1. Introduction

Download the PDF version of this Installation Manual by clicking here.

1. Background

The FuGE XSD STK is a lightweight Maven project that uses AndroMDA to read in the UML diagram, or diagrams, and produce an XSD from the UML. AndroMDA has limited XML support, and cannot yet generate FuGE-ML that can be validated against the XSD. Therefore the STK also includes a JAXB2 sub-project that will automatically build XML-binding Java classes based on the auto-generated XSD. The FuGE XSD and FuGE-JAXB2 jar built by this STK will be available on the FuGE Maven2 Repository for both other STKs and external developers to access. Syntactic validation of XML documents based on the generated XSD can be performed by utilizing the JAXB2 code that is created within the project. Example Java classes are also provided to show developers how to input, output, validate, manipulate, and generate validated random output.

2. Purpose of the XSD STK and its relation to the greater FuGE Toolkit Project

The FuGE software tools are separated based on their development goals. The FuGE XSD Project has been specifically created to aid

- those who wish to be able re-create or modify the FuGE XSD for their own purposes
- developers of community extension XSDs

When you check-out the xsd-stk subversion repository, you will get a Maven2 project. As provided, this project re-creates the FuGE xsd only. However, with a couple of simple modifications a custom UML model can be created, and a community-specific XSD based on the FuGE model can be made. Full instructions for compiling this project and generating the base FuGE XSD, as well as for creating community extensions of FuGE are included in this document in the Installation chapter.

3. What is the relationship between the FuGE XSD and the FuGE-OM?

In the FuGE development process, the master version of the FuGE structure is the Object Model, written with UML and, using Maven 2 and AndroMDA, translated into an XSD.

4. First Steps

If you're interested in FuGE, you can download the FuGE-OM and view it in MagicDraw 15.0 . The Community Edition can be downloaded for free, if you are an academic.

If you wish to develop a community extension based on FuGE, then this XSD Software ToolKit (STK) is right for you. Please continue on to the next chapter on installing this STK and creating a community extension.

2. Installation

1. Checking Out the FuGE XSD STK from Subversion

Access the subversion repository via directions present in Source Repository Documentation .

2. Installation and Generation of the FuGE XSD

What follows are the installation instructions for this STK. If followed correctly, you will have successfully generated the plain FuGE XSD, without any extra modifications or community extensions. If successful, you can then move on to creating your own community extension.

This FuGE toolkit can be installed on either Windows or Linux, and it may work on Mac systems, though it hasn't been tested.

These instructions are based on http://wiki.ficcs.org/ficcs/FuGE-to-XSD and the Andromda Forums , with further additions and modifications to suit this toolkit.

2.1. Download and Installation of Required Software

Please read through this section and ensure that you either install the missing software, or double-check that you already have the correct version of the software installed.

2.1.1. Java

- Install J2SE Development Kit 5.0 (JDK 5.0) or higher from http://java.sun.com/j2se/1.5.0/download.jsp or similar.
- Make sure that the JAVA_HOME environment variable is pointing to the directory where you installed the JDK.

2.1.2. Maven2

- Download and install Maven 2.0.7 or later from http://maven.apache.org/download.html

- Maven2 Setup. Create a directory in your home directory called .m2 with a single file inside called settings.xml . This is what one possible settings.xml looks like, where the local maven repository is in a different directory from the settings.xml file, and where there needs to be a proxy set up to connect to the outside world:

```
     <settings>
<localRepository>/media/share/synched/Documents/.m2/repository/</localRepository>

       <proxies>
         <proxy>
           <active>true</active>
           <protocol>http</protocol>
           <host><a href="http://wwwcache.ncl.ac.uk/"
target="_blank">my.proxy.host</a></host>
           <port>8080</port>
         </proxy>
       </proxies>
     </settings>
```

