request cannot proceed and no records are available to return. For example, if the client supplied an invalid query there is nothing that the server can do. A non-fatal diagnostic on the other hand 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 which are available rather than failing the entire request.

Non-fatal diagnostics are also divided into two categories 'surrogate' and 'non-surrogate'. Surrogate diagnostics take the place of a record. For example if the second of three records was not available in the requested schema, then the response would include the first record, a surrogate diagnostic explaining that the second record is not available, and then the final record. Non-surrogate, non-fatal diagnostics are diagnostics saying that while some or all the records are available, something else went wrong. For example the requested sorting algorithm might not be available.

Surrogate diagnostics take the place of the record for which they are a surrogate in the 'records' parameter of the response. Non-surrogate records, both fatal and non-fatal, are returned in the 'diagnostics' parameter.

4.1 Diagnostic Schema

Diagnostics are returned in a very simple schema which has only three elements, 'uri', 'details' and 'message'.

The required 'uri' field is a URI, identifying the particular diagnostic. When the URI begins with "info:srw/diagnostic/1/" (for example, 'info:srw/diagnostic/1/7') then the diagnostic is from the standard SRU diagnostic list. The 'details' part contains information specific to the diagnostic, format as specified by the individual diagnostic definition. The 'message' field contains a human readable message to be displayed. Only the uri field is required, the other two are optional.

It is recommended for all diagnostics that the final section should be a distinguishing integer (for example 'http://srw.cheshire3.org/diagnostics/1')

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Occurence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uri</td>
<td>xsd:anyURI</td>
<td>Mandatory</td>
<td>The diagnostic's</td>
</tr>
</tbody>
</table>
identifying URI.

Any supplementary information available, often in a format specified by the diagnostic.

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.

Examples

Non-surrogate, fatal diagnostic:

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

Surrogate, non-fatal diagnostic:

```
<records>
<record>
<recordSchema>info:srw/schema/1/diagnostics-v1.1</recordSchema>
<recordData>
<diagnostic xmlns="http://www.loc.gov/zing/srw/diagnostic/">
<uri>info:srw/diagnostic/1/65</uri>
<message>Record deleted by another user.</message>
</diagnostic>
</recordData>
</record>
</records>
```
4.2 Diagnostics List

The diagnostics below are defined for use with the SRU 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 is 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. Some of the diagnostics from earlier versions of the standards have been deprecated, however they are still listed here, suitably marked, for reference. For additional explanation of these diagnostics, see Notes.

Diagnostic Categories: General | CQL | Result Sets | Records | Sorting | Explain | Stylesheets | Scan

General Diagnostics

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
<th>Details Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>General system error</td>
<td>note</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Debugging information (traceback)</td>
</tr>
<tr>
<td>2</td>
<td>System temporarily unavailable</td>
<td>note</td>
</tr>
<tr>
<td>3</td>
<td>Authentication error</td>
<td>note</td>
</tr>
<tr>
<td>4</td>
<td>Unsupported operation</td>
<td>note</td>
</tr>
<tr>
<td>5</td>
<td>Unsupported version</td>
<td>note</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Highest version supported</td>
</tr>
<tr>
<td>6</td>
<td>Unsupported parameter value</td>
<td>note</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Name of parameter</td>
</tr>
<tr>
<td>7</td>
<td>Mandatory parameter not supplied</td>
<td>note</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Name of missing parameter</td>
</tr>
<tr>
<td>8</td>
<td>Unsupported Parameter</td>
<td>note</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Name of the unsupported parameter</td>
</tr>
</tbody>
</table>

