1. Introduction

Download the PDF version of this document on general information about SyMBA Manual.

1. Purpose of SyMBA

Systems and other integrative approaches to biology generate a wide variety of genome-scale and high-throughput data that must be annotated and reliably stored. Novel data repositories are required that not only allow the storage of multiple datasets of differing types, but also capture the metadata required to record the experimental details and provenance in a standard format that is amenable to data sharing. The Functional Genomics Experiment Mark-Up Language (FuGE-ML) and associated Object Model (FuGE-OM) were created to facilitate the development and uptake of metadata standards. FuGE developers have also created a number of base Software ToolKits (STKs) that are good starting-points for further development of FuGE-based tools. Systems and Molecular Biology Data and Metadata Archive (SyMBA) is based on the FuGE Version 1 Hibernate STK. SyMBA is based on the FuGE-OM and archives, stores, and retrieves raw high-throughput data. SyMBA integrates both multiple 'omics' data types and information about experiments in a single database.

SyMBA is a flexible data repository specifically designed to address the requirements for primary data storage for projects requiring the storage of many different data types. The metadata database stores all information using the FuGE-OM, and as such can integrate all metadata using the same structure, irrespective of associated data type. It is one of the first database implementations of Version 1 of the FuGE-OM. SyMBA features a metadata database, general and expert user interfaces, and utilizes a Life Science Identifier resolution and assigning service to uniquely identify objects and provide programmatic access to the database.

FuGE, an emerging data standard, provides fast development time in the short term, and an export format that is easy to extend and share in the longer term. Our implementation of FuGE also acts as an example for others wishing to apply FuGE for their own purposes. We encourage other groups with similar needs to install, evaluate, and contribute to its development. The most up-to-date version of this documentation can be found in the SyMBA SourceForge Subversion repository. The system - and its documentation - is available for download and installation via the SourceForge project site ( ).

2. Relation to the FuGE Toolkit Projects

Some sections of the documentation has its origins in the FuGE Hibernate STK documentation , which in turn was based on a number of sources, including the Milestone 3 documentation of SyMBA itself.

SyMBA utilizes Version 1.0 of the FuGE Standard. It provides a database and persistence layer based on the FuGE Object Model (FuGE-OM), together with other helper classes. These are ideal for community developers wishing to store their FuGE-related data in a more structured way than XML files provide, and as a foundation to build FuGE tools or a FuGE-based system.

SyMBA is built with Apache Maven 2, a software project management and build system similar to, but more comprehensive than other build tools such as Apache Ant or GNU Make. It also utilizes AndroMDA, a model-driven architecture (MDA) and code generation system, to build a variety of Java classes, configuration files, documents and database scripts. AndroMDA plugs into Maven and during the build process parses a developer's UML Object Model (OM) and generates the appropriate output. It is the core tool used within all FuGE toolkits, and is highly configurable. The type of output can be tailored using different AndroMDA conversion tools, called 'cartridges'. These encapsulated cartridges each do a separate task: for instance, there is an XML Schema cartridge that builds XSDs from UML, and there are Hibernate and Spring or EJB3 cartridges that build a database and persistence layer based on the provided UML.

SyMBA provides:

- a FuGE-structured relational database
- a object-relational persistence and query layer
- a set of Plain Old Java Object (POJO) entity classes representing FuGE UML entities
- a set of Data Access Object (DAO) POJO classes that facilitate and encapsulate access to entity classes
- unit tests of all queries and version retrieval
- a user-friendly web interface that interacts with the FuGE database and the remote file store for storing raw data outputted from experimental assays

A persistence layer bridges the gap between a relational database and POJOs, abstracting the low-level database code by providing the programmer with an object-oriented interface. Using AndroMDA-generated classes in conjunction with Hibernate + Spring allows the developer to work only with POJOs, and, when ready to populate, query or modify the database, the persistence layer handles the underlying database commands.

