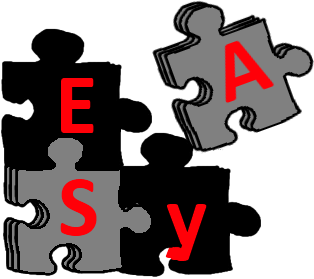
|  |  |  |
| --- | --- | --- |
| Uni_Logo.png |  | Zuschneiden.jpg |
| **Stiftung University of Hildesheim**  Marienburger Platz 22  31141 Hildesheim  Germany |  | **Software Systems Engineering (SSE)**  Institute for Computer Science  Faculty for Mathematics, Natural Science, Economics, and Computer Science |
|  |  |  |



**EASy Producer**

Engineering Adaptive Systems

**Developers Guide**

Version 0.2

10.09.2012

**Version**

|  |  |  |
| --- | --- | --- |
| 0.1 | 28.08.2012 | Initial version |
| 0.2 | 10.09.2012 | Reasoning section switched to the end of the document, prerequisite and installation added, debug flags added to section 3 |
|  |  |  |
|  |  |  |

Table of Contents

[1. Introduction 4](#_Toc335053565)

[2. Getting Started 5](#_Toc335053566)

[2.1. Prerequisites 5](#_Toc335053567)

[2.2. Installing EASy Producer 5](#_Toc335053568)

[2.3. Troubleshooting 7](#_Toc335053569)

[3. EASy Producer Extensions 8](#_Toc335053570)

[3.1. Implementing a New Instantiator 8](#_Toc335053571)

[3.1.1. Eclipse Plug-in Project Creation and Configuration 9](#_Toc335053572)

[3.1.2. Instantiator Implementation 13](#_Toc335053573)

[3.1.3. Instantiator Integration 16](#_Toc335053574)

[3.2. Implementing a New Reasoner 17](#_Toc335053575)

[4. Known Bugs 18](#_Toc335053576)

# Introduction

EASy Producer is a software product line engineering tool for the easy development of large-scale, multi-software product lines. This tool is available as an Eclipse plug-in using the EASy Producer update site: <http://projects.sse.uni-hildesheim.de/easy/>

This document provides a prototypical developers guide that introduces the reader to the basic capabilities of EASy and how to develop further extensions to this tool.

Please note, that this document is a living document which will evolve in accordance to the EASy Producer tool. We are working on a more exhaustive and complete developers guide to ease the development of extensions for EASy Producer.

# Getting Started

TODO

## Prerequisites

The following prerequisites are mandatory for using EASy Producer:

* Download an **Eclipse** package from <http://www.eclipse.org/downloads/>. We recommend using Eclipse version **3.7 (Indigo)**. All other Eclipse versions may also be sufficient as long as they work with Xtext version 2.0.1.
* Install **Xtext** **version 2.0.1** (typically, this is installed automatically when installing EASy Producer due to plug-in dependencies). We recommend using the Eclipse software update function if installing Xtext manually. However, the Eclipse update site for Eclipse versions greater than 3.7 may only provide Xtext greater than version 2.0.1. Please note that EASy may not work properly with Xtext versions unequal to version 2.0.1. In Eclipse 3.7, download the required Xtext plug-ins by selecting the Indigo update site (cf. Figure 1). It might be necessary to uncheck the “*Group items by category*” checkbox in the lower location of the dialog to make these plug-ins selectable.