Diagnostics Relating to CQL

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
<th>Details Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Query syntax error</td>
<td>note</td>
</tr>
<tr>
<td>11</td>
<td>DEPRECATED. Supported query type</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Message</td>
<td>Note</td>
</tr>
<tr>
<td>---</td>
<td>-------------------------------------------------------------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>12</td>
<td>Too many characters in query</td>
<td>note</td>
</tr>
<tr>
<td>13</td>
<td>Invalid or unsupported use of parentheses</td>
<td>note</td>
</tr>
<tr>
<td>14</td>
<td>Invalid or unsupported use of quotes</td>
<td>note</td>
</tr>
<tr>
<td>15</td>
<td>Unsupported context set</td>
<td>note</td>
</tr>
<tr>
<td>16</td>
<td>Unsupported index</td>
<td>note</td>
</tr>
<tr>
<td>17</td>
<td>DEPRECATED. Unsupported combination of index and index set</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Unsupported combination of indexes</td>
<td>note</td>
</tr>
<tr>
<td>19</td>
<td>Unsupported relation</td>
<td>note</td>
</tr>
<tr>
<td>20</td>
<td>Unsupported relation modifier</td>
<td>note</td>
</tr>
<tr>
<td>21</td>
<td>Unsupported combination of relation modifiers</td>
<td>note</td>
</tr>
<tr>
<td>22</td>
<td>Unsupported combination of relation and index</td>
<td>note</td>
</tr>
<tr>
<td>23</td>
<td>Too many characters in term</td>
<td>note</td>
</tr>
<tr>
<td>24</td>
<td>Unsupported combination of relation and term</td>
<td>note</td>
</tr>
<tr>
<td>25</td>
<td>DEPRECATED. Special characters not quoted in term</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>Non special character escaped in term</td>
<td>note</td>
</tr>
<tr>
<td>27</td>
<td>Empty term unsupported</td>
<td>note</td>
</tr>
<tr>
<td>28</td>
<td>Masking character not supported</td>
<td>note</td>
</tr>
<tr>
<td>29</td>
<td>Masked words too short</td>
<td>note</td>
</tr>
<tr>
<td>30</td>
<td>Too many masking characters in term</td>
<td>note</td>
</tr>
<tr>
<td>31</td>
<td>Anchoring character not supported</td>
<td>note</td>
</tr>
<tr>
<td>32</td>
<td>Anchoring character in unsupported position</td>
<td>note</td>
</tr>
<tr>
<td>33</td>
<td>Combination of proximity/adjacency and masking characters not</td>
<td>note</td>
</tr>
</tbody>
</table>
supported

Combination of proximity/adjacency and anchoring characters not supported

Term contains only stopwords

Term in invalid format for index or relation

Unsupported boolean operator

Too many boolean operators in query

Proximity not supported

Unsupported proximity relation

Unsupported proximity distance

Unsupported proximity unit

Unsupported proximity ordering

Unsupported combination of proximity modifiers

DEPRECATED. Prefix assigned to multiple identifiers.

Unsupported boolean modifier

Cannot process query; reason unknown

Query feature unsupported

Masking character in unsupported position

Diagnostics Relating to Result Sets

Number Description Details Format

50 Result sets not supported

51 Result set does not exist Result set identifier

52 Result set temporarily unavailable Result set identifier

53 Result sets only supported for retrieval

54 DEPRECATED. Retrieval may only occur from
Combination of result sets with search terms not supported

DEPRECATED. Only combination of single result set with search terms supported

DEPRECATED. Result set created but no records available

Result set created with unpredictable partial results available

Result set created with valid partial results available

Result set not created: too many matching records

Diagnostics Relating to Records

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
<th>Details Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>61</td>
<td>First record position out of range</td>
<td>note</td>
</tr>
<tr>
<td>62</td>
<td>Negative number of records requested</td>
<td>DEPRECATED</td>
</tr>
<tr>
<td>63</td>
<td>System error in retrieving records</td>
<td>DEPRECATED</td>
</tr>
<tr>
<td>64</td>
<td>Record temporarily unavailable</td>
<td>note</td>
</tr>
<tr>
<td>65</td>
<td>Record does not exist</td>
<td>note</td>
</tr>
<tr>
<td>66</td>
<td>Unknown schema for retrieval</td>
<td>note</td>
</tr>
<tr>
<td>67</td>
<td>Record not available in this schema</td>
<td>note</td>
</tr>
<tr>
<td>68</td>
<td>Not authorised to send record</td>
<td>note</td>
</tr>
<tr>
<td>69</td>
<td>Not authorised to send record in this schema</td>
<td>note</td>
</tr>
<tr>
<td>70</td>
<td>Record too large to send</td>
<td>note</td>
</tr>
<tr>
<td>71</td>
<td>Unsupported record packing</td>
<td>note</td>
</tr>
<tr>
<td>72</td>
<td>XPath retrieval unsupported</td>
<td>note</td>
</tr>
</tbody>
</table>
73   XPath expression contains unsupported feature  