SyMBA makes use of the FuGE XSD STK project and the FuGE Hibernate STK project but builds greatly upon these two. Changes to the FuGE-OM (mainly extra querying and the versioning expansion) means that both of these STKs need to be part of SyMBA, and re-built when SyMBA is built. A Maven 2 repository specifically holding these jars is available, can be accessed by adding the following repository to your own pom (if you wish to use SyMBA jars, but not SyMBA itself).

```
<repository>
  <id>carmen-snapshot</id>
  <name>Carmen snapshot repository</name>
  <url>http://carmen.ncl.ac.uk/maven/repo-snapshot</url>
</repository>
<repository>
  <id>carmen</id>
  <name>Carmen release repository</name>
  <url>http://carmen.ncl.ac.uk/maven/repo</url>
</repository>
```

Please note that these jars will not have any real database connection details, so cannot be used to connect to an in-house database, but are useful if you just want to make use of, or view, some of the helper classes.

When you check-out the symba subversion repository, you will get a Maven2 project. Full instructions for compiling this project and generating and accessing the FuGE database are included in this documentation.

2. Developing SyMBA Templates

1. Introduction

Please note that this section of the Documentation is NOT complete, and is a work in progress.

2. Describe Your Inputs

2.1. Describe Your Material

Inputs to the assays that create your data files are generally FuGE Materials.

- Specimen / Organism : FuGE = Material (Input)

2.2. Describe Your Software

2.3. Describe Your Equipment

3. Describe Your Protocols

4. Connect Software, Equipment, and Material to Your Protocols

4.1. Adding Materials

Materials are only connected to the run of the experiment when the GenericProtocolApplications are made. Therefore connecting your material to the appropriate Assay Protocol will happen using the "name" attribute of your Dummy Material. No additional steps are required here.

5. Describing Your Investigation

5.1. Investigation Components

5.1.1. Links to Protocol Applications

5.1.2. Creating Factors and Factor Values

6. Reserve Words in SyMBA Templates

In SyMBA templates, there are many reserved words to allow the web interface to interpret your intentions correctly. To ensure that they do not conflict with any real words you might wish to use, they are all prefixed with net.sourceforge.symba.keywords. . Below is a list of reserve words that may be used in the template, and a short definition of their meaning. In the following sections, which of these reserved words are allowed in the different sections are defined.

- net.sourceforge.symba.keywords.noDatafile : used to let the web interface know that this application of a protocol has inputs and/or outputs that do not contain data files (only materials).
- net.sourceforge.symba.keywords.dummy : used to show that the XML element in the template isn't itself the one to copy into the users's experiment, but is instead a dummy object that is meant to be copied into a real object before saving.

7. Create Your Input XML

7.1. Describe Your Material

Inputs to the assays that create your data files are generally FuGE Materials.

7.1.1. Name Your Material

You should name your material using a number of reserve words to modify the behaviour of the dummy Material in the web interface. Any Material you put in your SyMBA template will remain untouched by users: instead, a copy is made of the Material, and the appropriate user-defined parameters are added before it gets associated with a particular user's experiment.

7.1.1.1. Reserve Words for Material

Below is an example name attribute for a GenericMaterial in a SyMBA Template. A full explanation of how this name was built follows.

```
Material Characteristics net.sourceforge.symba.keywords.dummy Noname Notreatment for
Recording Protocol (Component of Carmen Electrophysiology Investigation)
```

- "Material Characteristics" : Anything before the "Dummy" Keyword will be used to title the fieldset that the material metadata is grouped into in the web interface.
- "net.sourceforge.symba.keywords.dummy" : This keyword MUST be used in the Material name, as it tells SyMBA that this is not a real Material. It is also a keyword that is used in the Search methods within SyMBA so that Dummy Materials won't be shown in search results.
- "Noname" : This keyword ensures that the user isn't asked for a general name for their Material. This value would otherwise go into the "name" attribute of the user's GenericMaterial element. If you want to control the name of the Material (e.g. through an ontology term), you should use this keyword.
- "Notreatment" : This keyword will disable SyMBA's default display of a text box that would allow users to enter free text about the treatments of their Material. You may wish to disable this if you have specific parameters describing your treatments.
- "for Recording Protocol (Component of Carmen Electrophysiology Investigation)" : This section is REQUIRED, and must contain the full name, including the "(Component of...") section, of the Protocol that this Material will be associated with as an input material.

