

**Hydra Development Guide**

Version: 1.0

Last Modified: 10/12/2010

Team : caCORE SDK Team

Client : National Cancer Institute -   
 Center for Bioinformatics and Information Technology,

National Institutes of Health,

US Department of Health and Human Services

Document History

Revision History

| **Version Number** | **Revision Date** | **Author** | **Summary of Changes** |
| --- | --- | --- | --- |
| 1.0 | 10/08/2010 | John Chen | Initial Draft |
| 1.0 | 10/08/2010 | Bediako George | Updated |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Review

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Team/Role** | **Version** | **Date Reviewed** | **Reviewer Comments** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Related Documents

More information can be found in the following related documents:

|  |
| --- |
| **Document Name** |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

Product Team

|  |  |  |
| --- | --- | --- |
| **Management** | **Role** | **Comment** |
| Sichen Liu | Product Manager |  |
| Sreenath Nampally | COTR |  |
| Greg Raley | Technical Project Manager |  |
|  |  |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| **Development** | **Role** | **Comment** |
| Satish Patel | Technical Architect |  |
| Bediako George | Developer |  |
| John Chen | Developer |  |
| Vijay Parmar | Developer |  |
| Dan Dumitru | Developer |  |

|  |  |  |
| --- | --- | --- |
| **Testing** | **Role** | **Comment** |
| Ye Wu | QA Manager |  |
| Xiaoling Chen | QA Engineer |  |
| Aynur Abdurazik | QA Engineer |  |
|  |  |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| **Documentation** | **Role** | **Comment** |
| Jill Hadfield | Documentation Manager |  |
| Ann Wiley | Wiki Documentation |  |
| Bronwyn Gagne | Documentation |  |
| Carolyn Klinger | Documentation |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| **Other** | **Role** | **Comment** |
| Kelley Hunsecker | Download Center Updates |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Product Information

|  |  |  |
| --- | --- | --- |
| **Resource** | **URL** | **Comment** |
| Product URL |  |  |
| SCM URL | <http://gforge.nci.nih.gov/svnroot/cacoresdk/trunk/hydra/> |  |
| Defect Tracking |  |  |
| Feature Request |  |  |
| Listserv |  |  |
| Support |  |  |
| Documentation | <http://gforge.nci.nih.gov/svnroot/cacoresdk/trunk/hydra/src/doc/> |  |

Cross-Product Dependencies

|  |  |  |
| --- | --- | --- |
| **Resource** | **Contact** | **Comment** |
| Semantic Infrastructure v2.0 | Dave Hau (Product Manager) |  |
| caGrid (PST) v2.0 | Avinash Shanbhag (Product Manager) |  |
| caAdapter v5.0 | Sichen Liu (Product Manager) |  |
| CSM | Sichen Liu (Product Manager) |  |
| CLM | Sichen Liu (Product Manager) |  |
| ISO 21090 | Sichen Liu (Product Manager) |  |
|  |  |  |

External Software Dependencies

|  |  |  |
| --- | --- | --- |
| **Resource** | **URL** | **Comment** |
| Java |  |  |
| Enterprise Architect |  |  |
| ArgoUML |  |  |
| JBoss |  |  |
| Tomcat |  |  |
| Oracle |  |  |
| Postgres |  |  |
| MySQL |  |  |
| Ant |  |  |
| Eclipse |  |  |
| Windows |  |  |
| Linux |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

External Library Dependencies

|  |  |  |
| --- | --- | --- |
| **Resource** | **URL** | **Comment** |
| Eclipse |  |  |
| OSGI |  |  |
| UML |  |  |
| Spring |  | We only use this for the sdk example. |
| Hibernate |  | We are not using this |
| JAXB |  | WE only use this for the SDL example |
| JDOM |  | We are not using this. |
| JSR 286 |  | We are not using this. |
| Acegi |  | We are not using this. |
| Apache Commons |  |  |
| CXF |  |  |
| StringTemplate |  |  |
|  |  |  |

NCI CBIIT Specific Information

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Resource** | **App Server** | **DB** | **CI Server** | **Comment** |
| DEV Tier |  |  |  |  |
| QA Tier |  |  |  |  |
| STAGE Tier |  |  |  |  |
| PROD Tier |  |  |  |  |
| TRAIN Tier |  |  |  |  |
| DEV CI Server |  |  |  |  |
| AntHillPro CI Server |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Table of Contents

1. Introduction 8

1.1 Intended Audience 8

1.2 Purpose of the Document 8

2. Knowledge Required 8

3. Skills required 8

4. System requirement 9

5. Hydra directory structure 10

6. Example Eclipse installation and directory structure 11

7. Building the Project (command line) 12

7.1 Build the core generator and ide 13

7.2 Creating the SDK Example Plugin 13

7.3 Test driven development 13

7.3.1 Test Inventory 14

8. Building and running Hydra from Eclipse 14

9. Class Diagrams of Hydra 29

9.1 Logical Model of Hydra 30

9.2 Model Converter class diagram 31

9.3 IDE Plug-in class diagram 32

9.4 Core class diagram 33

10. Notes on creating generator plugins 33

10.1 Structure 33

10.2 Writing a generator script 34

# Introduction

## Intended Audience

This document is intended for Software Developers who intend to develop the software. This document contains high level information that may be of interest to the end users of the product but the document itself is not intended to cater them.

## Purpose of the Document

The purpose of this document is to provide detailed guidance on developing Hydra software. This document outlines skills, domain knowledge for anyone who wants to develop for Hydra. It also specifies how to setup Hydra development environment. Class diagrams of major Hydra IDE classes are provided in a separate artifact.

# Knowledge Required

1. Model driven engineering knowledge, including domain or class representations and code generation of artifacts from domain classes.
2. Semantic infrastructure knowledge, including an understanding of concepts, classes, operations, relationships, and properties.

# Skills required

The following skills are required for being a Hydra developer.

1. UML
2. Java 1.6
3. JUnit4.0
4. Eclipse IDE
5. Eclipse plug-ins and OSGI
6. EMF, SWT, JFace
7. String Template
8. JSR-223 and Java scripting
9. Dynamic class loading

To support the SDK example an understanding of the following is helpful:

1. Apache CXF
2. XSD
3. XML
4. JAXB
5. Java compiler
6. ANT
7. Spring
8. Web.xml and servlets
9. SOAP
10. Grid computing

# System requirement

The following software must be installed before developing Hydra.

1. JDK1.6 (<http://www.oracle.com/technetwork/java/index.html>)
2. Eclipse IDE 3.5 (<http://www.eclipse.org/galileo>)
3. Eclipse UML2 plug-in (<http://www.eclipse.org/uml2>)
4. The latest Hydra code (<https://gforge.nci.nih.gov/svnroot/cacoresdk/trunk/hydra>)
5. Subversion repository (<https://gforge.nci.nih.gov/svnroot/cacoresdk/trunk/hydra>)
6. Ant (for command line builds)

Please note that the current Hydra release does not support the latest Eclipse IDE 3.6 version. Also Hydra’s build script is developed under the JDK1.6 environment. Hydra can be checked out anonymously from the subversion repository URI listed above using this command:

svn co <https://gforge.nci.nih.gov/svnroot/cacoresdk/trunk/hydra> **hydra**

# Hydra directory structure

Since Hydra is an Eclipse plug-in, its directory structure follows closely to the default plug-in structure of Eclipse. The following shows the description of some basic directory and file names of Hydra.