  All sections are optional. However, depending on your circumstances, you may wish to use one or more of these settings. The "localRepository" element names a location separate from the default home directory for the Maven2 repository. This may be beneficial if you have limited space on your home directory, as this repository directory can grow quite large. The "proxies" element should only be used for those developers who must access the internet via a proxy.
- Environment Variables and Settings for Maven . Set up the environment variable M2_HOME to point to your maven installation directory, and then ensure that both $M2_HOME/bin and $JAVA_HOME/bin are present in your PATH. It is also useful, but not required, to specify $M2_REPO, which is the location of your Maven2 repository.
- Test Maven2 (Part One). You'll know if you've gotten this right if, when you run mvn --version, you something similar (but not necessarily identical) to the following:

```
$ mvn --version
Maven version: 2.0.7
Java version: 1.5.0_08
OS name: "linux" version: "2.6.17-12-386" arch: "i386"
```

- Test Maven2 (Part Two). Check that Maven2 is working properly by creating a temporary, empty project with the following command:

```
mvn archetype:create -DgroupId=testapp -DartifactId=testapp
```

  Check for the BUILD SUCCESSFUL message and, once you have received this message, please delete the created testapp folder.

2.1.3. AndroMDA

This is the only AndroMDA artifact that we will install explicitly. All other artifacts, such as AndroMDA cartridges, will be automatically downloaded by the Maven2 scripts generated by the plugin. Install the plugin by following the steps below.

- Go to http://team.andromda.org/maven2/org/andromda/maven/plugins/andromdapp-maven-plugin/3.2/andromdapp-maven-plugin-install-3.2.zip to download the the AndroMDA plugin installer.
- Unzip the contents of the installer into your Maven repository at $M2_REPO (or whatever you have set "localRepository" to be, or by default, it resides in your-home-dir/.m2/repository).
- Verify that the following directory was created (switch the slashes around for Windows): $M2_REPO/org/andromda/maven/plugins/andromdapp-maven-plugin
- Create a temporary directory, e.g. C:\andromda-temp or $HOME/andromda-temp .
- Create a file called pom.xml in this directory with the following content:

```
<project>
<modelVersion>4.0.0</modelVersion>
<groupId>samples.test</groupId>
<artifactId>test</artifactId>
<version>1.0</version>
<packaging>jar</packaging>
<name>test</name>
<build>
<defaultGoal>compile</defaultGoal>
<plugins>
<plugin>
<groupId>org.andromda.maven.plugins</groupId>
<artifactId>andromdapp-maven-plugin</artifactId>
<version>3.2</version>
</plugin>
</plugins>
</build>
<repositories>
<repository>
<id>andromda</id>
<name>AndroMDA Repository</name>
<url>http://team.andromda.org/maven2</url>
</repository>
</repositories>
<pluginRepositories>
<pluginRepository>
```

```
<id>andromda</id>
<name>AndroMDA Repository</name>
<url>http://team.andromda.org/maven2</url>
</pluginRepository>
</pluginRepositories></project>
```

- Open a Command Prompt in the directory where you created this pom.xml and run the command mvn without any arguments. Make sure the command completes successfully by displaying the BUILD SUCCESSFUL message.
- You can now delete the temporary directory you just created.

2.2. Checkout from Subversion

1. Instructions for Subversion checkout are available on the Source Repository help page.

2.3. Compilation and Creation of the XSD

2.3.1. Extra Step for the First Time You Compile

- Change into the top-level trunk/ directory, and run this command:

```
mvn install
```

- Though this is exactly the command you will use later on to generate the XSD and the JAXB2 Java code, this needs to be run once to download the required AndroMDA profiles. These profiles are *.xml.zip files that are neccessary for correctly using MagicDraw. Before you can fix the bug in the AndroMDA plugin mentioned in the next section, you need to download the AndroMDA plugin itself. This first mvn install command will do that for you.

2.3.2. Fixing a Bug in the AndroMDA plugin

When you downloaded and installed the AndroMDA plugin for Maven 2, there was a small bug in one of the jars. (Details of this bug can be found on the AndroMDA forum , and thanks to Olga Tchuvatkina for including the instructions for fixing it.)

