FCS 2 Endpoint Developer's tutorial

Version: 2016-01

Requirements

- Reference libraries: SRUServer, SRUClient, FCS-QL or your own selected FCS 2.0 and SRU 2.0 compatible libraries.
- Endpoint reference library: FCSSimpleEndpoint or you own from scratch.
- Translation library (optional)

Resources

- Specifications
  - FCS 2.0 specification
  - SRU 2.0 specification
- Maven dependencies
  - Reference libraries: server, client, and endpoint (simple as well as other ones). See Configuration section.

Adaptation

The easiest way to get started is to adapt the FCSSimpleEndpoint.

SRUSearchEngine/SRUSearchEngineBase

By extending the *SimpleEndpointSearchEngineBase*, or if it suits your search engine's needs better the *SRUSearchEngineBase* directly, you adapt the behaviour to your search engine. A few notes:

- do not override *init()* use *doInit()*.
- If you need to do cleanup do not override *destroy()* use *doDestroy()*.
- Implementing the *scan* method is optional. If you want to provide custom *scan* behavior for a different index, override the *doScan()* method.
- Implementing the *explain* method is optional. Only needed if you need to fill writeExtraResponseData block of the SRU response. The implementation of this method must be thread-safe. The SimpleEndpointSearchEngineBase implementation has a on request parameter only response of SRUExplainResult with diagnostics.

Initialize the search engine

The initialization should be tailored towards your environment and needs. You need to provide the context (ServletContext), config (SRUServerConfig) and a query parser builder SRUQueryParserRegistry.Builder if you want to register additional query parsers. In addition you can provide parameters gathered from servlet configuration and the servlet context.

EndpointDescription

SimpleEndpointDescription is an implementtion of an endpoint description that is initialized from static information supplied at construction time. You will probably use the *SimpleEndpointDescriptionParser* to provide the endpoint description, but you can generate the list of resource info records in any way suitable to your situation. Though probably this is not the first behaviour you need to adapt since it supports both URL or w3 Document instantiation.

EndPointDescriptionParser

The SimpleEndpointDescriptionParser is able to do the heavy lifting for you by parsing and extracting the information from the endpoint description including everything needed for basic and required FCS 2.0 features like capabilities, supported layers and dataviews, resource enumeration etc. It also already provide simpe consistency checks like checking unique IDs and that the declared capabilities and dataviews match. See Configuration section for further details.

SRUSearchResultSet

This class needs to be implemented to support your search engine's behaviour. Implement these methods:

writeRecord(), getResultCountPrecision(), getRecordIdentifier(), nextRecord(), getRecordSchemaIdentifier(), getRecordCount(), and *getTotalRecordCount()*.

SRUScanResultSet

This class needs to be implemented to support your search engine's beahviour. Implement these methods:

getWhereInList(), getNumberOfRecords(), getDisplayTerm(), getValue(), and *getNextTerm()*

SRUExplainResult

This class needs to be implemented to support your search engine's data source.