.classpath ----------- Eclipse .classpath file

.project ------------- Eclipse .project file

build.properties ----- Eclipse plug-in file

plugin.xml ----------- Eclipse plug-in configuration file

build.xml ------------ Ant script

properties.build ----- Ant script properties file

META-INF/ ------------ Eclipse plug-in configuration file

build-only-lib/ ------ library jar files needed to compile Hydra

runtime-lib/ --------- library jar files needed at runtime

sdkexample/ ---------- Sample example application. This application’s directory structure is modeled after the hydra directory structure.

src/ ----------------- Hydra source code

doc/ ------------- Hydra design documents, user guide, etc.

images/ ---------- Any image files used by Hydra

java/ ------------ Java source code

test/ ------------ Test related code

model/ ------- Test model file and xmi file

dist/ ---------------- All library artifacts created by the build are placed here.

deploy/ -------------- If any deployable artifacts are created they should be delivered here. Hydra does not yet create a deployable artifact.

result/ -------------- Build processes that create readable results should place those files here. JUnit is an example of one such process.

stage/ --------------- Source code is sometimes prepared for compilation or other processing in the stage directory.

workspace/ ----------- Not to be confused with the eclipse workspace, developers can use this directory as a scratch pad.

###### More directory notes

Under the src/java package the “generator” directory contains the code that implements the core generator. The “ide” directory contains all the code that is used to create the eclipse perspective.

The sdkexample directory contains all the code that is needed to implement the example sdk plugin generator. Its directory structure is mirrored after the hydra directory structure.

# Example Eclipse installation and directory structure

To make maximum use of the examples and screenshots in this guide, you should use the same directory structures and versions of Eclipse that were used when creating the guide. (The examples were created on Windows XP.)

Create empty directories c:\tools\UIProtoTest\ and c:\tools\UIProtoTest\workspace for Eclipse and the Eclipse workspace.

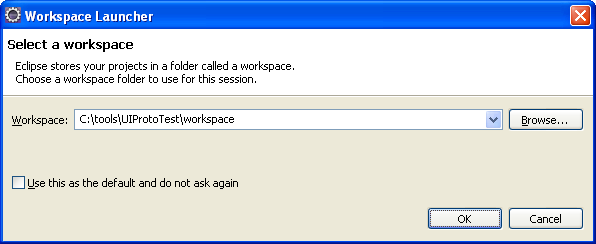
Create c:\main\sdk\Hydra and download the source code into it.

Go to Eclipse Galileo Page at <http://www.eclipse.org/downloads/packages/release/galileo/r> and download “Eclipse IDE for Java EE Developers” from this link: http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/galileo/R/eclipse-jee-galileo-win32.zip

Unzip (with paths) and then move so the root eclipse directory is at c:\tools\UIProtoTest\eclipse

Run Eclipse from c:\tools\UIProtoTest\eclipse\eclipse.exe

Select c:\tools\UIProtoTest\workspace as the workspace when Eclipse starts:

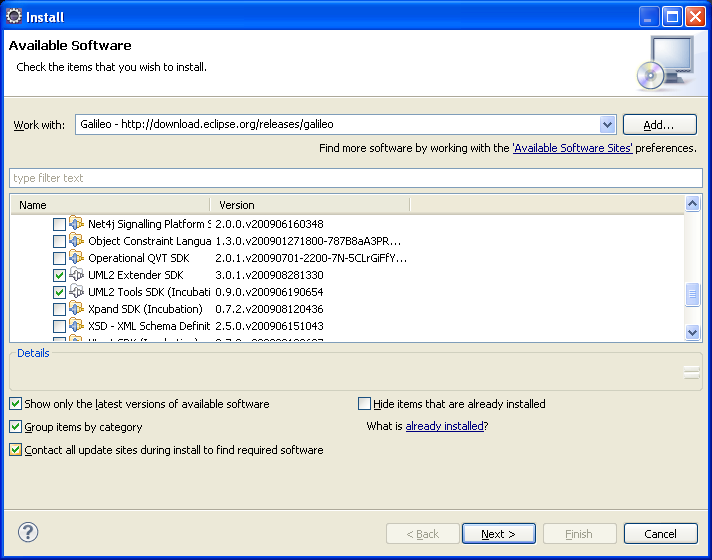


Open menu Help -> About Eclipse. Verify the version number is 20090621-0832

Open menu Help -> Install New Software

Select the Galileo update site: Galileo - http://download.eclipse.org/releases/galileo

Expand Modeling and check the two UML2 plugins:



Complete the wizard to download and install UML2 plugins. (Note: Be sure to checkmark items for security exceptions if asked whether to grant security exceptions. The highlighting is misleading.)

# Building the Project (command line)

Hydra builds are primarily accomplished via ant scripts. The core generator and the sdk example can each be built from the command line by typing “ant redeploy” from their root directory. This command will build an application and produce a jar file artifact in the dist/lib directory. Note that the sdk example plugin generator is in its own directory structure called “sdkexample” under the “hydra” directory.

## Build the core generator and ide

C:\main\sdk\hydra>ant redeploy

## Creating the SDK Example Plugin

The SDK Example plugin generator can be created by using the makePlugin target of the ant script. To do this simply type at the command line in “hydra/sdkexample”:

ant makePlugin

This will create a zip file that contains all the generator code needed to create a generator plugin. To install this plugin simply use the installPlugin target within ant as follows:

ant -DinstallDir=<your install dir full path name> installPlugin

C:\main\sdk\hydra\sdkexample>

ant -DinstallDir="c:\tools\UIProtoTest\eclipse\generators\sdkexample" installPlugin

This will copy your plugin zip file to the specified directory and unzip your newly created plugin into that directory. This is how the plugin is deployed for demo purposes. It is intended that the plugins of the future will be OSGI compliant. As such this target should be refactored to accomplish this task.

## Test driven development

We follow test driven development (TDD) as much as we can. As you can see from the above listing, sample model files are provided by Hydra. We support two types of testing, first standard unit testing using JUnit to create test cases. The second way is via JavaScript scripting. Both of these testing measures can be executed from the command line via ant. To run unit tests, you simply use the “unit-test” target in the ant build file like so:

ant unit-test

e.g. C:\main\sdk\hydra>ant unit-test

and to execute scripts from the command line simply use:

ant -Dscript=<relative path name of your script from src/script, omit .js extension> runScript

e.g. C:\main\sdk\hydra\sdkexample>ant -Dscript=test\testGeneratorUtil runScript

Both of these methods support testing of application functionality. There are already several tests that rely on these features. Use these tests as an example of how to test Hydra code. Review the src/script directory in both the hydra and SDK example root directories.

### Test Inventory

The following is a list of test suites already created in the Hydra project.

1. ModelConverterUtilTest.java – tests methods that execute functionality around model conversion.
2. AnnotationTagTest.java – tests methods that execute functionality around Ecore annotation building.