74   Unable to evaluate XPath expression  

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
<th>Details Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>Sort not supported</td>
<td>note</td>
</tr>
<tr>
<td>81</td>
<td>DEPRECATED. Unsupported sort type</td>
<td></td>
</tr>
<tr>
<td>82</td>
<td>Unsupported sort sequence</td>
<td>note</td>
</tr>
<tr>
<td>83</td>
<td>Too many records to sort</td>
<td>note</td>
</tr>
<tr>
<td>84</td>
<td>Too many sort keys to sort</td>
<td>note</td>
</tr>
<tr>
<td>85</td>
<td>DEPRECATED. Duplicate sort keys</td>
<td></td>
</tr>
<tr>
<td>86</td>
<td>Cannot sort: incompatible record formats</td>
<td>note</td>
</tr>
<tr>
<td>87</td>
<td>Unsupported schema for sort</td>
<td>note</td>
</tr>
<tr>
<td>88</td>
<td>Unsupported path for sort</td>
<td>note</td>
</tr>
<tr>
<td>89</td>
<td>Path unsupported for schema</td>
<td>note</td>
</tr>
<tr>
<td>90</td>
<td>Unsupported direction</td>
<td>note</td>
</tr>
<tr>
<td>91</td>
<td>Unsupported case</td>
<td>note</td>
</tr>
<tr>
<td>92</td>
<td>Unsupported missing value action</td>
<td>note</td>
</tr>
<tr>
<td>93</td>
<td>Sort ended due to missing value</td>
<td>note</td>
</tr>
</tbody>
</table>

Diagnostics Relating to Explain

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
<th>Details Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>DEPRECATED. Explain not supported (Use 4)</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>DEPRECATED. Explain request type not supported</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>DEPRECATED. Explain record temporarily unavailable (Use 64)</td>
<td></td>
</tr>
</tbody>
</table>

Diagnostics relating to Stylesheets
### Number  Description _  Details Format

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
<th>Details Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>110</td>
<td>Stylesheets not supported</td>
<td>note</td>
</tr>
<tr>
<td>111</td>
<td>Unsupported stylesheet</td>
<td>note</td>
</tr>
<tr>
<td></td>
<td>URL of stylesheet</td>
<td></td>
</tr>
</tbody>
</table>

**Diagnostics relating to Scan**

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
<th>Details Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>120</td>
<td>Response position out of range</td>
<td>note</td>
</tr>
<tr>
<td>121</td>
<td>Too many terms requested</td>
<td>note</td>
</tr>
<tr>
<td></td>
<td>maximum number of terms</td>
<td></td>
</tr>
</tbody>
</table>

**Notes**

<table>
<thead>
<tr>
<th>No.</th>
<th>added</th>
<th>Cat.</th>
<th>Description</th>
<th>Notes/Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>general</td>
<td>General system error</td>
<td>The server returns this error when it is unable to supply a more specific diagnostic. The server may also optionally supply debugging information.</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>general</td>
<td>System temporarily unavailable</td>
<td>The server cannot respond right now, perhaps because it's in a maintenance cycle, but will be able to in the future.</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>general</td>
<td>Authentication error</td>
<td>The request could not be processed due to lack of authentication.</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>general</td>
<td>Unsupported operation</td>
<td>Currently three operations are defined -- searchRetrieves, explain, and scan. searchRetrieves and explain are mandatory, so this diagnostic would apply only to scan, or in SRU where an undefined operation is sent.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>general</td>
<td>Unsupported version</td>
<td>This diagnostic might be returned for a searchRetrieves request which includes the recordPacking parameter with a value of 'xml', when the server does not support that</td>
</tr>
</tbody>
</table>
value. The diagnostic might supply the name of parameter, in this case 'recordPacking'.