Without this fix, the XSDs generated will be virtually empty, containing only the top-level namespace information.

1. Go into the top-level directory (directly underneath the trunk/ directory) of the XSD STK you've just downloaded. You'll find a file called andromda-xmlschema-cartridge-3.2.jar there.
2. Copy this into the correct part of your Maven 2 Repository. Below the environment variable $M2_REPO is used to mark the location of your local Maven 2 Repository. Overwrite the default jar file with the fixed one:

```
    cp andromda-xmlschema-cartridge-3.2.jar
$M2_REPO/org/andromda/cartridges/andromda-xmlschema-cartridge/3.2/
```

2.3.3. Generate all of the automatically-generated AndroMDA sources

Change into the top-level trunk/ directory, and run this command:

```
mvn install
```

You should see a "BUILD SUCCESSFUL" message at the end of it. You must be connected to the internet for this step to work the first time you run it, as there will be many jars that need to be downloaded. Further AndroMDA-specific maven commands are available in the trunk/readme.txt generated by AndroMDA.

- You now have all auto-generated code. This includes
    - the XSD, found in fuge-jaxb2/src/main/resources/xmlSchema.xsd . The XSD needs to be directly written into the fuge-jaxb2 directory so that the JAXB2 plugin can access it.
    - the JAXB2 code that fits with the XSD, present both as Java classes fuge-jaxb2/target and as a jar copied into your local maven repository.

2.4. Preparing Your UML Tool

You will need a UML tool. AndroMDA has been tested with both ArgoUML 0.20 and MagicDraw 15.0 . However, MagicDraw 15.0 is the recommend UML Editor for the FuGE toolkits. You can download the Community Edition for free, if you are an academic.

- Open up MagicDraw 15.0. Before you open any of the FuGE mdzip files, you need to set the maven2.repository path variable. This is equivalent to the $M2_REPO environment variable you will have already set when installing maven. It points to the location of the andromda 3.2 profiles, which are inside the maven repository. This value should be wherever you have your maven repository: if you installed Maven2 using the default repository location, then maven2.repository would be $HOME/.m2/repository for linux, or the equivalent in windows.
- If you are having problems, you can manually copy all of the andromda profiles from the maven repository ($M2_REPO/org/andromda/profiles) into the same directory as the model file, or into the profiles/ directory of the MagicDraw installation location.
- Test that your setup works after setting this variable by opening trunk/src/mda/src/main/uml/FuGE-v1-profile.mdzip . If it loads without error, you MagicDraw installation is complete.

2.5. Integration of SyMBA with your favourite IDE

You can auto-generate Java IDE project files using maven2. For example, to autogenerate IntelliJ files (please do this after successfully running mvn install) change directory to the top-level directory and run the following command:

```
            mvn idea:idea
```

There are other similar commands for IDEs such as Eclipse. Details can be found on the Maven website.

3. Extending the FuGE XSD

1. Extending the FuGE-OM

1.1. To Get You Started

You will need to look in your mda/src/main/uml for the MagicDraw 15.0 zip file (.mdzip) called NewFuGEExtension.mdzip . This is an empty template UML diagram that imports the FuGE-OM. New modellers should start with this UML version.

1. Rename the file to match the name of your community extension.
2. Open the file in MagicDraw 15 and start editing. You build your extension on FuGE by altering the NewFuGEExtension package. You will want to rename the package name, and then put all of your extensions in this package.

1.2. Full Documentation

Extensive directions on extending the UML are available in the documents on the FuGE Developers'website. This includes the full Specification as well as the Reference Manual . Further, there is a FuGE paper in Nature Biotechnology .

Please continue to the next section of the chapter to see how to convert your modified UML into an XSD.

2. Update the FuGE-OM from previous version

2.1. Steps

If you have already built an extension based on previous FuGE-OM, and you want to upgrade it to latest FuGE-OM, follow the below steps.