#### XMI2EcoreModelConverterTest.java – test methods that execute functionality around EA model to ECore conversion.

#### SDKUtilTest.java – test methods that execute functionality around SDK Hydra tag values.

1. testGeneratorUtil.js – tests functionality of the core SDK Hydra generator.
2. testSdkExampleGenerator.js – test the creation and execution of a pluggable sdk example generator

# Building and running Hydra from Eclipse

After setting up the environment, checking out the Hydra code, and running makePlugin and installPlugin, you can start up Eclipse and use the Hydra IDE in development mode.

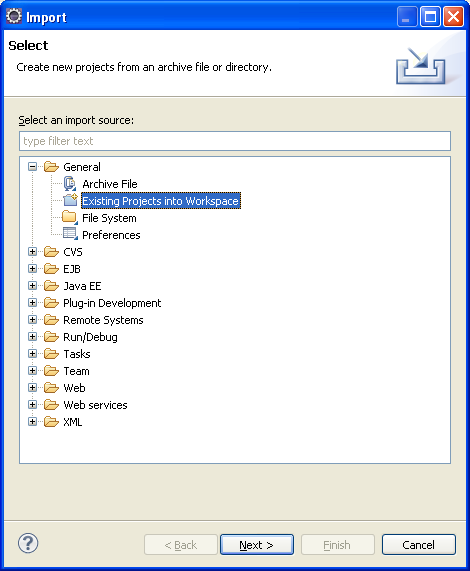
The following screen snapshots give step-by-step instructions on creating a project and running Hydra as an Eclipse plug-in.

In this example, the Hydra code was previously checked out to c:\main\sdk\hydra.

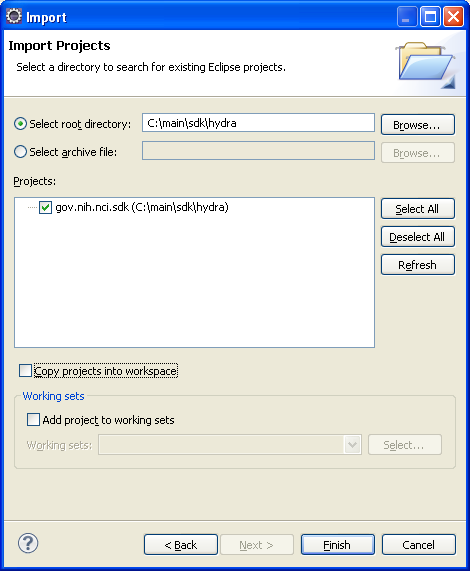
Start Eclipse and select the c:\tools\UIProtoTest\workspace workspace.

Open the Java Perspective.

Right-click in the Package Explorer and select Import…

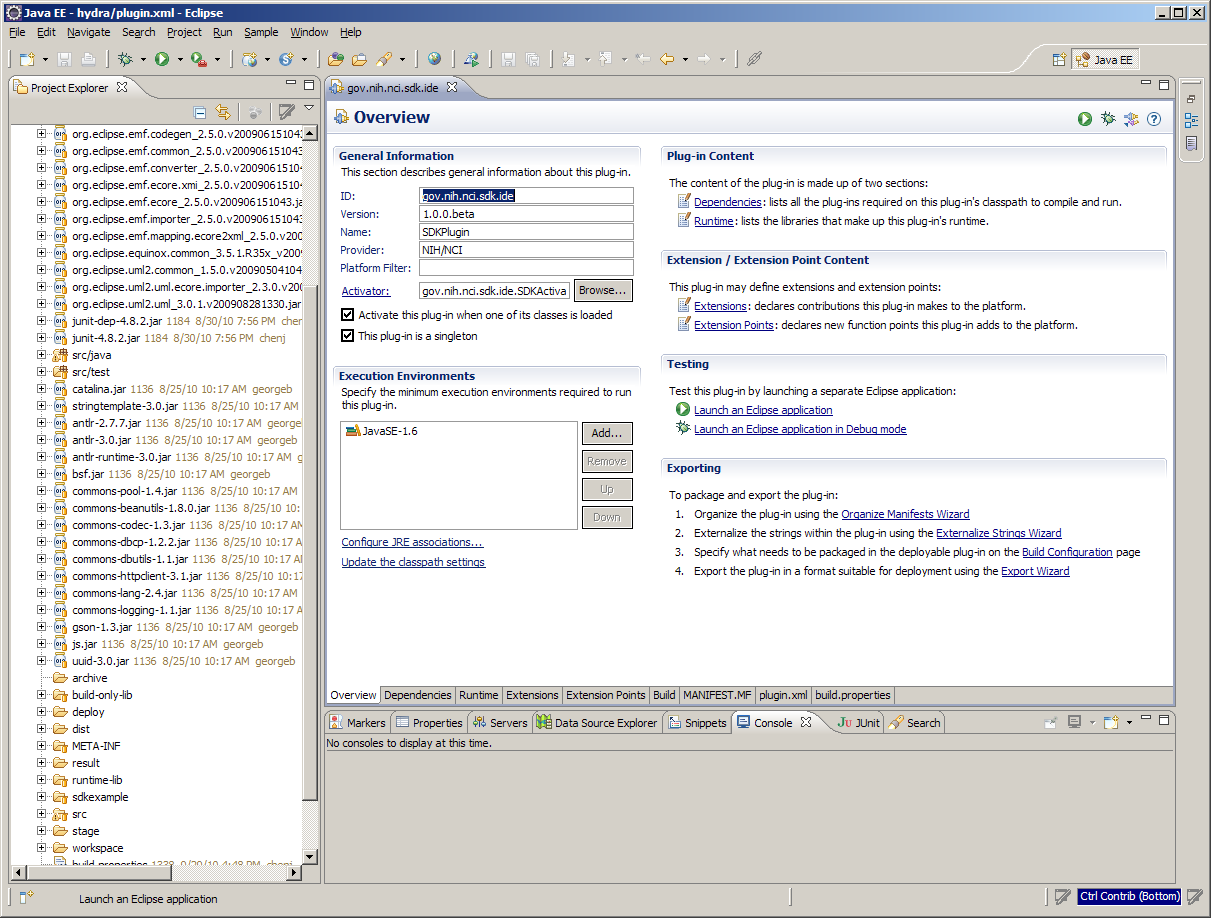


Select General—Existing Projects into Workspace and press Next.