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'.

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'.

The query was invalid, but no information is given for exactly what was wrong with it. Eg. dc.title foo fish (The reason is that foo isn't a valid relation in the default context set, but the server isn't telling you this for some reason)

The length (number of characters) of the query exceeds the maximum length supported by the server.

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)

The query couldn't be processed due to the use of quotes. Typically that they are mismatched Eg. 'fish'

A context set given in the query isn't known to the server. Eg. foo.title any fish

The index isn't known, possibly within a context set. Eg. dc.author any sanderson (dc has a creator index, not author)

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.

A relation in the query is unknown or unsupported. Eg. The server can't handle 'within' searches for dates, but can handle equality searches.

A relation modifier in the query is unknown or unsupported by the server. Eg. 'dc.title
<table>
<thead>
<tr>
<th>Query</th>
<th>Issue Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsupported combination of relation modifiers</td>
<td>Two (or more) relation modifiers can't be used together. Eg. dc.title any/cql.word/cql.string &quot;star fish&quot;</td>
</tr>
<tr>
<td>Unsupported combination of relation and index</td>
<td>While the index and relation are supported, they can't be used together. Eg. dc.author within &quot;I S&quot;</td>
</tr>
<tr>
<td>Too many characters in term</td>
<td>The term is too long. Eg. The server may simply refuse to process a term longer than a given length.</td>
</tr>
<tr>
<td>Unsupported combination of relation and term</td>
<td>The relation cannot be used to process the term. Eg dc.title within &quot;sanderson&quot;</td>
</tr>
<tr>
<td>Non special character escaped in term</td>
<td>Characters may be escaped incorrectly Eg &quot;\a\n\n\n&quot;</td>
</tr>
<tr>
<td>Empty term unsupported</td>
<td>Some servers do not support the use of an empty term for search or for scan. Eg: dc.title &gt; &quot;&quot;</td>
</tr>
<tr>
<td>Masking character not supported</td>
<td>A masking character given in the query is not supported. Eg. The server may not support * or ? or both</td>
</tr>
<tr>
<td>Masked words too short</td>
<td>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 *</td>
</tr>
<tr>
<td>Too many masking characters in term</td>
<td>The query has too many masking characters, so the server won't process them. Eg. dc.title any &quot;??a?f??b* *a?&quot;</td>
</tr>
<tr>
<td>Anchoring character not supported</td>
<td>The server doesn't support the anchoring character (^) Eg dc.title = &quot;^jaws&quot;</td>
</tr>
<tr>
<td>Anchoring character in unsupported position</td>
<td>The anchoring character appears in an invalid part of the term, typically the middle of a word. Eg dc.title any &quot;fi^sh&quot;</td>
</tr>
<tr>
<td>Combination of proximity/adjacency and masking characters not supported</td>
<td>The server cannot handle both adjacency (= relation for words) or proximity (the boolean) in combination with masking characters. Eg. dc.title = &quot;this is a titl* fo? a b*k&quot;</td>
</tr>
<tr>
<td>Combination of proximity/adjacency and anchoring characters not supported</td>
<td>Similarly, the server cannot handle anchoring characters.</td>
</tr>
<tr>
<td>Query</td>
<td>Message</td>
</tr>
<tr>
<td>-------</td>
<td>---------</td>
</tr>
<tr>
<td>35</td>
<td>Term contains only stopwords</td>
</tr>
<tr>
<td></td>
<td>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 contains only &quot;the&quot;</td>
</tr>
<tr>
<td>36</td>
<td>Term in invalid format for index or relation</td>
</tr>
<tr>
<td></td>
<td>This might happen when the index is of dates or numbers, but the term given is a word. Eg dc.date &gt; &quot;fish&quot;</td>
</tr>
<tr>
<td>37</td>