7.2. Describe the Parameters of the Protocol Application (PA)

7.2.1. Name Your PA

You should name your material using a number of reserve words to modify the behaviour of the dummy Material in the web interface. Any Material you put in your SyMBA template will remain untouched by users: instead, a copy is made of the Material, and the appropriate user-defined parameters are added before it gets associated with a particular user's experiment.

7.2.1.1. Reserve Words for PA

Below is an example name attribute for a GenericProtocolApplication in a SyMBA Template. A full explanation of how this name was built follows.

```
Parameters Associated with Creating the Data File Dummy for Real-Time PCR Protocol
(Component of MiMage Investigation)
```

- "Parameters Associated with Creating the Data File" : Not used by the web interface, but useful for the template developer as a comment.
- "Dummy" : This keyword MUST be used in the PA name, as it tells SyMBA that this is not a real Material. It is also a keyword that is used in the Search methods within SyMBA so that Dummy Materials won't be shown in search results.
- "for Real-Time PCR Protocol (Component of MiMage Investigation)" : This section is REQUIRED, and must contain the word "for", the full name, including the "(Component of...") section, of the Protocol that this PA will be associated with.

3. Using SyMBA webservices

1. Introduction to SyMBA Web Services

Web services (WS) are used to make simple applications available as web applications (more information can be found here and here). SyMBA makes use of Crossfire (CXF) in a Maven 2 plugin to build web services into a war that can be loaded into a Tomcat server. CXF is "an open source services framework [...] that helps you build and develop services using frontend programming APIs, like JAX-WS. These services can speak a variety of protocols such as SOAP, XML/HTTP, RESTful HTTP, or CORBA and work over a variety of transports such as HTTP, JMS or JBI."

2. Services Currently Available

Currently, SyMBA has 3 services with a total of 4 methods. All of these services relate to the LSIDs contained within SyMBA, and via these LSIDs perform a number of useful tasks.

- LsidAssigner
    - assignLsid(). Create a new LSID.
- LsidResolver
    - getAvailableServices(). Check that the available web services can deal with your LSID.
- LsidDataRetriever
    - getMetadata(). Get some information about an LSID.
    - getData(). Get the data pointed to from an LSID.

3. Deviations From the LSID Specification.

The SyMBA LSID web services mentioned currently implement only those parts of the specification in active use within SyMBA (LSID specification). In particular, these web services (as well as the general SyMBA API) implement LSIDs as Java Strings rather than as complex objects. SyMBA developers have found no need to use the parameters and properties portion of the specification, and wish to keep the SyMBA implementation as straightforward to use and understand as possible. Additionally, only those methods from the LSID specification that SyMBA developers have found a use for have been implemented at this stage. If you wish a more complete implementation, please email the symba-devel@lists.sourceforge.net mailing list and let us know what you need.

4. LSID Metadata Compared With FuGE-OM Metadata

There are differences between what the LSID specification describes as "metadata", and what the FuGE-OM means by the same term.

LSID metadata is information about the LSID , as opposed to LSID Data, which is the information associated with the LSID. LSID metadata tells you things about the status of the LSID: it includes a expiration timestamp (which, if null, means the LSID does not expire), and any other information that the supplier of the LSID wishes to provide. The LSID metadata, unlike the LSID data, can change at any time, and is not required to always return the same thing. In SyMBA, the timestamp will always be null, and the metadata currently returns an RDF-formatted object with a title and a value. At the moment, this object is just the LSID of the latest version of Identifiable object associated with the same Endurant object as the LSID you provided.