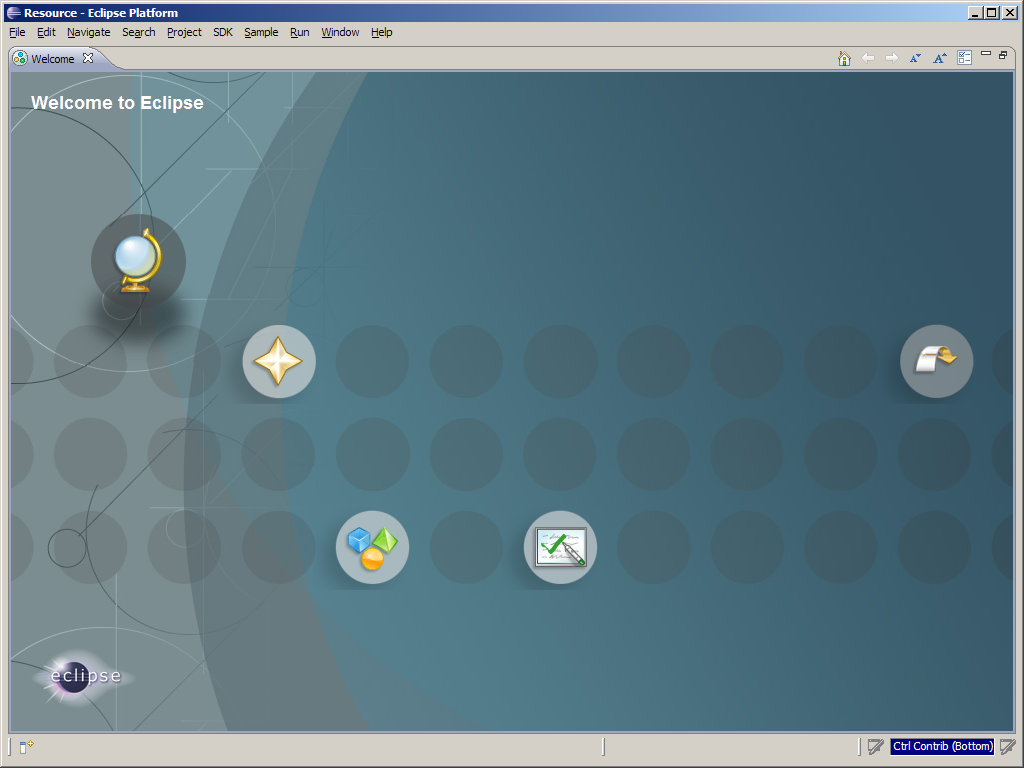


Search for the Hydra code in c:\main\sdk\Hydra and checkmark the project. Press Finish.

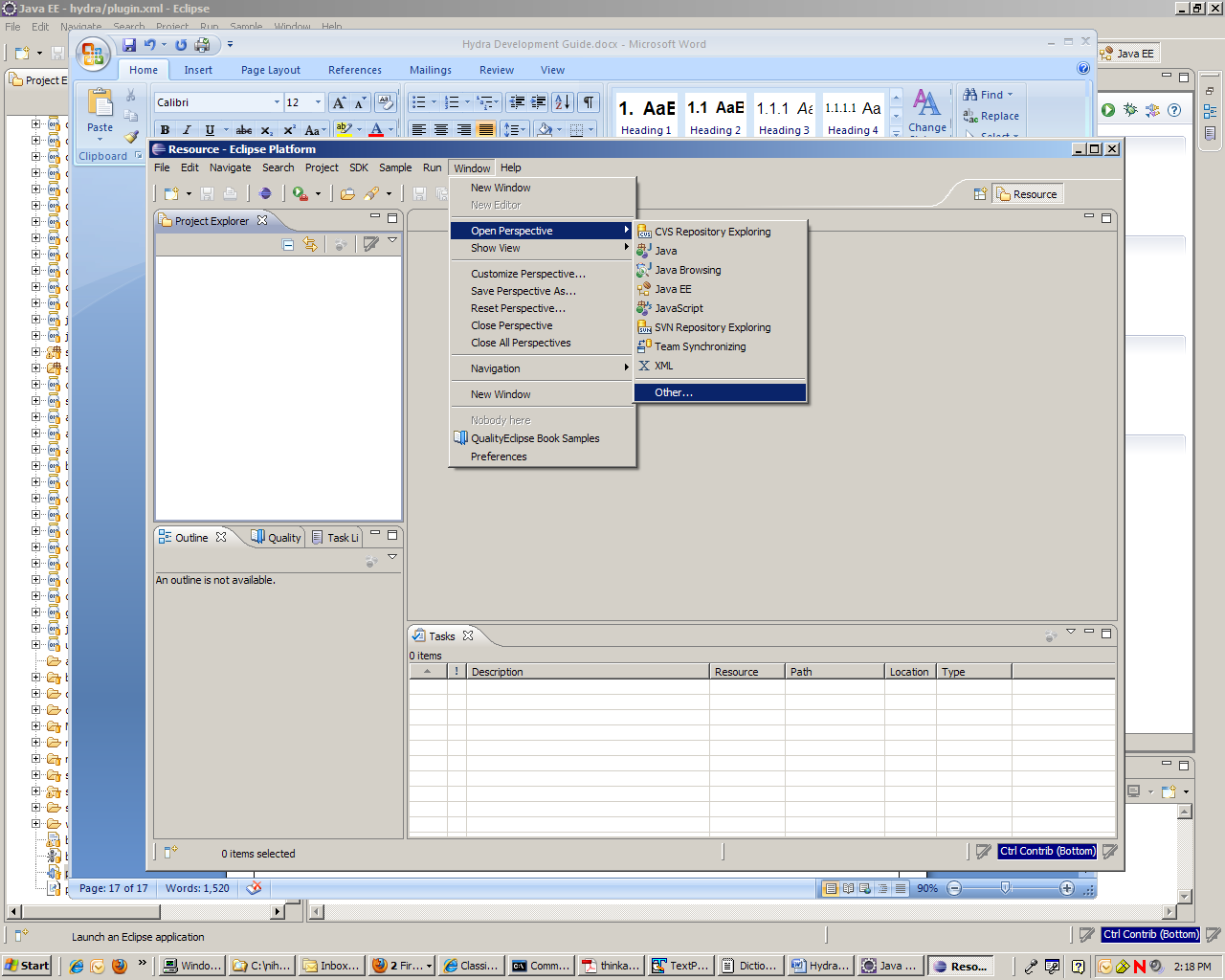
After the import (which automatically builds), open the project tree and double-click plugin.xml to open the plugin editor.



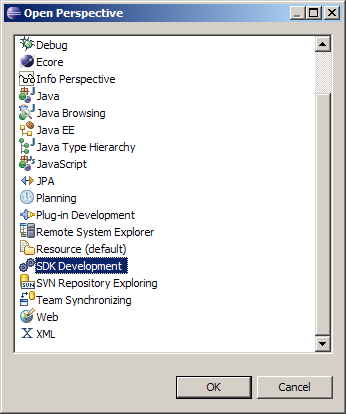
In the Overview tab of plugin editor, click on the link named “Launch an Eclipse application”