<td>Unsupported boolean operator</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>38</td>
<td>Too many boolean operators in query</td>
</tr>
<tr>
<td></td>
<td>There were too many search clauses given for the server to process.</td>
</tr>
<tr>
<td>39</td>
<td>Proximity not supported</td>
</tr>
<tr>
<td></td>
<td>Proximity is not supported at all.</td>
</tr>
<tr>
<td>40</td>
<td>Unsupported proximity relation</td>
</tr>
<tr>
<td></td>
<td>The relation given for the proximity is unsupported. Eg the server can only process = and &gt; was given.</td>
</tr>
<tr>
<td>41</td>
<td>Unsupported proximity distance</td>
</tr>
<tr>
<td></td>
<td>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</td>
</tr>
<tr>
<td>42</td>
<td>Unsupported proximity unit</td>
</tr>
<tr>
<td></td>
<td>The unit of proximity is unsupported, possibly because it is not defined.</td>
</tr>
<tr>
<td>43</td>
<td>Unsupported proximity ordering</td>
</tr>
<tr>
<td></td>
<td>The server cannot process the requested order or lack thereof for the proximity boolean</td>
</tr>
<tr>
<td>44</td>
<td>Unsupported combination of proximity modifiers</td>
</tr>
<tr>
<td></td>
<td>While all of the modifiers are supported individually, this particular combination is not.</td>
</tr>
<tr>
<td>46</td>
<td>Unsupported boolean modifier</td>
</tr>
<tr>
<td></td>
<td>A boolean modifier on the request isn't supported.</td>
</tr>
<tr>
<td>47</td>
<td>Cannot process query; reason unknown</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>48</td>
<td>Query feature unsupported</td>
</tr>
<tr>
<td></td>
<td>The server is able (contrast with 47) to tell you that something you asked for is not supported.</td>
</tr>
<tr>
<td>49</td>
<td>Masking character in unsupported</td>
</tr>
<tr>
<td></td>
<td>eg, a server that can handle xyz* but not *xyz or x^n yz</td>
</tr>
<tr>
<td>Page</td>
<td>Result Set Event</td>
</tr>
<tr>
<td>------</td>
<td>-----------------</td>
</tr>
<tr>
<td>50</td>
<td>Result set not supported</td>
</tr>
<tr>
<td>51</td>
<td>Result set does not exist</td>
</tr>
<tr>
<td>52</td>
<td>Result set temporarily unavailable</td>
</tr>
<tr>
<td>53</td>
<td>Result sets only supported for retrieval</td>
</tr>
<tr>
<td>55</td>
<td>Combination of result sets with search terms not supported</td>
</tr>
<tr>
<td>58</td>
<td>Result set created with unpredictable partial results available</td>
</tr>
<tr>
<td>59</td>
<td>Result set created with valid partial results available</td>
</tr>
<tr>
<td>60</td>
<td>Result set not created: too many matching records</td>
</tr>
<tr>
<td>61</td>
<td>First record position out of range</td>
</tr>
<tr>
<td>64</td>
<td>Record temporarily unavailable</td>
</tr>
<tr>
<td>65</td>
<td>Record does not exist</td>
</tr>
<tr>
<td>66</td>
<td>Unknown schema for retrieval</td>
</tr>
<tr>
<td>67</td>
<td>Record not available in this schema</td>
</tr>
<tr>
<td>68</td>
<td>Not authorised to</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>send record</td>
<td>The record can be retrieved in other schemas, but the one requested requires further authorisation.</td>
</tr>
<tr>
<td>records</td>
<td>Not authorised to send record in this schema</td>
</tr>
<tr>
<td>records</td>
<td>Record too large to send</td>
</tr>
<tr>
<td>records</td>
<td>Unsupported record packing</td>
</tr>
<tr>
<td>records</td>
<td>XPath retrieval unsupported</td>
</tr>
<tr>
<td>records</td>
<td>XPath expression contains unsupported feature</td>
</tr>
<tr>
<td>records</td>
<td>Unable to evaluate XPath expression</td>
</tr>
<tr>
<td>sort</td>
<td>Sort not supported</td>
</tr>
<tr>
<td>sort</td>
<td>Unsupported sort sequence</td>
</tr>
<tr>
<td>sort</td>
<td>Too many records to sort</td>
</tr>
<tr>
<td>sort</td>
<td>Too many sort keys to sort</td>
</tr>
<tr>
<td>sort</td>
<td>Cannot sort: incompatible record formats</td>
</tr>
<tr>
<td>sort</td>
<td>Unsupported schema for sort</td>
</tr>
<tr>
<td>sort</td>
<td>Unsupported path</td>
</tr>
</tbody>
</table>