FuGE metadata is every bit of information stored in the FuGE-OM structure, whereas the data is the actual data file(s) associated with this metadata. Each Identifiable object in FuGE (which includes the reference to the data file(s)) is marked with an LSID. Therefore, to retrieve FuGE metadata concerning a given LSID, call the getData() method in the LSID Data Retrieval WS. To retrieve the data file itself, you also call the getData() method, but provide the LSID that points to the raw data rather than the LSID that points to one of the structures that forms the FuGE-OM.

5. Structure of the SyMBA Web Services

There are four SyMBA modules dealing with WS. The separation of a relatively small number of classes into a relatively large number of Maven modules allows for greater flexibility and simpler installation and use of the WS.

- symba-lsid-ws-api. The API provides the Java interfaces that must be implemented if you wish to create your own version of the SyMBA WS. It is these classes that are annotated with the JAXB and CXF Java annotations, and it is from these classes that the WS files are auto-generated by CXF where necessary. Within this module are interfaces for LSID assigning (LsidAssigner.java), LSID service checking (LsidResolver.java), and Lsid Data and Metadata Retrieval (LsidDataRetriever.java). There are two other non-interface classes that provide the response / return structures for LSID metadata (LsidMetadataResponse.java), and for returning available WS for a given LSID (LsidDataServiceResponse.java).
- symba-lsid-ws-assigner. The assigner module is an implementation of the LsidAssigner interface. Its main method, assignLsid(), creates an LSID with a pre-set authority identification of "cisban.cisbs.org", and a namespace you specify. In SyMBA this namespace must be the class name of the object you are referencing.
- symba-lsid-ws-api-resolver. The resolver module is an implementation of the LsidDataRetriever and the LsidResolver interfaces.

- symba-lsid-ws-api-client. The client module provides example Java classes and Taverna workflows that can be used to access the two WS.

6. Accessing the SyMBA Sandbox Web Services

6.1. LsidAssigner

- assignLSID(). The LsidAssigner webservice contains a single method, assignLSID(), which returns an auto-generated LSID based on a namespace you specify. There is a sandbox installation of this webservice running on http://bsu.ncl.ac.uk:8081/symba-lsid-ws-assigner/services/LsidAssigner?wsdl . There are two examples of how to use this webservice.
    - Via a Java client class. You can examine and use the net.sourceforge.symba.lsid.webservices.client.LsidAssignerClient to connect to any running installation of the symba-lsid-ws-assigner module of the SyMBA project. You specify which WS you wish to connect to by modifying the address property of the clientAssignerFactory bean. The value of the address property, as it is checked-out from SyMBA subversion, is the value required to connect to the SyMBA sandbox WS installation.
    - Via Taverna. Within the symba-lsid-ws-client/src/main/resource/taverna-workflows directory, you'll find some example Taverna workflows that connect to the WS running on the SyMBA sandbox server. Load these files up in Taverna, and you'll be able to connect the SyMBA WS to your own Taverna workflows, or even run them on their own. There are also sample workflow inputs, those these may be out of date, as they give sample LSIDs that were in the sandbox at the time of creating the input files, but which are not necessarily always in the sandbox.

6.2. LsidResolver

- getAvailableServices(). Check that the available web services can deal with your LSID. This method only sends you the location of the client-beans.xml file appropriate to the current service setup, which you can get from the SyMBA checkout anyway. In other words, you shouldn't really need this method, and it's just implemented to get closer to the LSID specification. However, if you wish to use it, there are two ways, as described for the assigning WS, i.e.
    - Via a Java client class. You can examine and use the net.sourceforge.symba.lsid.webservices.client.LsidResolveAndRetrieveClient to connect to any running installation of the symba-lsid-ws-resolver module of the SyMBA project. You specify which WS you wish to connect to by modifying the address property of the clientAssignerFactory bean. The value of the address property, as it is checked-out from

    SyMBA subversion, is the value required to connect to the SyMBA sandbox WS installation.
    - Via Taverna. Within the symba-lsid-ws-client/src/main/resource/taverna-workflows directory, you'll find some example Taverna workflows that connect to the WS running on the SyMBA sandbox server. Load these files up in Taverna, and you'll be able to connect the SyMBA WS to your own Taverna workflows, or even run them on their own. There are also sample workflow inputs, those these may be out of date, as they give sample LSIDs that were in the sandbox at the time of creating the input files, but which are not necessarily always in the sandbox.