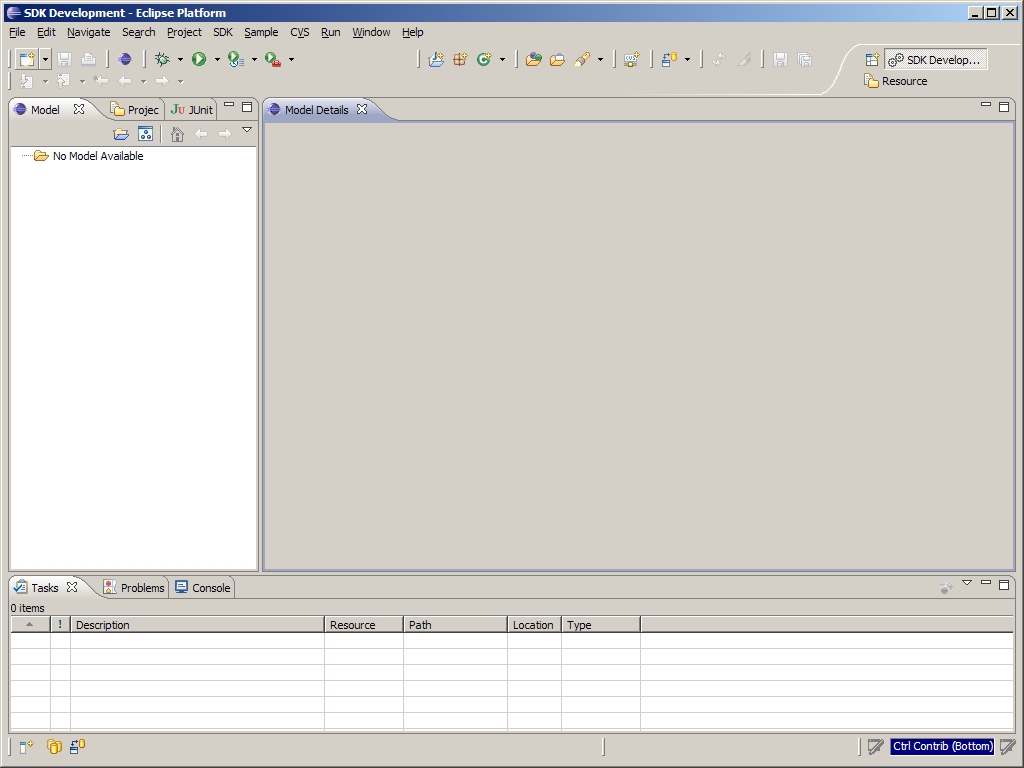
A new Eclipse instance appears. This is the Eclipse plugin test instance.



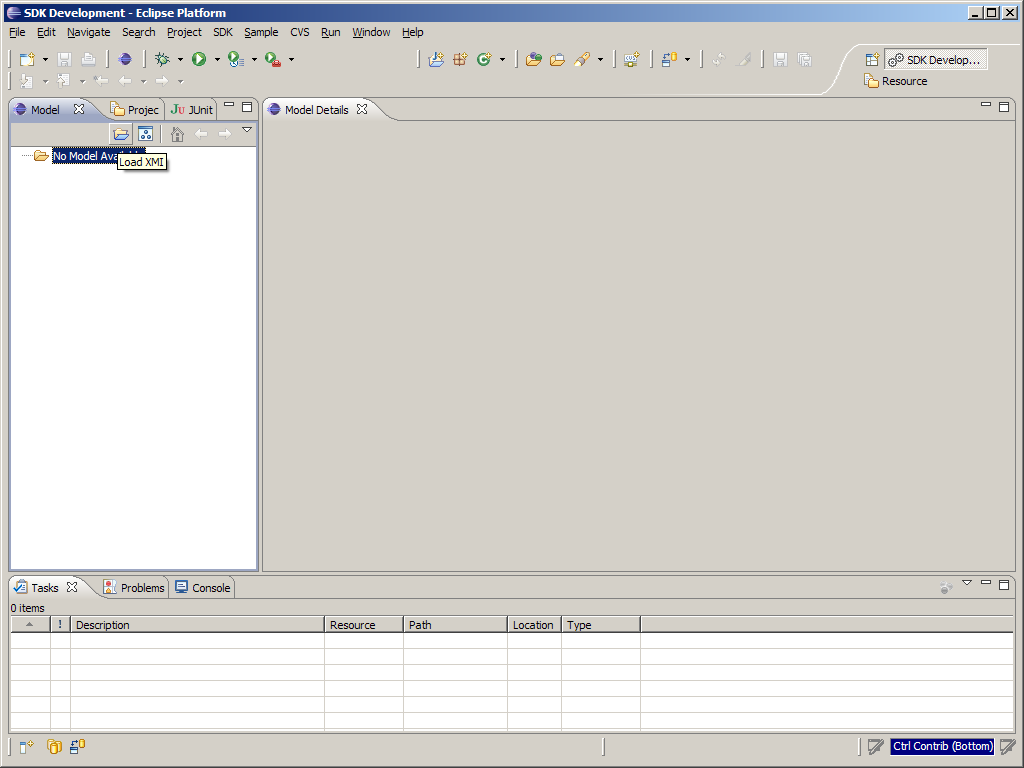
In the newly launched Eclipse Test app, go to Window -> Open Perspective -> Other.



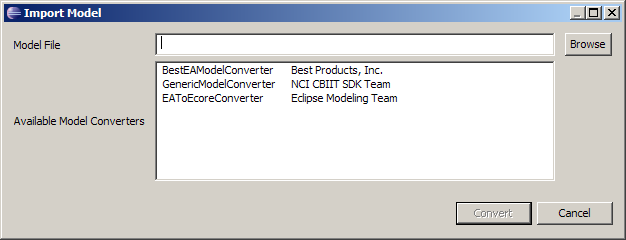
Select “SDK Development” and click on OK button. The Hydra SDK development perspective views appear as follows. Opening the SDK Development perspective “runs” the prototype.



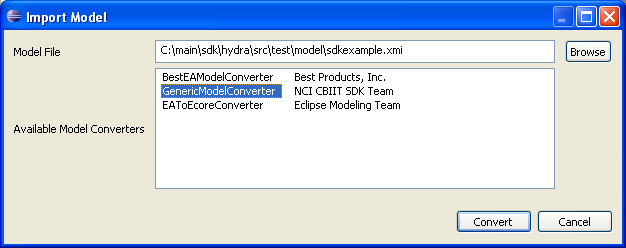
This is the SDK Perspective. It consists of two major panels. The left navigational master panel and the right model details panel.



Click on the Open folder icon on top of the Model selection panel. A file dialog window is going to show up.



Click on the Browse button to select the XMI file to be converted.