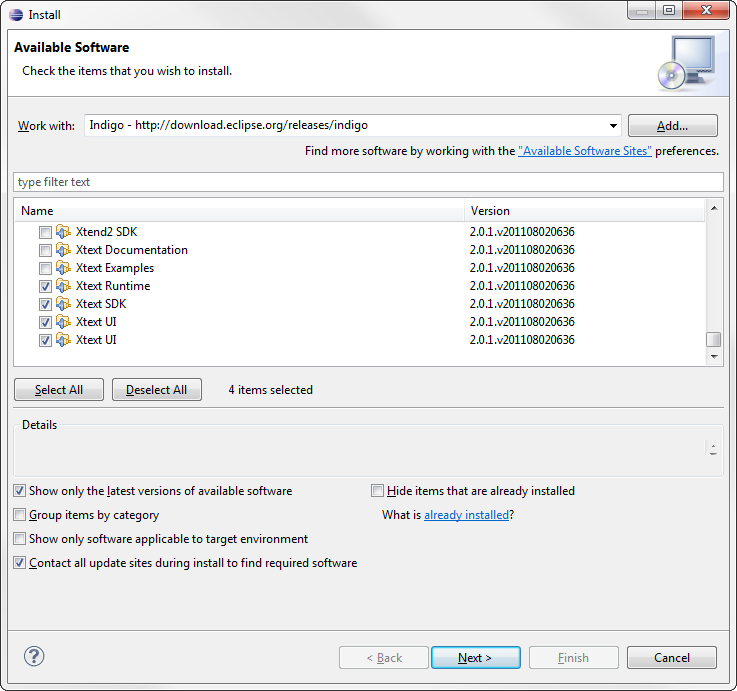


Figure 1: Installing Xtext from the Eclipse Indigo update site

If you are not familiar with the Eclipse software update function, please consider Section 2.2 on how to install new software from software update sites.

## Installing EASy Producer

We recommend using the EASy Producer update site[[1]](#footnote-1) for installing the required plug-ins. Thus, the first step is to open Eclipse and then open the “*Install New Software*” dialog as shown in :

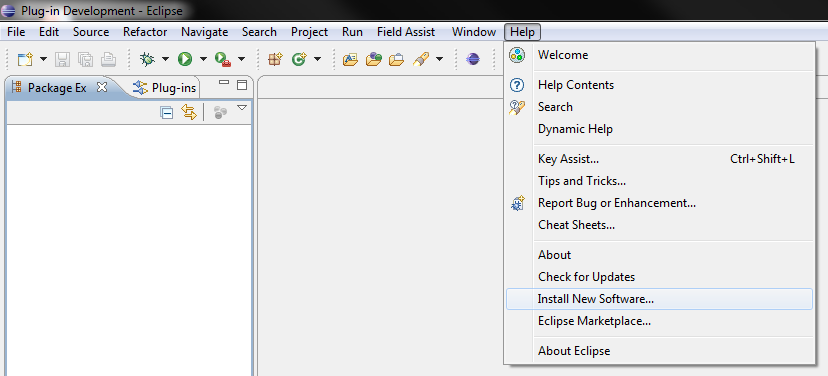


Figure 2: Open the “Install New Software” dialog

The “Install” Dialog will appear (cf. ). In this dialog, a new location for available software has to be added. Thus, click on the “Add…” button in the upper right location of the dialog.