Code examples

In this section the most probable classes or methods to override or implement are walked through with code examples from one or more of the reference implementations.

```
if (request.isQueryType(Constants.FCS_QUERY_TYPE_FCS)) {
            /*
             * Got a FCS query (SRU 2.0).
             * Translate to a proper Lucene query
             */
            final FCSQueryParser.FCSQuery q =
                    request.getQuery(FCSQueryParser.FCSQuery.class);
            query = makeSpanQueryFromFCS(q);

request.isQueryType(Constants.FCS_QUERY_TYPE_FCS)) {
            /*
             * Got a FCS query (SRU 2.0).
             * Translate to a proper CQP query
             */
            final FCSQueryParser.FCSQuery q =
                    request.getQuery(FCSQueryParser.FCSQuery.class);
            query = makeCQPQueryFromFCS(q);

   private SpanQuery makeSpanQueryFromFCS(FCSQueryParser.FCSQuery query)
            throws SRUException {
        QueryNode tree = query.getParsedQuery();
        logger.debug("FCS-Query: {}", tree.toString());

        // crude query translator
        if (tree instanceof QuerySegment) {
            QuerySegment segment = (QuerySegment) tree;
            if ((segment.getMinOccurs() == 1) && (segment.getMaxOccurs() == 1))
{
                QueryNode child = segment.getExpression();
                if (child instanceof Expression) {
                    Expression expression = (Expression) child;
                    if (expression.getLayerIdentifier().equals("text") &&
                            (expression.getLayerQualifier() == null) &&
                            (expression.getOperator() == Operator.EQUALS) &&
                            (expression.getRegexFlags() == null)) {

                        return new SpanTermQuery(new Term("text",
                                expression.getRegexValue().toLowerCase()));
                    } else {
                        throw new SRUException(

Constants.FCS_DIAGNOSTIC_GENERAL_QUERY_TOO_COMPLEX_CANNOT_PERFORM_QUERY,
                                "Endpoint only supports 'text' layer, the '='
operator and no regex flags");
                    }
                } else {
                    throw new SRUException(

Constants.FCS_DIAGNOSTIC_GENERAL_QUERY_TOO_COMPLEX_CANNOT_PERFORM_QUERY,
                            "Endpoint only supports simple expressions");
                }
            } else {
                throw new SRUException(

Constants.FCS_DIAGNOSTIC_GENERAL_QUERY_TOO_COMPLEX_CANNOT_PERFORM_QUERY,
                        "Endpoint only supports default occurances in
segments");
            }
        } else {
            throw new SRUException(

Constants.FCS_DIAGNOSTIC_GENERAL_QUERY_TOO_COMPLEX_CANNOT_PERFORM_QUERY,
                    "Endpoint only supports single segment queries");
        }
```

```
    }

@Override
    public void writeRecord(XMLStreamWriter writer) throws XMLStreamException {
        XMLStreamWriterHelper.writeStartResource(writer, idno, null);
        XMLStreamWriterHelper.writeStartResourceFragment(writer, null, null);

        /*
         * NOTE: use only AdvancedDataViewWriter, even if we are only doing
         * legacy/simple FCS.
         * The AdvancedDataViewWriter instance could also be
         * reused, by calling reset(), if it was used in a smarter fashion.
         */
        AdvancedDataViewWriter helper =
                new AdvancedDataViewWriter(AdvancedDataViewWriter.Unit.ITEM);
        URI layerId = URI.create("http://endpoint.example.org/Layers/orth1");
        String[] words;

        long start = 1;
        if ((left != null) && !left.isEmpty()) {
            words = left.split("\\s+");
            for (int i = 0; i < words.length; i++) {
                long end = start + words[i].length();
                helper.addSpan(layerId, start, end, words[i]);
                start = end + 1;
            }
        }

        words = keyword.split("\\s+");
        for (int i = 0; i < words.length; i++) {
            long end = start + words[i].length();
            helper.addSpan(layerId, start, end, words[i], 1);
            start = end + 1;
        }

        if ((right != null) && !right.isEmpty()) {
            words = right.split("\\s+");
            for (int i = 0; i < words.length; i++) {
                long end = start + words[i].length();
                helper.addSpan(layerId, start, end, words[i]);
                start = end + 1;
            }
        }
        helper.writeHitsDataView(writer, layerId);
        if (advancedFCS) {

            helper.writeAdvancedDataView(writer);
        }
        XMLStreamWriterHelper.writeEndResourceFragment(writer);
        XMLStreamWriterHelper.writeEndResource(writer);
    }
```

Configuration

Maven

To include FCSSimpleEndpoint these are the dependencies:

```
<dependencies>
    <dependency>
      <groupId>eu.clarin.sru.fcs</groupId>
      <artifactId>fcs-simple-endpoint</artifactId>
```

```
      <version>1.3.0</version>
    </dependency>
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>servlet-api</artifactId>
      <version>2.5</version>
      <type>jar</type>
      <scope>provided</scope>
    </dependency>
  </dependencies>
```

The version is currently 1.4-SNAPSHOT if you want and enable the Clarin snapshots repository.

Endpoint

To enable SRU 2.0 which is required for FCS 2.0 functionality you need to provide the following initialization parameters to the servlet context:

```
    <init-param>
        <param-name>eu.clarin.sru.server.sruSupportedVersionMax</param-name>
        <param-value>2.0</param-value>
    </init-param>
    <init-param>
        <param-name>eu.clarin.sru.server.legacyNamespaceMode</param-name>
        <param-value>loc</param-value>
    </init-param>
```

The endpoint configurations consists of the already mentionend context (ServletContext), a config (SRUServerConfig) and if you want further query parsers (SRUQueryParserRegistry.Builder). Also additional parameters gathered from servlet configuration and the servlet context are available.