The server supports only one of string or xml, or the client requested a record Packing which is unknown.
The server does not support the retrieval of nodes from within the record.
Some aspect of the XPath expression is unsupported. For example, the server might be able to process element nodes, but not functions.
The server could not evaluate the expression, either because it was invalid or it lacks some capability.
The server cannot perform any sort; that is the server only returns data in the default sequence.
The particular sequence of sort keys is not supported, but the keys may be supported individually.
used when the server will only sort result sets under a certain size and the request returned a set larger than that limit.
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".
The result set includes records in different schemas and there is insufficient commonality among the schemas to enable a 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.
for sort

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. For example, 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 Dec 2005 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.

120 scan Response position out of range

Say you ask for 500 terms and the server has a (fixed) maximum of 300. It would supply a value of '300' for details. If 'details' is not supplied, this might mean that the server doesn't have a fixed maximum and was just unable to deliver all the requested terms.
5. Explain Operation

The Explain operation allows a client to retrieve a description of the facilities available at an SRU server. It can then be used by the client to self-configure and provide an appropriate interface to the user. The record is in XML and follows the ZeeRex Schema.

There are two methods for getting the explain record:

- Via the SRU Explain Operation.
- Via the http GET request at the base URL for the service (no parameters).

This can be considered an SRU request (and hence have a default recordPacking of 'xml'), with no extraRequestData and leaving it up to the server to determine the version of the response. Otherwise, the response is identical to an SRU explainResponse message.

Parameters

Request Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>occurence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>operation</td>
<td>Mandatory</td>
<td>The string: 'explain'.</td>
</tr>
<tr>
<td>version</td>
<td>Mandatory</td>
<td>The version of the request, and a statement by the client that it wants the response to be less than, or preferably equal to, that version. See Versions.</td>
</tr>
<tr>
<td>recordPacking</td>
<td>Optional</td>
<td>A string to determine how the explain record should be escaped in the response. Defined values are 'string' and 'xml'. The default is 'xml'. See Records.</td>
</tr>
<tr>
<td>stylesheet</td>
<td>Optional</td>
<td>A URL for a stylesheet. The client requests that the server simply return this URL in the response. See Stylesheets.</td>
</tr>
<tr>
<td>extraRequestData</td>
<td>Optional</td>
<td>Provides additional information for the server to process. See Extensions.</td>
</tr>
</tbody>
</table>

Response Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>occurence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>xsd:string</td>
<td>Mandatory</td>
<td>The version of the response. This MUST be less than or equal to the</td>
</tr>
</tbody>
</table>
version requested by the client. >> See Versions.

A single Explain record, wrapped in the record metadata fields. >> See Records.

Additional information returned by the server. >> See Extensions.

The request parameters echoed back to the client in a simple XML form. >> See Echoing the Request

### 5.1 ZeeRex for SRU

ZeeRex/SRU Summary:

- The protocol attribute on the serverInfo element MUST have the value: SRU
- The transport attribute on the serverInfo element MUST be one of: http or https
- The method attribute on the serverInfo element MUST be a space separated list, comprising any number of the following values: GET POST SOAP
- The database element within serverInfo MUST contain the path section of the URL to the server, without the first / and up to the ?
- The set element within indexInfo is used to define the short names of context sets.
- Indexes are described by including the name of the index in the name element within map, and the short name for the context set in the set attribute on that element.
- The schemaInfo section is used to described the schemas supported by the server.