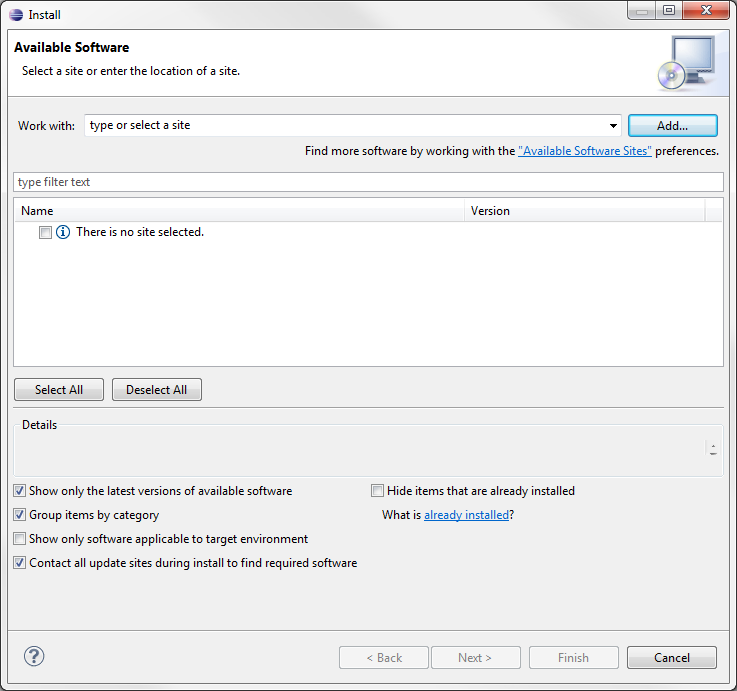


Figure 3: Add a new location for software updates

The “*Add Repository*” dialog requires the definition of a name for the new update site and a location. The name is up to the user. For example, enter “*EASy Producer Update Site*”. The location is the URL of the update site. Finish the definition of the new update site by clicking the “*OK*” button of the “*Add Repository*” dialog.

The “*Install*” Dialog will now contain multiple categories. We recommend to select at least the EASy Producer category, EASy Producer Instantiation category (or at least one of the instantiators listed as feature), and EASy Producer Reasoning (or at least one of the reasoners listed as feature).

Please note that Jess Reasoner must be downloaded manually due to license restrictions.

Follow the steps for installing EASy Producer (accept the license agreement and ignore the security warning for installing software of unsigned content, etc.), and restart Eclipse as prompted.

Finally, you have successfully installed the EASy Producer.

## Troubleshooting

TODO

We recommend increasing the memory of your eclipse application in which you are running EASy Producer.

Open the eclipse.ini file in your eclipse directory and enter the following parameters at the end of the file:

-vmargs

-Xms40m

-Xmx512m

-XXMaxPermSize=128m

# EASy Producer Extensions

EASy Producer provides an extension point mechanism to add additional functionality to the basic implementation. An extension is always implemented as an Eclipse plug-in and may provide customer-specific functionalities in terms of individual reasoner or instantiators. A new reasoner may provide new or adapted capabilities to check, for example, whether a variability model or a specific product configuration is valid. Custom instantiators may be capable of instantiating artifacts of different types or in a specific way. In order to easy the development and integration of such extensions, EASy Producer is capable of automatically search and integrate new plug-ins. Thus, developers only have to provide the necessary information to EASy Producer to include their desired functionalities.

In this section, we will describe how to implement extensions to the EASy Producer tool. In Section 3.1, we will describe the implementation of a new reasoner and its integration with EASy Producer, while Section 3.2, we will describe the implementation and integration of a new instantiator. Both sections will provide detailed guidance from project creation and configuration to the actual application of the custom plug-ins.

In order to debug errors and failures during the development of EASy Producer extensions, add the following flags to the “Run Configuration” of your Eclipse as needed (introduce the new flags with a single prefixed “-D” in the Run Configuration):

* -debug: This flag will print information on the variability model of EASy Producer
* -log: This flag will print EASy-internal debug messages, such as errors, etc.
* -equinox.ds.debug: This flag will print debug messages regarding the service registration mechanism. For more details, see Section 3.2.1.
* -equinox.ds.print: This flag will print additional information regarding the service registration mechanism. For more details, see Section 3.2.1.

## Implementing a New Instantiator

An instantiator is an external and maybe third-party tool that processes product line artifacts in its specific way. For example, the Velocity instantiator, which is shipped as a default instantiator with EASy Producer, resolves Velocity-specific tags within Java code in accordance to a specific configuration[[2]](#footnote-2). This resolution capability allows deriving individual product variants based on the configuration values and the corresponding manipulation of the Java code. However, the default instantiators of EASy Producer may be insufficient in some situations. Thus, we provide a simple extension mechanism for integrating custom instantiators with Easy Producer.

In the first part of this section, we will describe how to set-up a new plug-in project in Eclipse. This also includes the specific configurations that have to be done to utilize the automated search and integration mechanism provided by EASy Producer. The second part will discuss the methods that are required when implementing a new instantiator. The focus of this part will be on how, when and why EASy Producer invokes specific methods of an instantiator. In the third part, we will finally show how to integrate a new instantiator.

