




Custom Domain Modeling with UML Profiles

The Basics of Generating Tooling for Elements from a UML Profile

Wayne Diu

February 13, 2009

INTRODUCTION	4
PREREQUISITES	4
MODELING IN A SPECIFIC DOMAIN	4
EXTENDING UML WITH PROFILES.....	5
WHAT'S NEW IN 7.5.....	5
BEFORE YOU START.....	6
THE PROFILE TOOLING WIZARD	8
1.1 STARTING THE PROFILE TOOLING WIZARD	8
1.2 CHOOSING A PROFILE	9
1.3 BASIC TOOLING OPTIONS	10
1.3.1 When the tooling model should not be generated	10
1.3.2 When the tooling code should not be generated	10
1.4 ADVANCED TOOLING OPTIONS.....	10
1.4.1 General Tab.....	10
1.4.2 Elements Tab.....	11
1.4.3 Shapes Tab.....	13
2 THE GENERATED PLUG-IN	14
3 THE TOOLING MODEL.....	15
CUSTOMIZING THE TOOLING	17
3.1 THE TOOLING MODEL.....	17
3.2 THE CLASS STEREOTYPED AS PROFILE - 	18
3.3 THE CLASS STEREOTYPED AS PATHMAP - 	18
3.4 CLASS STEREOTYPED AS ACTIVITY - 	18
3.5 ELEMENT TYPES PACKAGE.....	19
3.6 MENUS PACKAGE	19
3.7 PALETTES PACKAGE	21
3.8 SHAPES.....	22
3.8.1 Using default UML notation	22
3.8.2 Generating custom shapes	24
3.9 PROPERTIES	26
4 WIZARDS	27
5 DYNAMIC PROFILE TOOLING DIAGRAMS.....	28
6 CHANGE MANAGEMENT	29
6.1 RESPONDING TO CHANGES IN THE PROFILE	29
6.2 UPDATING TOOLING MODEL GENERATION PROPERTIES	30
6.3 GENERATING CODE FROM THE TOOLING MODEL	31
7 RUNNING THE GENERATED PLUG-IN	33

7.1	LAUNCHING A NEW INSTANCE OF THE RUNTIME WORKBENCH	33
7.2	THE NEW MODEL WIZARD	34
7.3	ENABLING ADDITIONAL CAPABILITIES	36
7.4	WORKING WITH EXISTING MODELS	37
8	DEPLOYING THE TOOLING	39
8.1	DEPLOYING TOOLING THE EASY WAY	39
8.2	DEPLOYING TOOLING AS A JAR FILE	39
8.2.1	Importing the JAR.....	40
8.2.2	Including the JAR in the plugins directory.....	40
8.3	DEPLOYING TOOLING USING AN UPDATE SITE.....	42
9	UPGRADING FROM EARLIER VERSIONS.....	44
10	CONCLUSION	45
11	ACKNOWLEDGEMENTS	46
12	ABOUT THE AUTHOR	46
13	RESOURCES	46

Introduction

This article guides you through the basics of generating tooling for elements from an existing UML profile or from a new UML profile. The steps described in this article pertain to Rational Software Architect for WebSphere Software 7.5, Rational Software Architect Standard Edition 7.5 and Rational Software Modeler 7.5.

Prerequisites

If you have a basic understanding of Eclipse and UML modeling, you should be able to generate and deploy your own custom tooling. To be able to customize the tooling, you will need a working knowledge of developing UML models. A high level understanding of Graphical Modeling Framework (GMF) is beneficial but not required. For advanced users who wish to perform major customizations, knowledge of Java™ technology, GMF, and Eclipse plug-in development is required.

Modeling in a specific domain

First, what is the point of Profile Tooling? It enables you to act as a *toolsmith*, a person who creates tools. Profile Tooling generates tooling for a particular domain, starting from an existing UML profile. By *tooling*, we are referring to a custom diagram editor that contains custom palette entries, menu items, wizard templates, property sheets, and even custom shapes. The generated tooling can be deployed to end users, known as *domain modelers*. Using the generated tooling, the domain modelers are able to model elements in their specific domain, using the notation specific to their domain.