**EXAMPLES**

The following URLs would all retrieve the explain document:

http://myserver.com/cgi/mysru?operation=explain&version=1.1&recordPacking=xml
http://myserver.com/cgi/mysru?
http://myserver.com/cgi/mysru
The corresponding response from the server would be:

```xml
<srw:explainResponse xmlns:srw="http://www.loc.gov/zing/srw/">
<srw:version>1.1</srw:version>
<srw:record>
<srw:recordPacking>XML</srw:recordPacking>
<srw:recordSchema>http://explain.z3950.org/dtd/2.1/</srw:recordSchema>
<srw:recordData>

<zr:explain xmlns:zr="http://explain.z3950.org/dtd/2.1/">
<zr:serverInfo protocol="SRU" version="1.2" transport="http" method="GET POST SOAP">
<zr:host>myserver.com</zr:host>
<zr:port>80</zr:port>
<zr:database>cgi/mysru</zr:database>
</zr:serverInfo>
<zr:databaseInfo>
<title lang="en" primary="true">SRU Test Database</title>
</zr:databaseInfo>
<zr:indexInfo>
<zr:set name="dc" identifier="info:srw/cql-context-set/1/dc-v1.1"/>
<zr:index>
<zr:map><zr:name set="dc">title</zr:name></zr:map>
</zr:index>
</zr:indexInfo>
<zr:schemaInfo>
<zr:schema name="dc" identifier="info:srw/schema/1/dc-v1.1">
<title>Simple Dublin Core</title>
</zr:schema>
</zr:schemaInfo>
<zr:configInfo>
<zr:default type="numberOfRecords">1</zr:default>
<zr:setting type="maximumRecords">50</zr:setting>
<zr:supports type="proximity"/>
</zr:configInfo>
</zr:explain>

</srw:recordData>
</srw:record>
</srw:explainResponse>
```
6. Scan Operation

While the searchRetrieve operation enables searches for a specific terms within the records, the scan operation allows the client to request a range of the available terms at a given point within a list of indexed terms. This enables clients to present an ordered list of values and, if supported, how many hits there would be for a search on that term. Scan is often used to select terms for subsequent searching or to verify a negative search result.

The index to be browsed and the start point within it is given in the scanClause parameter as a complete index, relation, term clause in CQL. The relation and relation modifiers may be used to determine the format of the terms returned. For example 'dc.title any fish' will return a list of keywords, whereas 'dc.title exact fish' would return a list of full title fields. Range relations, such as <, >=, within and so forth, are prohibited for use with scan, and diagnostic 'info:srw/diagnostic/1/19' should be returned. See below for a clarifying example.

The term given in the clause is the position within the ordered list of terms at which to start, however see the responsePosition parameter below for more information. If the empty term is given, then even if searching for it is unsupported by the server, it may be interpreted as the beginning of the term list.

6.1 Parameters

Request Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Occurence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>operation</td>
<td>mandatory</td>
<td>The string: 'scan'.</td>
</tr>
<tr>
<td>version</td>
<td>mandatory</td>
<td>The version of the request, and a statement by the client that it wants the response to be less than, or preferably equal to, that version. See Versions.</td>
</tr>
<tr>
<td>scanClause</td>
<td>mandatory</td>
<td>The index to be browsed and the start point within it, expressed as a complete index, relation, term clause in CQL. See CQL.</td>
</tr>
<tr>
<td>responsePosition</td>
<td>optional</td>
<td>The position within the list of terms returned where the client would like the start term to occur. If the position given is 0, then the term should be immediately before the first term in the response. If the position given is 1, then the term should be first in the list, and so forth up to the number of terms requested plus 1, meaning that the term should be immediately after the last term in the response, even if the number of terms returned is less than the number</td>
</tr>
</tbody>
</table>
maximumTerms | optional | The number of terms which the client requests be returned. The actual number returned may be less than this, for example if the end of the term list is reached, but may not be more. The explain record for the database may indicate the maximum number of terms which the server will return at once. All positive integers are valid for this parameter. If not specified, the default is server determined.

stylesheet | optional | A URL for a stylesheet. The client requests that the server simply return this URL in the response. See Stylesheets.

extraRequestData | optional | Provides additional information for the server to process. See Extensions.