### Eclipse Plug-in Project Creation and Configuration

The first step towards a new instantiator is to create new Eclipse plug-in project: *File 🡪 New 🡪 Other…* . In the emerging wizard, open the category *Plug-in Development*, select *Plug-in Project*, and click the *Next* button. In the *New Plug-in Project* wizard, define a name for your project. We will use the following name throughout this section: *EASyExampleInstantiator*. Further, define the target platform with which the plug-in should run. In this case, the instantiator plug-in will run with a *standard OSGi framework*. Figure 2 shows how the first configuration page for the new plug-in project must look like.



Figure 1: Configuration of a new Eclipse plug-in project for a new Instantiator

Click on the *Next* button and define the properties of your plug-in. We will use the following values for the required properties:

* *ID*: de.uni\_hildesheim.sse.easy.instantiator.exampleInstantiator
* *Version*: 0.0.1
* *Name*: EASyExampleInstantiator
* *Provider*: University of Hildesheim – SSE

Leave all other properties and options as-is and finish the configuration by clicking the *Finish* button of the *New Plug-in Project* wizard. Please note that some of the following steps described in this section can also be done by using the wizard. However, we decided to do these steps manually to provide a more detailed explanation.

The plug-in manifest file will open by default. In the *Overview* tab check the *Activate this plug-in when one of its classes is loaded* checkbox and the *This plug-in is a singleton* checkbox. The first check will guarantee that the plug-in is activated when EASy Producer loads one of its classes, while the second check is related to one of the concepts of EASy Producer: each instantiator exists only once (only one instance) and can be accessed by any product line project. Thus, this check guarantees that the new instantiator will follow the concepts of EASy Producer.

The next step is to define the dependencies of the new plug-in. Thus, open the plug-in manifest and select the *Dependencies* tab. On the left side click the *Add…* button in order to specify the following plug-ins:

* *org.eclipse.equinox.ds*: This plug-in simplifies the task of authoring OSGi services by performing the work of registering the service and handling service dependencies[[3]](#footnote-3).
* *org.eclipse.core.runtime*: This plug-in provides support for the Eclipse runtime platform, core utility methods, and the extension registry[[4]](#footnote-4). The latter is important for the EASy Producer extension mechanism.
* *de.uni-hildesheim.sse.easy.instantiatorCore*: This plug-in provides the core capabilities of the EASy Producer instantiator concept. We will use parts of this plug-in in Section 3.2.2.

By default, Eclipse adds the package *org.osgi.framework* as *Imported Packages* because of the selected target platform in the *New Plug-in Project* wizard. However, this package is not required for the integration with EASy Producer and, thus, can be removed. Select the package on the right side of the *Dependencies* tab and click the *Remove* button. Then, click the *Add…* button and select *org.osgi.service.component* as *Imported Packages*. This package provides support for service components and their interaction with the context in which they are executed[[5]](#footnote-5). The *Dependencies* tab should now look like the one illustrated by Figure 2.



Figure 2: Definition of the required plug-ins for the new instantiator

In order to register the new plug-in to EASy Producer, the service component has to be declared. Thus, switch to the *MANIFEST.MF* tab in the plug-in manifest and add the following *Service-Component* declaration:

Service-Component: OSGI-INF/instantiator.xml

This Service-Component declaration specifies the location where to find the information about the service component, which shall be integrated into EASy Producer. The declared XML file will be defined in the next step. Figure 3 shows how the manifest file must look like.



Figure 3: Declaration of the service-component for the new instantiator

The definition of the service component requires the creation of a new folder within the plug-in project. Right click on the plug-in project and select *New 🡪 Folder*. The name of the folder has to be *OSGI-INF*. Then, create a new XML file within this folder. Right click on the folder and select *New 🡪 Other…* . In the emerging wizard, open the category *XML*[[6]](#footnote-6), select *XML File*, and click the *Next* button. Define the name of the file in accordance to the file declared in the manifest illustrated in Figure 3: *instantiator.xml*. Clicking the *Finish* button will open the XML editor. Switch to the source tab and edit the file as follows:

<?xml version="1.0" encoding="UTF-8"?>

<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0"

immediate="true"

name="EASy Example Instantiator">

<implementation class="easyexampleinstantiator.ExampleEngine"/>

<service>

<provide interface="de.uni\_hildesheim.sse.easy\_producer.

instantiator.InstantiatorEngine"/>

</service>

</scr:component>

Figure 4 shows the final XML file. Please note that we used the names and package-structure of our example in Figure 4. Thus, with respect to different implementations the name of the service component in line 4 as well as the package and the class name of the implementation class element in line 6 (the class, which will implement the instantiator) have to be adapted. Please ignore the warning in line 6 as the class currently does not exist. This will be part of Section 3.2.2.