Extending UML with profiles

Take, for example, an airline company with a system used by customer service representatives as a result of customer interactions. This system includes the ability to perform queries on flights and availabilities, create reservations, and – yes – even handle complaints about lost luggage or other matters (after being transferred to a separate representative, of course). A corresponding UML profile might contain various stereotypes with metaclass extensions to Actor for the various people involved (customer, booking agent, luggage agent), stereotypes with metaclass extensions to Use Case for various actions (standard reservations, redeeming loyalty points, lost luggage reports and queries), and stereotypes for other items in the system (e.g. queries, e-tickets, or complaints). The person responsible for modeling such a system may not be familiar with UML. However, they are familiar with the concepts the system entails, and assuming the profile is reasonably complete, these concepts are represented by the profile elements.

You will learn how to use the new Profile Tooling workflow introduced in Rational Software Architect and Rational Software Modeler 7.5 and later, including:

- generating the tooling model;
- how to customize the tooling model;
- creating dynamic tooling diagrams;
- generating code from the tooling model;
- deploying the tooling;
- updating the tooling model to respond to changes in the profile; and
- upgrading from previous versions.

Because Java code will be generated, you will be able to modify the code to customize various aspects, such as custom shapes. The generated code is consistent with standard GMF and UML Modeler API conventions. For more details on customization of the generated code, see the [Resources](#) section for links to GMF and Modeler API documentation.

What's new in 7.5

The workflow for Profile Tooling has been updated, from the way the tooling project is generated to the way the tooling model is customized. Here are some of the highlights:

- the Profile Tooling Wizard is now invoked through the Eclipse New Project wizards;
- a UML model is generated instead of GMF models;
- the code generation parameters are customized by editing the UML model;
- updates to the model to respond to changes are now handled through visual merge;
- additional code is generated to support customizing properties and wizards; and
- various other code generation settings have been introduced to allow more flexibility.

Before you start

Before beginning to work with Profile Tooling, be sure that you have installed the Modeling Extensibility option using the Installation Manager.