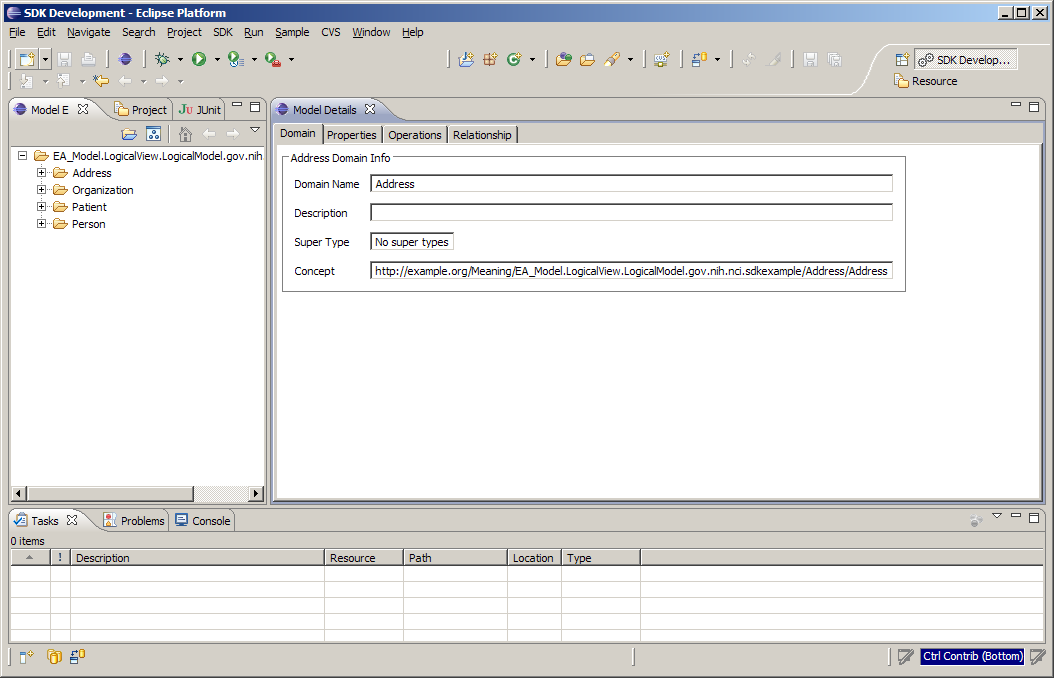


Navigate to C:\workspace\hydra\src\test\model\ directory and select sdkexample.xmi file.

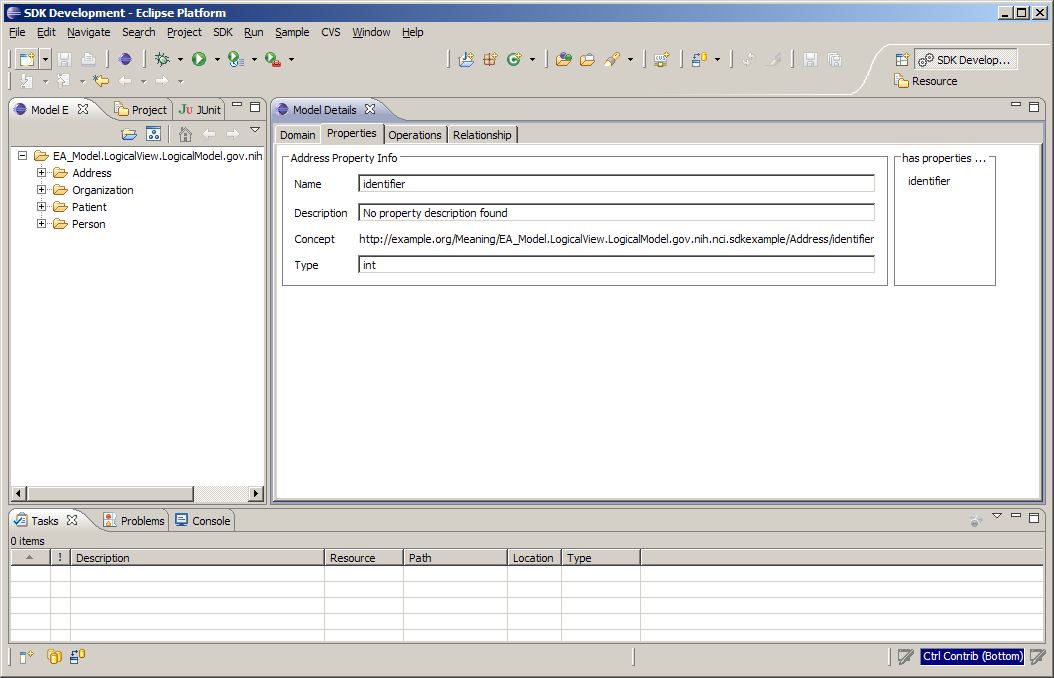
Also select a converter from the list of converters available. Once both are selected, the Convert button will be enabled.

Click on the Convert button to convert the XMI file to Eclipse modeling framework Ecore model.

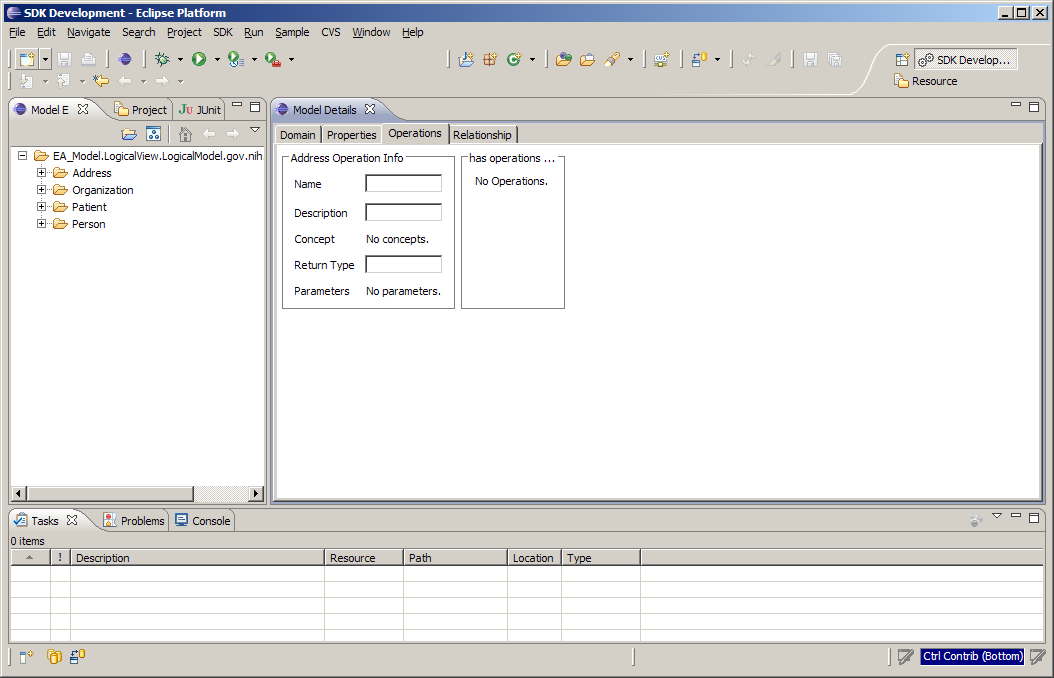
Once this is done, the following screen shows up.