Figure 4: Definition of the service-component for the new instantiator

The previously defined XML file must be included in the binary build. Thus, open the manifest file again and switch to the *Build* tab. In the left lower part of this tab select the *OSGI-INF* folder to be included in the binary build.

The last step is the inclusion of external, third-party libraries – the actual instantiator. Please note that this step is only required if the main implementation of the instantiator or required other required functionalities are implemented in another plug-in. In such a case, build the plug-in first[[7]](#footnote-7). Then, right click on the current instantiator plug-in project, select *New* and *Folder*. The name of the new folder must be *lib*. Include all libraries in this folder that are required by the new instantiator. The folder and the required libraries have to be included in the *Classpath* of the new plug-in. Thus, open the plug-in manifest and switch to the *Runtime* tab. Add the libraries to the *Classpath* by clicking on the *Add…* button on the right side of the *Runtime* tab. Select all required libraries of the *lib* folder and click the *Ok* button. Switch to the *Build* tab of the plug-in manifest and select the *lib* folder to be part of the *Binary Build* in the left lower part of this tab. Figure 5 and Figure 6 show the result in the context of our example. Figure 5 shows the included library de.uni\_hildesheim.sse\_0.0.1.jar, which provides the main functionality of our prototypical instantiator and, thus, has to be available at runtime. Figure 6 illustrates the build configuration in which the library (highlighted) is selected as part of the *Binary Build*.



Figure 5: Classpath specification of external, required libraries



Figure 6: Binary Build selection of external, required libraries

Finally, the plug-in project is set up, configured and ready to use. In the next section, we will further develop this plug-in by implementing instantiator-specific functionality based on the results of this section.

### Instantiator Implementation

In the previous section, we set up the Eclipse plug-in project for implementing a new instantiator for EASy Producer. In this section, we will describe how to implement the (basic) functionalities of an instantiator. However, as each instantiator provides its individual capabilities and is used for different purposes, this description will only include the basic functionalities that are common to each instantiator.

The first step is to create a new Java class file. Right click on the package that was defined as the implementation class package in Section 3.2.1 and select *New 🡪 Class*. In the emerging *Java Class* wizard, define the name of the new class in accordance to the name of the implementation class (cf. Section 3.2.1). In our example, we use the name *ExampleEngine*. Leave all other options as-is.

Each new instantiator has to extend the *InstantiatorEngine* class of the EASy Producer instantiator core package. Thus, the next step is to edit the new class file as follows (please note that we use the packages and class names of our example):

package easyexampleinstantiator;

import org.osgi.service.component.ComponentContext;

import de.uni\_hildesheim.sse.easy\_producer.instantiator.\*;

import de.uni\_hildesheim.sse.model.confModel.DecisionVariable;

public class ExampleEngine extends InstantiatorEngine{

protected void activate(ComponentContext context) {

Transformator.addEngine(this);

}

protected void deactivate(ComponentContext context) {

Transformator.removeEngine(this);

}

@Override

protected void addValue2Context(DecisionVariable arg0) {

}

@Override

protected void clearContext() {

}

@Override

protected void initEngine() {

}

@Override

public void instantiate() throws InstantiatorException {

}

}

We will now discuss each of these methods in detail:

* *activate*: This method is used to activate the instantiator plug-in. We recommend not changing this method in order to guarantee that EASy Producer activates the instantiator properly.
* *deactivate*: This method is used to deactivate the instantiator plug-in. We recommend not changing this method in order to guarantee that EASy Producer deactivates the instantiator properly.
* *addValue2Context*: This method will be invoked by EASy Producer in order to pass the all decision variable objects to the instantiator engine. The method is called for each decision variable in the current configuration, while the way of saving or processing this data is up to the engine. For example, define a local list of type *DecisionVariable*, which stores each passed variable in order to process it when the instantiation process is invoked.
* *clearContext*: This method will be invoked by EASy Producer in order to clear the current context of the instantiator engine. The context of an instantiator is all information that will be saved or used in the context of a specific product line project, i.e. the list of decision variables of the current variability model. EASy Producer treats every instantiator as a singleton and, thus, tries to clear the context of an instantiator each time it will be used. What has to be cleared and how is up to the developer. However, it is important to note, that clearing all project-specific information has to be cleared in order for the instantiation to work properly.
* *initEngine*: This method will be invoked by EASy Producer in order to initialize the instantiator engine. This method can be used to initialize any instantiator-specific attributes, objects, etc. Please note that this method also calls the *clearContext* to guarantee that initialization will be successful.
* *instantiate*: This method will be invoked by EASy Producer in case that the instantiation process has been started. Thus, this method includes the core functionalities of the instantiator, i.e. copying specific files or manipulating their content with respect to the values of the decision variables passed by the method *addValue2Context*. For this purpose, the method *getFilesToInstantiate* of the *Instantiator Engine* class can be used to get the list of files this instantiator was assigned to[[8]](#footnote-8).

Figure 7 shows a very simple example of how to use these methods to implement instantiator-specific functionalities. The prototypical *SEEInstantiator* will only display the files that have been assigned to it using EASy Producer[[9]](#footnote-9). When the instantiator is initialized (*initEngine*), a new instance of the *SSEInstantiator* is created (line 37). This method will also invoke the method *clearContext*, which will clear the private file list of this instantiator (line 32). The file list is used to store the files that have been assigned to the instantiator (line 42) and to pass these files to the actual *SSEInstantiator* to be displayed if instantiation is invoked (line 43). This will open a simple message dialog in EASy Producer that lists all assigned files of the *SSEInstantiator*.

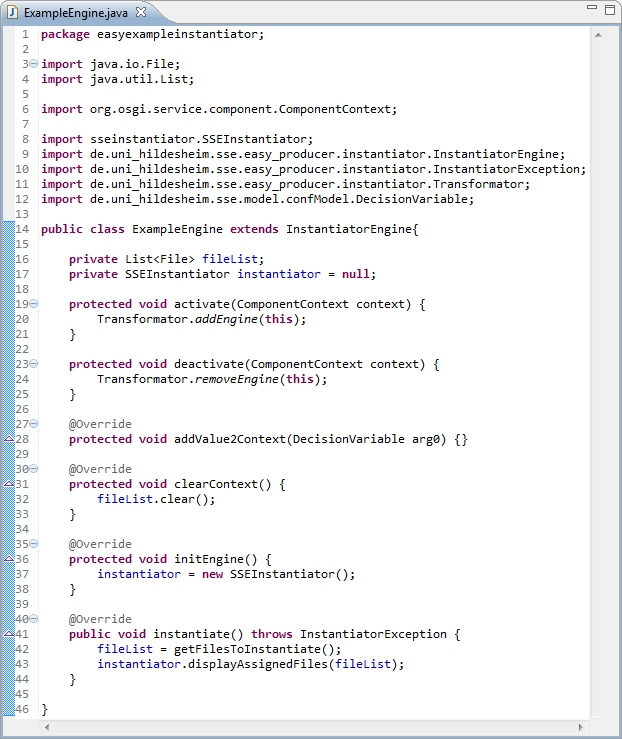


Figure 7: Implementation of instantiator engine functionalities

The last step is to build the plug-in. Open the plug-in manifest file and click on the *Export deployable plug-ins and fragments* button in the upper right corner. In the emerging wizard select the current plug-in project, specify the desired destination[[10]](#footnote-10), and click on the *Finish* button.

Finally, the plug-in and, thus, the instantiator are implemented, build, and ready for use. In the next section, we will describe how to integrate a new instantiator within EASy Producer. We will also have a quick look on how to use it. However, for detailed information on how to use an instantiator, please consider the [EASy Producer User Guide](http://projects.sse.uni-hildesheim.de/easy/docs/guide.pdf).