Example:

http://myserver.com/sru?operation=scan&version=1.2&scanClause=dc.title=frog
&responsePosition=1&maximumTerms=25

Response Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Occurrence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>xsd:string</td>
<td>mandatory</td>
<td>The version of the response. This MUST be less than or equal to the version requested by the client. See Versions.</td>
</tr>
<tr>
<td>terms</td>
<td>sequence of &lt;term&gt;</td>
<td>optional</td>
<td>A sequence of terms which match the request. See Terms</td>
</tr>
<tr>
<td>diagnostics</td>
<td>sequence of &lt;diagnostic&gt;</td>
<td>Optional</td>
<td>A sequence of non surrogate diagnostics generated during execution. See Diagnostics.</td>
</tr>
<tr>
<td>extraResponseData</td>
<td>xmlFragment</td>
<td>Optional</td>
<td>Additional information returned by the server. See Extensions.</td>
</tr>
</tbody>
</table>
echoedScanRequest  <echoedScanRequest>  Optional

The request parameters echoed back to the client in a simple XML form. See Echoing the Request.

6.2 Terms

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>occurrence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>xsd:string</td>
<td>mandatory</td>
<td>The term, exactly as it appears in the index.</td>
</tr>
<tr>
<td>numberOfRecords</td>
<td>xsd:nonNegativeInteger</td>
<td>optional</td>
<td>The number of records which would be matched if the index in the request's scanClause was searched with the term in the 'value' field.</td>
</tr>
<tr>
<td>displayTerm</td>
<td>xsd:string</td>
<td>optional</td>
<td>A string to display to the end user in place of the term itself. For example this might add back in diacritics or capitalisation which do not appear in the index.</td>
</tr>
<tr>
<td>whereInList</td>
<td>xsd:string</td>
<td>optional</td>
<td>A flag to indicate the position of the term within the complete term list. It must be one of the following values: 'first' (the first term), 'last' (the last term), 'only' (the only term) or 'inner' (any other term)</td>
</tr>
<tr>
<td>extraTermData</td>
<td>xmlFragment</td>
<td>optional</td>
<td>Additional information concerning the term. See Extensions.</td>
</tr>
</tbody>
</table>
6.3 Example Scan Response

```xml
<sru:scanResponse xmlns:srw="http://www.loc.gov/zing/srw/
xmlns:diag="http://www.loc.gov/zing/srw/diagnostic/
xmlns:myServer="http://myServer.com/">
  <sru:version>1.1</sru:version>
  <sru:terms>
    <sru:term>
      <sru:value>cartesian</sru:value>
      <sru:numberOfRecords>35645</sru:numberOfRecords>
      <sru:displayTerm>Carthesian</sru:displayTerm>
    </sru:term>
    <sru:term>
      <sru:value>carthesian</sru:value>
      <sru:numberOfRecords>2154</sru:numberOfRecords>
      <sru:displayTerm>CarthÉsian</sru:displayTerm>
    </sru:term>
    <sru:term>
      <sru:value>cat</sru:value>
      <sru:numberOfRecords>8739972</sru:numberOfRecords>
      <sru:displayTerm>Cat</sru:displayTerm>
    </sru:term>
    <sru:term>
      <sru:value>catholic</sru:value>
      <sru:numberOfRecords>35</sru:numberOfRecords>
      <sru:displayTerm>Catholic</sru:displayTerm>
      <sru:whereInList>last</sru:whereInList>
      <sru:extraTermData>
        <myServer:ID>4456888</myServer:ID>
      </sru:extraTermData>
    </sru:term>
  </sru:terms>
  <sru:echoedScanRequest>
    <sru:version>1.1</sru:version>
    <sru:scanClause>dc.title="cat"</sru:scanClause>
    <sru:maximumTerms>3</sru:maximumTerms>
    <sru:stylesheet>http://myServer.com/myStyle</sru:stylesheet>
    <sru:echoedScanRequest/>
  </sru:echoedScanRequest>
</sru:scanResponse>
```