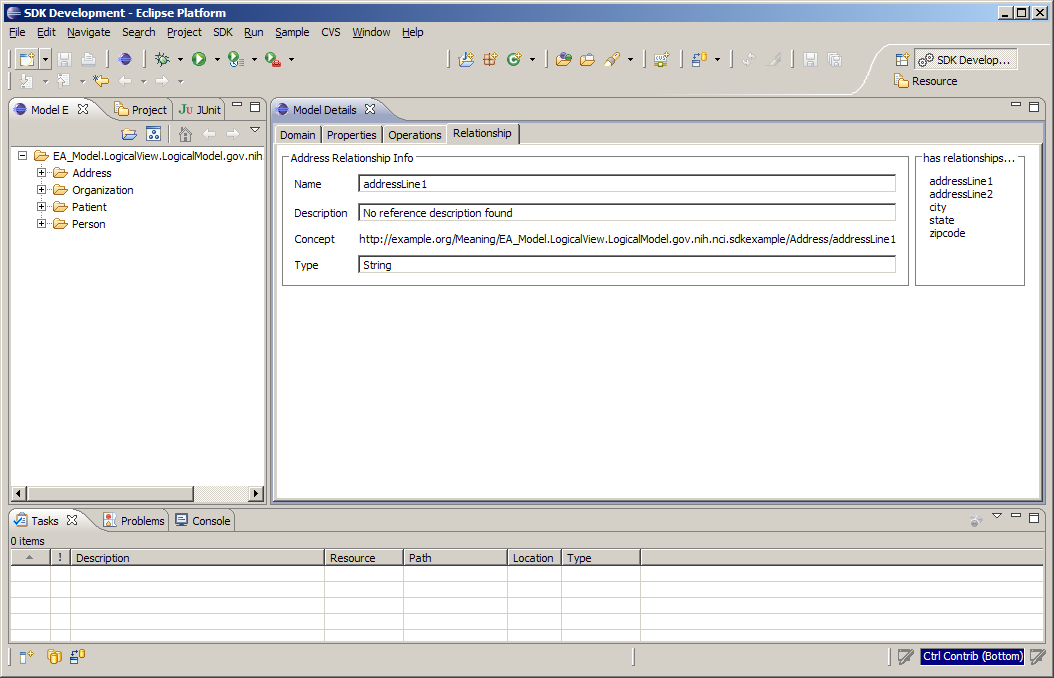
A list of models shows up on the left panel. On the right panel, meaning details of the first model “Address” is displayed.



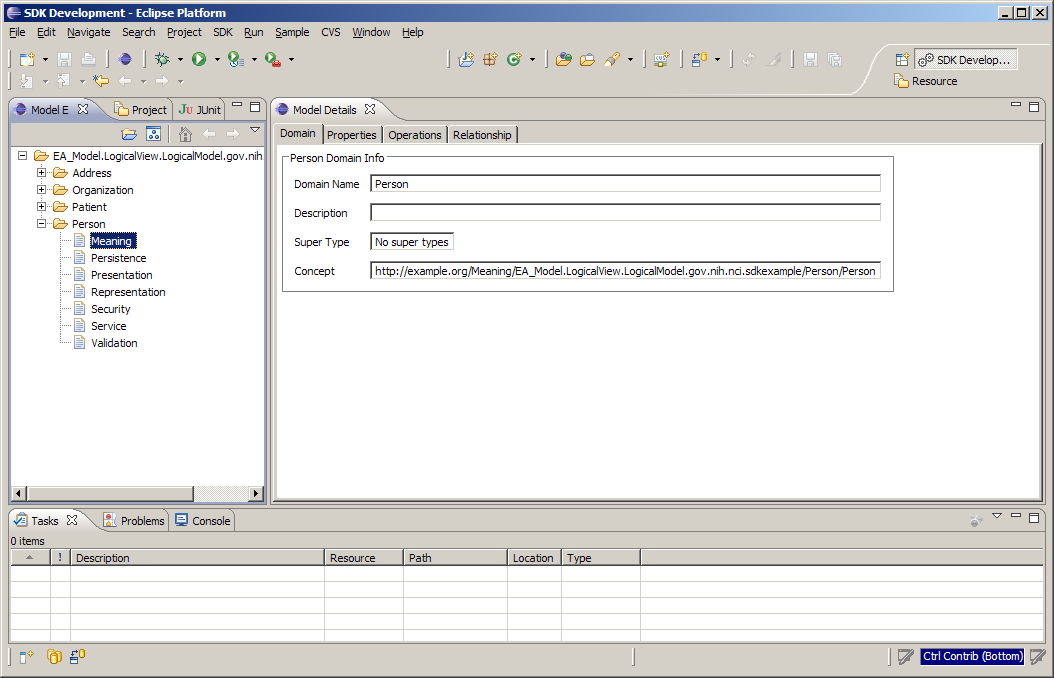
You can click on the Properties tab to view properties of the Address model.



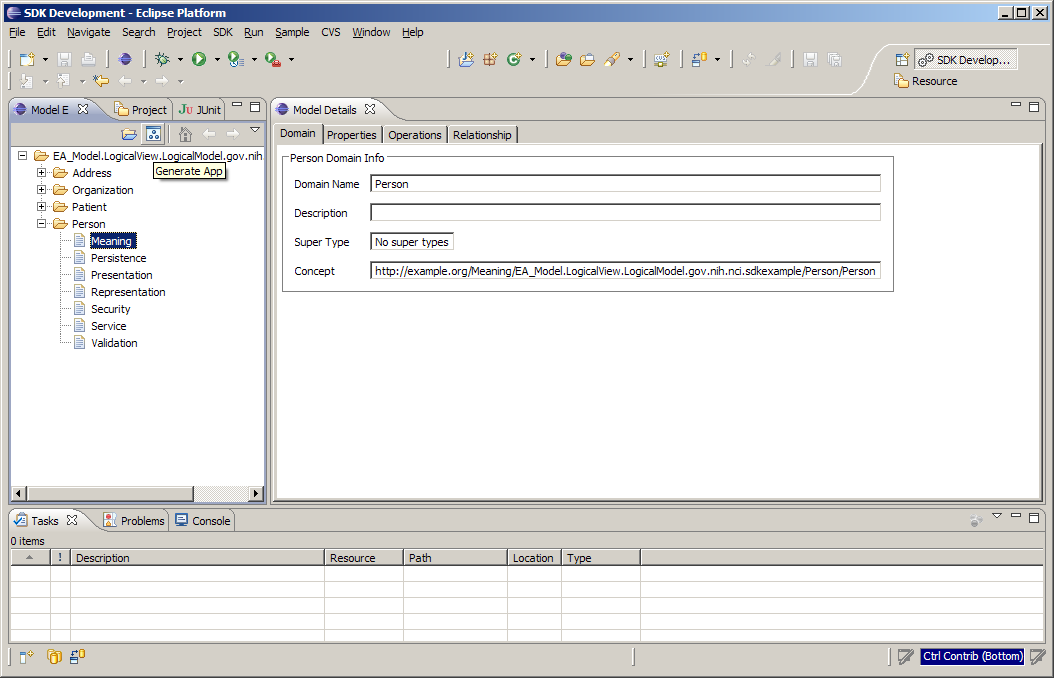
You can click on the Operations tab to view operations of the Address model.