### Instantiator Integration

In the previous section, we implemented the (basic) functionalities of a new instantiator. Further, we build a deployable plug-in, which we will use in this section for integrating the new instantiator within an EASy Producer installation.

The first only step is to copy the previously build instantiator plug-in into the *dropins* folder of the Eclipse application in which EASy Producer installed. Start the Eclipse application and create a new product line project: *File 🡪 New 🡪 Other… 🡪 EASy-Producer 🡪 New EASy-Producer Project*. The name of the new project is up to the developer. The product line editor will emerge. Switch to the *Instantiator View* tab, and expand the drop-down menu to choose an instantiator. In this list the new instantiator will appear as illustrated by Figure 8. Add this instantiator by clicking on the *Add Instantiator* button and select the files of the product line project that should be instantiated by this instantiator.

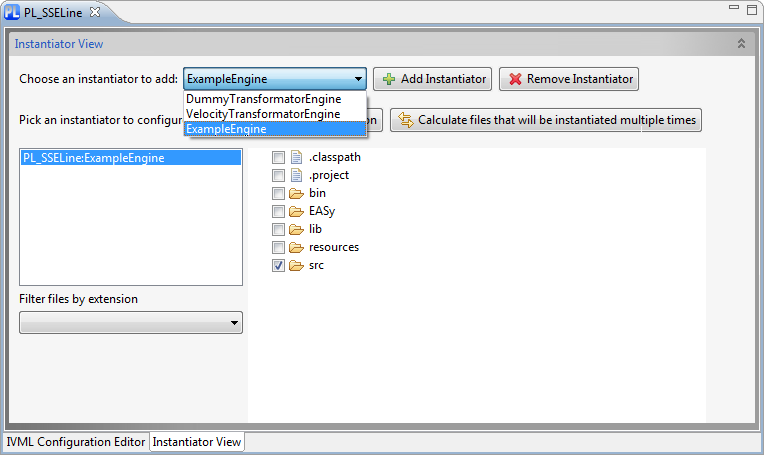


Figure 8: Using the new instantiator

## Implementing a New Reasoner

TODO

The basic mechanism is the same, thus, just write what is different and reference the existing sections as needed!!

# Known Bugs

None regarding the implementation of extensions.

1. <http://projects.sse.uni-hildesheim.de/easy/> [↑](#footnote-ref-1)
2. Detailed information regarding the Velocity instantiator can be found in the [EASy Producer User Guide](http://projects.sse.uni-hildesheim.de/easy/docs/guide.pdf). [↑](#footnote-ref-2)
3. For more information visit: <http://eclipse.org/equinox/> [↑](#footnote-ref-3)
4. For more information visit: [Eclipse API – org.eclipse.core.runtime](http://help.eclipse.org/helios/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Freference%2Fapi%2Forg%2Feclipse%2Fcore%2Fruntime%2Fpackage-summary.html) [↑](#footnote-ref-4)
5. For more information visit: [OSGi API – org.osgi.service.component](http://www.osgi.org/javadoc/r4v42/org/osgi/service/component/package-summary.html) [↑](#footnote-ref-5)
6. If the category *XML* does not exist, install XML support using *Help 🡪 Install New Software* or open the category *General*, select *File*, and define the name as well as the file-type manually. [↑](#footnote-ref-6)
7. If you do not know how to build a plug-in, please consider Section 3.2.2. [↑](#footnote-ref-7)
8. The instantiator core plug-in of EASy Producer provides a Java documentation including the full list of available methods of the *Instantiator Engine* class. [↑](#footnote-ref-8)
9. We include an external library that provides the *SSEInstantiator* indicated by the import statement in line 6 of Figure 7. How to include external libraries is described in Section 3.2.1. [↑](#footnote-ref-9)
10. The destination is up to the developer. However, we recommend using a location, which is easy to find as we will need the location for integrating the new instantiator in Section 3.2.3. [↑](#footnote-ref-10)