1. download all of the andromda profiles from the andromda's maven2 site, put them all in one directory called andromda-profiles.
2. create new MagicDraw project named it like "magev2"

3. select MagicDraw menu File->use module, choose andromda-profile-3.2-.xml.zip in the "andromda-profiles" directory. MagicDraw will then ask for the androMDA's xml, webservice, service, process profile files, find them in that "andromda-profiles" directory.
4. select MagicDraw menu File->use module, choose FuGE-v1-profile.mdzip
5. select MagicDraw menu File->import MagicDraw project, choose your extension's MagicDraw project file which uses previous version of FuGE-OM, the MagicDraw will load it, automatically figure out the dependence on latest FuGE profile, update the project root name and import the MAGE package.

3. Creating Your Community XSD

You've created a plain vanilla FuGE XSD. You've also extended the FuGE UML and made your own community extension. What's next? The final step is to translate your new UML into an XSD.

These instructions are based on http://wiki.ficcs.org/ficcs/FuGE-to-XSD, with further additions and modifications to suit this toolkit.

The default setup as provided when you checked out this project from Subversion is to generate the FuGE XSD only. There are a couple of changes to both the Maven properties file and the Andromda properties file

3.1. Change Maven 2 Setup

- If you have changed the name of the NewFuGEExtension.mdzip, then please update the value in trunk/mda/pom.xml to match your file name:

```
<extension.model.uri>file:${project.build.sourceDirectory}/NewFuGEExtension.uml2</exten
sion.model.uri>
```

3.2. Change AndroMDA Setup

The default setup as provided when you checked out this project from Subversion is to generate the FuGE XSD only. There are a few small changes to the AndroMDA configuration files that you'll need to make to ensure that your new community extension is recognized and processed by AndroMDA.

1. Specify a joined or separate XSD. By default, the XSD generated will contain all of the FuGE XSD as well as your community XSD. If you do not wish this to occur, you will need to tell AndroMDA this. You do this by going into the mda/src/cartridge/custom/templates/xmlschema directory and selecting a different vsl file than the one provided by default. First, rename the original XmlSchema.vsl file so you can go back to using it if you ever need to:

```
mv XmlSchema.vsl XmlSchema-shared.vsl
```

Then, rename XmlSchema-for-extensions.vsl to XmlSchema.vsl :

```
mv XmlSchema-for-extensions.vsl XmlSchema.vsl
```

2. Follow the instructions in the new XmlSchema.vsl. Open up your new XmlSchema.vsl in a text editor and read the instructions at the top of the file. Make the changes within the vsl file as instructed.
3. Modify the AndroMDA configuration file , mda/src/main/config/andromda.xml . This tells AndroMDA what bits of the XSDs to create. We will be modifying this part of the configuration file:

```
<modelPackages processAll="true">
   <modelPackage process="true">FuGE::**</modelPackage>
</modelPackages>
```

If you decided in the previous step to generate a joined XSD with both the FuGE and your community's schema, then you just have to add your own package name as a modelPackage element:

```
<modelPackages processAll="true">
   <modelPackage process="true">FuGE::**</modelPackage>
   <modelPackage process="true">your.package.name::**</modelPackage>
</modelPackages>
```

If you decided in the previous step to just generate your community's schema, then you need to set processAll as well as the processing of the FuGE package to false :

```
<modelPackages processAll="false">
   <modelPackage process="false">FuGE::**</modelPackage>
   <modelPackage process="true">your.package.name::**</modelPackage>
</modelPackages>
```

For general interest, you can flip which package is generated at any time by changing the process attribute of the modelPackage element and re-generating the XSD.

- Un-comment the following line from the andromda.xml file you were modifying. It is just above the modelPackages element:

```
<!--<uri>${extension.model.uri}</uri>-->
```

3.3. Export the UML in XMI/UML2 format

FuGE, and the extension you're working on, is stored in Subversion as a MagicDraw 15.0 zip file. In order for AndroMDA to be able to process it, you need to convert it into a more standard file format.