6.3. LsidDataRetriever

- getMetadata(). Get some information about an LSID. This method is used to return documents containing the metadata associated with a particular lsid at this particular data retrieval service. According to the LSID specification, metadata can be returned in multiple formats. The acceptedFormats argument is an array of strings, each of which contains a media type. SyMBA only accepts LsidMetadataResponse.LSID_RDF_FORMAT, which equals "application/rdf+xml". You can access this method in the same way as the other WS described, i.e.
    - Via a Java client class. You can examine and use the net.sourceforge.symba.lsid.webservices.client.LsidResolveAndRetrieveClient to connect to any running installation of the symba-lsid-ws-resolver module of the SyMBA project. You specify which WS you wish to connect to by modifying the address property of the clientAssignerFactory bean. The value of the address property, as it is checked-out from SyMBA subversion, is the value required to connect to the SyMBA sandbox WS installation.
    - Via Taverna. Within the symba-lsid-ws-client/src/main/resource/taverna-workflows directory, you'll find some example Taverna workflows that connect to the WS running on the SyMBA sandbox server. Load these files up in Taverna, and you'll be able to connect the SyMBA WS to your own Taverna workflows, or even run them on their own. There are also sample workflow inputs, those these may be out of date, as they give sample LSIDs that were in the sandbox at the time of creating the input files, but which are not necessarily always in the sandbox.
- getData(). Get the data pointed to from an LSID. The LSID specification states that "This method is used to return data associated with the given lsid. If a copy of the data represented by an LSID cannot be returned for any reason, an exception should be raised. If the given lsid represents an abstract entity (a concept), this method returns an empty array of bytes. Note that the semantics of the returned bytes is not defined by this specification." You can access this method in the same way as the other WS described, i.e.
    - Via a Java client class. You can examine and use the net.sourceforge.symba.lsid.webservices.client.LsidResolveAndRetrieveClient to connect to any running installation of the symba-lsid-ws-resolver module of the SyMBA project. You specify which WS you wish to connect to by modifying the address property of the

clientAssignerFactory bean. The value of the address property, as it is checked-out from SyMBA subversion, is the value required to connect to the SyMBA sandbox WS installation.

- Via Taverna. Within the symba-lsid-ws-client/src/main/resource/taverna-workflows directory, you'll find some example Taverna workflows that connect to the WS running on the SyMBA sandbox server. Load these files up in Taverna, and you'll be able to connect the SyMBA WS to your own Taverna workflows, or even run them on their own. There are also sample workflow inputs, those these may be out of date, as they give sample LSIDs that were in the sandbox at the time of creating the input files, but which are not necessarily always in the sandbox.

7. Modifying and Running Your Own SyMBA Web Services

If you wish to use the SyMBA LSID web service API, but need a different implementation of the Assigner and/or the Resolver/Retriever, you can easily accomplish this via your own implementation of the three interfaces present within the symba-lsid-ws-api module. Create your own implementations for whichever class needs to have behaviour different from the provided implementations. You can do this by

- Copy the SyMBA module you wish to make your own implementation of (e.g. the symba-lsid-ws-assigner module), giving a different name to the new directory.
- Change the name of the module in the copied pom.xml.
- Change the LsidAssignerImpl code to suit your needs, ensuring that you also change the name of the class and/or the package it is contained in to avoid any possibility of a clash.
- Build the war with "mvn install" and then access your new WS with your own version of the client code, as exemplified in the symba-lsid-ws-client module.
- If you wish this module to be built as part of the parent's "mvn install" command from the trunk/ directory, then you will need to add the module name to the trunk/pom.xml, in the module section.