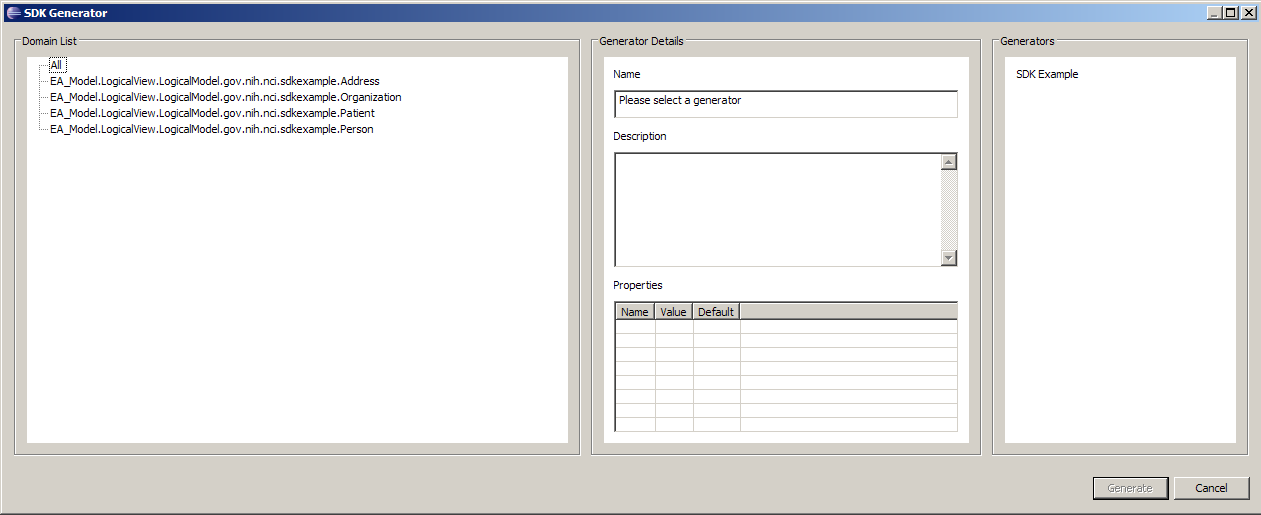
You can click on the Relationship tab to view relationships of the Address model.



You can also select another model and view its meaning, persistence information, etc.



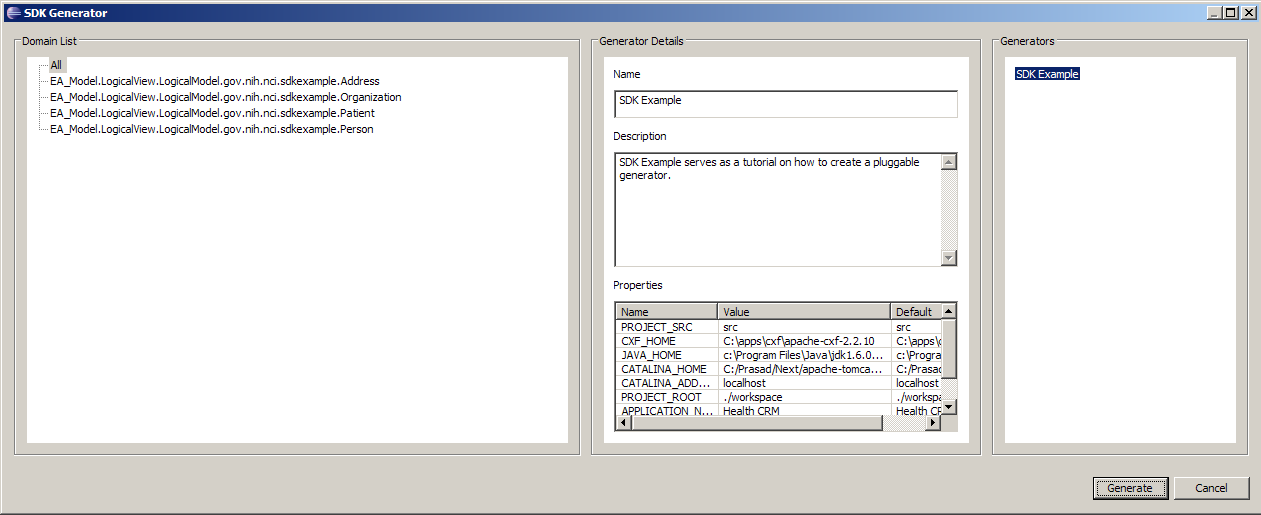
Once you are done reviewing the models, you can click on the “Generate App” icon which is located on top of the left panel besides the “Load XMI” icon.



This is the pop-up screen after you’ve clicked on the “Generate App” icon.

This screen consists of three panels. The left panel shows a list of domain models. The right panel shows a list of available generators. Currently there is only one generator “SDK Example”. The middle panel shows properties of the “SDK Example” generator.

Click on the “SDK Example” link on the right panel.



This screen shows details of the “SDK Example” generator.

Select a domain model or multiple domain models, the “Generate” button on the lower-right corner is going to be enabled.

Click on the “Generate” button, the application code is going to be generated in the location specified by the properties of the generator.

# Class Diagrams of Hydra

The diagrams below give developers a picture of the relationships among classes. For more details, please refer to accompanying source code and JavaDoc of the classes.

## Logical Model of Hydra



## Model Converter class diagram



## IDE Plug-in class diagram



## Core class diagram



# Notes on creating generator plugins

## Structure

Creating a generator plugin for Hydra simply requires conformance to a file and directory layout. All plugins should contain the following file structure.

* A root directory: The root directory will contain all of the files pertaining to the plugin.
* info.properties: This is a Java property file that contains a name property and a description property.
* generator.properties: This is a Java property file that contains properties that are of importance to the plugin generator.
* A set of script files: These are files that end with an extension that points to the scripting language supported by the Hydra generator. Hydra supports all scripting languages that obey the JSR-223 specifications. Hydra natively supports JavaScript, but will support Python, Ruby, Groovy, PHP, and other languages if the scripting libraries are included. These scripts are executed by the core generator on a per domain basis. The order of the execution is not defined; scripts may be executed in any order.
* A set of script file libraries: these are files that contain library code that can be executed by the aforementioned script files. They have extensions that are of the form "<scriptExtension>Lib". These files will not be executed by the core generator, but will be still accessible by the script files that accompany the plugin installation.
* A lib directory: This directory contains all the Java libraries needed to support the scripts that make up the plugin generator logic.
* A template directory: This directory contains StringTemplate template files that are to be used to create artifacts. All templates in this directory are made available to the script at runtime by a provided StringTemplateGroup object.

All of the above artifacts should be placed in a single zip file with a single root directory. This zip file should be exploded into the generator target directory upon installation. If the convention is followed, Hydra wil automatically find the scripts needed to execute the plugin, and will exceute those scripts against the specified ECore package representation.

## Writing a generator script

A script should expect to have access to a script context upon invokation. Script contexts contain all the information needed to completely generate any artifact from an Ecore model. The following lists the objects provided by a script context.

* StringTemplateGroup object: this object provides access to the templates located in the plugin's template directory.
* Logger object: this object is a java.util.logging.Logger object that provides logging capabilities to the script.
* ErrorManager and WarningManager: the script can use these managers to report warnings and errors to the core generator.
* Global memory: This is a hash map that allows scripts to store information that is accessible by other scripts for the duration of the generation run.
* Script memory: This is a hash map that allows scripts to store information that is accessible by itself and no other scripts during the duration of the generation run.
* ECore package: The root ECore package that contains all the model elements.
* FocusDomain: This is the Ecore class that stipulates the domain that is currently be processed.
* Properties: This is a java.util.Properties object that contains the properties that were declared in the generator.properties file.