- Open up your mdzip file in MagicDraw 15.0. Once opened, perform the following export:

```
File -> Export -> EMF UML2 (v1.x) XMI
Ensure you select the trunk/mda/src/main/uml directory as the export directory
Click "Export"
```

  This conversion will produce a number of files, all ending with the extension .uml2 . If you make any changes to the FuGE-OM, you will need to re-export all of the files again by following the directions above.

3.4. Re-run the Maven 2 build process

Change into the top-level trunk/ directory, and run this command:

```
mvn install
```

You should see a "BUILD SUCCESSFUL" message at the end of it.

4. Example Java Code Provided in the STK

1. Using RandomXmlGenerator

You will find a main() method in the GenerateRandomXml that tells you how to access the RandomXmlGenerator class. Currently there is only one public method, generate() . Use this to generate a FuGE-ML file compliant with the FuGE XSD generated by AndroMDA. There are example command-line statments within the fuge-jaxb2/ sub-project inside readme.txt that will help you get started using these classes. For example, you can use Maven2 to call a Java class: the benefit of this is that you will automatically make use of the Maven2 classpaths, and will not have to set something up yourself. An example command to run GenerateRandomXml is as follows:

```
    cd fuge-jabx2/
    mvn exec:java -Dexec.mainClass="net.sourceforge.fuge.util.GenerateRandomXml" -
Dexec.args="src/main/resources/xmlSchema.xsd output.xml"
```

Please note that RandomXmlGenerator is not guaranteed to create a FuGE-ML document that uses every single possible element in the FuGE XSD. However, we would certainly like to be complete, so if you spot something that needs adding, please feel free to either contribute to the code yourself, or send a mail to fuge-devel@lists.sourceforge.net with your request.

If you modify this STK to generate a community extension, you will need to also modify this code to handle that extension.

2. Using XmlRoundTrip

You will find a main() method in the XmlRoundTrip that tells you how to input, validate, modify, and output valid FuGE-ML file compliant with the FuGE XSD generated by AndroMDA. (Further code that you can examine to see how to process the JAXB2-generated code can be found in RandomXmlGenerator >.) There are example command-line statments within the fuge-jaxb2/ sub-project inside readme.txt that will help you get started using this class. For example, you can use Maven2 to call a Java class: the benefit of this is that you will automatically make use of the Maven2 classpaths, and will not have to set something up yourself. An example command to run XmlRoundTrip is as follows:

```
    cd fuge-jabx2/
    mvn exec:java -Dexec.mainClass="net.sourceforge.fuge.util.XmlRoundTrip" -
```

```
Dexec.args="src/main/resources/xmlSchema.xsd output0.xml roundtrip.xml"
```

If you spot something that needs adding or could be improved, please feel free to either contribute to the code yourself, or send a mail to fuge-devel@lists.sourceforge.net with your request.

If you modify this STK to generate a community extension, you will need to also modify this code to handle that extension.

5. Would You Like to Contribute?

1. Do you wish to contribute to the FuGE XSD STK?

If you have enjoyed using this STK, but also have some ideas for improvement, or perhaps wish to include some of your own code in this project, please begin by contacting fuge-devel@lists.sourceforge.net. Explain what things you would like to see included, and whether or not you would like to help add them.

If appropriate, the FuGE Administrators could add you as a developer, allowing you to commit code back to the FuGE subversion repository. There are some guidelines to follow, however, and please read on to see them.

2. Introduction

This section is for FuGE developers only: that is, those who are identified on the SourceForge project site as developers or administrators. It offers guidelines in modifying and documenting this STK. You will only be able to modify the website or the code in Subversion if you are a FuGE developer.