EndPointDescriptionParser

You probably start out using the provided EndPointdescriptionParser. It will parse and make available what is required and also do some sanity checkning.

- Capabilities, basic search capability is required and advanced is available for FCS 2.0, checks that any given capability is encoded as a proper URI and that the IDs are unique.
- Supported Data views, checks that <SupportedDataView> elements have:
  - a proper @id attribute and that the value is unique.
  - a @delivery-policy attribute, eg DeliveryPolicy.SEND_BY_DEFAULT, DeliveryPolicy.NEED_TO_REQUEST.
  - a child text node with a MIME-type as its content, eg for basics search (hits): application/x-clarin-fcs-hits+xml and for advanced search: application/x-clarin-fcs-adv+xml

    Sample: <SupportedDataView id="adv" delivery-policy="send-by-default">application/x-clarin-fcs-adv+xml</SupportedDataView>

Makes sure capabilities and declared dataviews actually match otherwise it will warn you.

- Supported Layers, checks that <SupportedLayer> elements have:

    - a proper @id attribute and that the value is unique.
    - a proper @result-id attribute and that is is encoded as a proper URI, ant that the child text node is "text", "lemma", "pos", "orth", "norm", "phonetic", or other value starting with "x-".
    - if a @alt-value-info-uri attribute that is encoded as proper URI, eg tag description
    - if Advanced search is given in capabilities that it is also available.

- Resources, checks that some resources are actually defined, and have:
    - a proper @xml:lang attribute on its Description elelement.
    - a child LandingPageURI element
    - a child Language element and that is must use ISO-639-3 three letter language codes

Translation library

For the current version of the translation library a mapping for UD-17 to your used word classes for the word class layer is needed. It currently also does X-SAMPA conversion for the phonetic layer. The mappings are specified in one configuration file, an XML document. This will mostly be 1-to-1, but might require lossy translation either way. To guide you in this we walk through configuration and mapping examples from the reference implemetations.

Part-of-Speech (PoS)

The PoS translation configuration is expressed in a TranslationTable element with the attributes *@fromResourceLayer, @toResourceLayer* and *@translationType*:

...<TranslationTable fromResourceLayer="FCSAggregator/PoS" toResourceLayer="Korp/PoS" translationType="replaceWhole">...

@translationType is currently a closed set of two values, but could be extended by any definition on how to replace something in to. The values are *replaceWhole* and *replaceSegments*, but replaceSegments require further defintions of trellis segment translations which will not be addressed by this tutorial.

The values of *@fromResourceLayer* and *@toResourceLayer* only depends on these being declared by ResourceLayer elements under /AnnotationTranslation/Resources:

<ResourceLayer resource="FCSAggregator" layer="phonetic" formalism="X-SAMPA" />

The attributes of ResourceLayer are *@resource*, *@layer* and *@formalism*. The value of *@layer* is (most easily) the identifier which is used for the layer in the FCS 2.0 specification. @formalism is (most easily) the namespace value prefix or an URI. E g for PoS this can be SUC-PoS for the already mentionend SUC PoS tagset, CGN or UD-17. These tag sets often also includes morphosyntactic descriptions MSD in its original form, but since MSD is not part of the FCS 2.0 specification we are only dealing with the PoS tags here.

Going from UD-17's *VERB* tag to Stockholm Umeå Corpus (SUC) Part-of-Speech you get two tags *VB* and *PC*:

<Pair from="VERB" to="VB" />

<Pair from="VERB" to="PC" />

Adding the translation of the UD-17 *AUX* tag which gives *VB* in SUC-PoS too, but this is a 1-to-1 translation this way.

<Pair from="AUX" to="VB" />

As you can see from this the precision is varying and could become too bad to be useful going both ways from the FCSAggregator to the endpoint and then back. For this you can use the available alerting methods given in the FCS 2.0 specification.

With non-1-to-1 translations you need to know how alternatives are expressed in the endpoints query language. This is where the not yet available conversion library would use the translation library adding rule-based knowledge on how to translate to eg CQP [pos = “VB” | pos = “PC”]

Other resources

http://clarin.ids-mannheim.de/downloads/clarin/DigiBibSRU-source-2016-02-08.zip

https://clarin.ids-mannheim.de/digibibsru-new

Korp Endpoint