3. What to include in new Java Classes

If you're contributing new Java classes to the STK, please ensure that you have put in useful javadoc for all methods. If you've created a new package or sub-project, you'll also need to check the top-level pom.xml in the reporting element, and ensure that your source code will get picked up by the javadoc plugin when the mvn site:site command is run.

Also, please copy the license section from one of the existing Java classes and include it in your file.

4. Subversion Best-Practices

You can commit changes if one of the FuGE SourceForge administrators adds you as a FuGE developer. You will need to contact fuge-devel@lists.sourceforge.net if you wish to be added.

- Run "svn update"Every time you start a programming session with the STK, and also just prior to running "svn commit". This will ensure that you resolve any potential conflicts prior to committing changes.

- Please only commit when you have a working STK: do not commit code that will break the build. We need to ensure that anyone who checks out our STK from the trunk/ can at least build the project, even if the trunk is in a snapshot state (which it will normally be in).
- Each time we create a point or full release, it will be tagged and marked in the tags/ subdirectory of the project. This means that users can always choose to stay with a particular release. If we need to bug-fix a particular release separately from what we're already building in the trunk, then we can create a branch for that release in the branches/ subdirectory and work from there. The tags/ subdirectory is never to be committed to, as that would be against Subversion best-practices.

More information on Subversion can be found in its Manual .

5. Beyond Javadoc: Documenting Your Work

If any of the changes you make change how a user would install or use this STK, please ensure that you update the XSD documentation, which we make available via the website both in html and pdf format. The actual documentation is built using the APT (Almost Plain Text) Format and then generated by maven into other formats.

The APT files are made into a book using the doxia maven plugin - this is a plugin that, in a more generic usage, also builds the maven site documentation when you run the command mvn site:site . The benefit of using the Doxia book plugin is that it can create the book in multiple versions: currently we build the book in xdoc format, which is used by the mvn site:site command, and in pdf, which is manually copied to the FuGE website for download and offline use. Please see the Doxia Plugin Manual and information on writing in APT format .

You build the new version of the book by going into the books/ sub-directory and running the book-generation command:

```
cd books/
mvn doxia:render-books
```

This builds the book in the books/target/generated-site directory.

6. Re-Building the FuGE XSD STK Website

When you are ready to re-build the entire website, go into the top-level directory and generate the site docs in each of the sub-projects:

```

```

```
mvn site:site
```

This actually builds all of the html files. However, they are still in their individual sub-directory's target/site directory. To put them all together and copy them to the FuGE website, you need to use the mvn site:deploy command. However, to prevent accidental copying, the shell.sf.net site element is commented out. Please ensure this remains the case when you are committing changes to SVN.

Also edit your $M2_HOME/settings.xml file by adding the following:

```
<server>
<id>shell.sf.net</id>
<username>your sf username</username>
<password>your sf password</password>
</server>
```

mvn site:deploy JUST copies the already-generated site docs. If you make a change to any of the APT files, you must run mvn site:site FIRST, to re-generate the site docs, THEN mvn site:deploy to copy them to the final location described in the distributionManagement element of the pom.xml

It is a good idea to test the maven-generated site before publishing to the fuge website. Change the url element of the local-test site element below to local directory that is right for you, then deploy locally:

```
mvn site:deploy
```

and check the resulting site. If all looks OK, comment-out the local-test site element and un-comment the shell.sf.net site element. Only developers with write-access to the shell.sf.net server and the fuge group area will be able to perform this goal. Run the deploy command again to copy to the real SourceForge webserver:

```
mvn site:deploy
```

The only problem with the site deployment at the moment is that it doesn't copy the non-html books properly. This means that the pdf version of the installation instructions doesn't get put on the server. So, as a final step, you need to scp the pdf manually:

```
 scp books/target/generated-site/pdf/fuge-xsd/fuge-xsd.pdf  your-
username@shell.sf.net:/home/groups/f/fu/fuge/htdocs/stks/xsd-stk/
```

You can only run site:deploy successfully if you are down as a FuGE developer on the SF site.