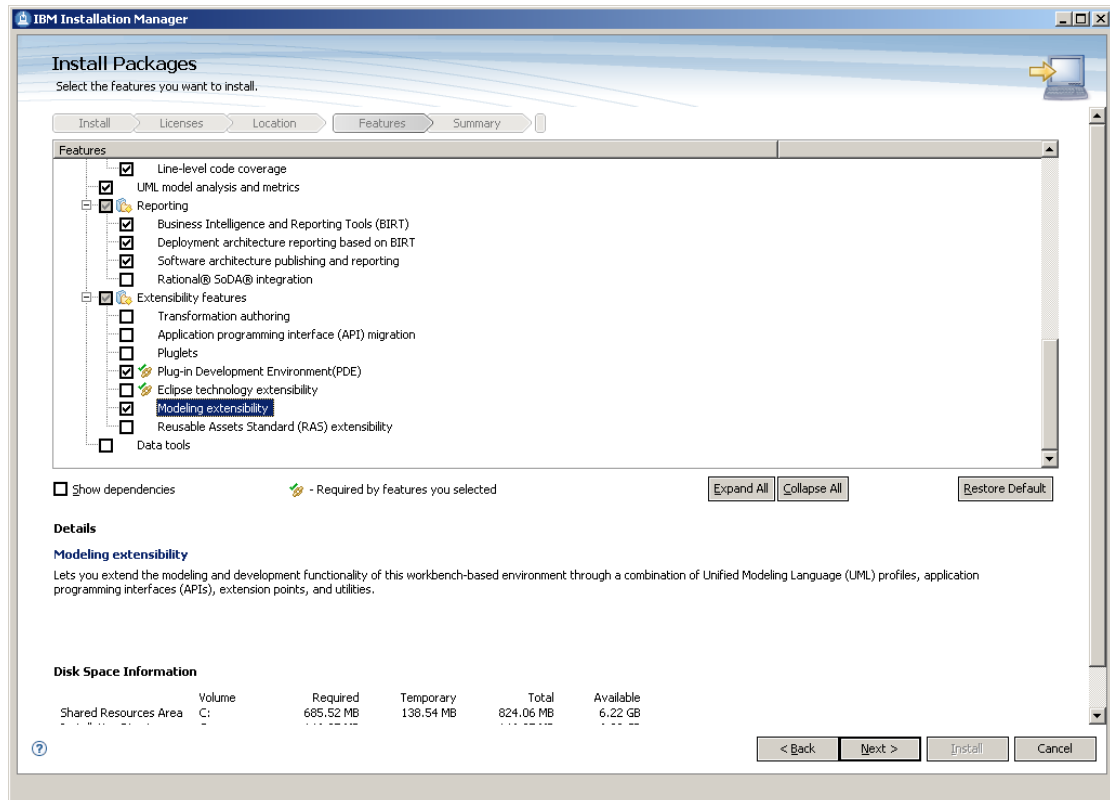


Figure. Installing the Modeling Extensibility option.

If your profile contains non-ASCII characters, or if you anticipate that you will be using non-ASCII characters anywhere in the tooling, be sure that the workspace preferences are set to UTF-8 encoding. You can do this by selecting **Window, Preferences, General, Workspace**, and choosing **UTF-8** as the **Text file encoding** (see Figure). Otherwise, the Java files that are generated will not compile properly.

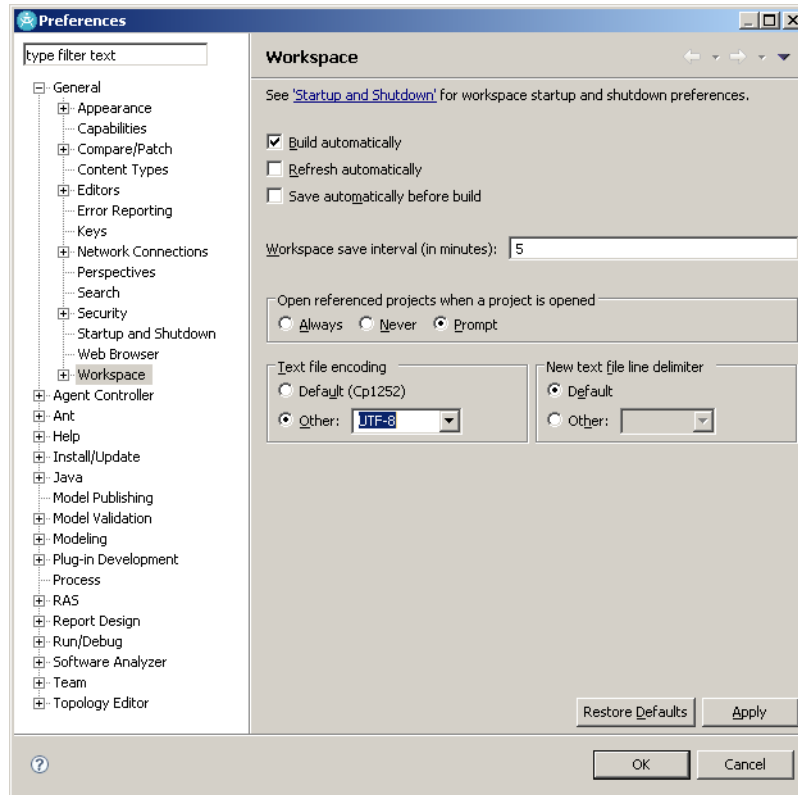


Figure. Setting the text file encoding.

The Profile Tooling Wizard

Profile Tooling was first introduced in version 7.0.5 of the product. Those familiar with the earlier release will recall starting off with a UML profile, right clicking on it, and choosing Generate Profile Tooling.... After running through the wizard, your tooling plug-in would be generated. In version 7.5, the starting point is from the Eclipse New wizards.

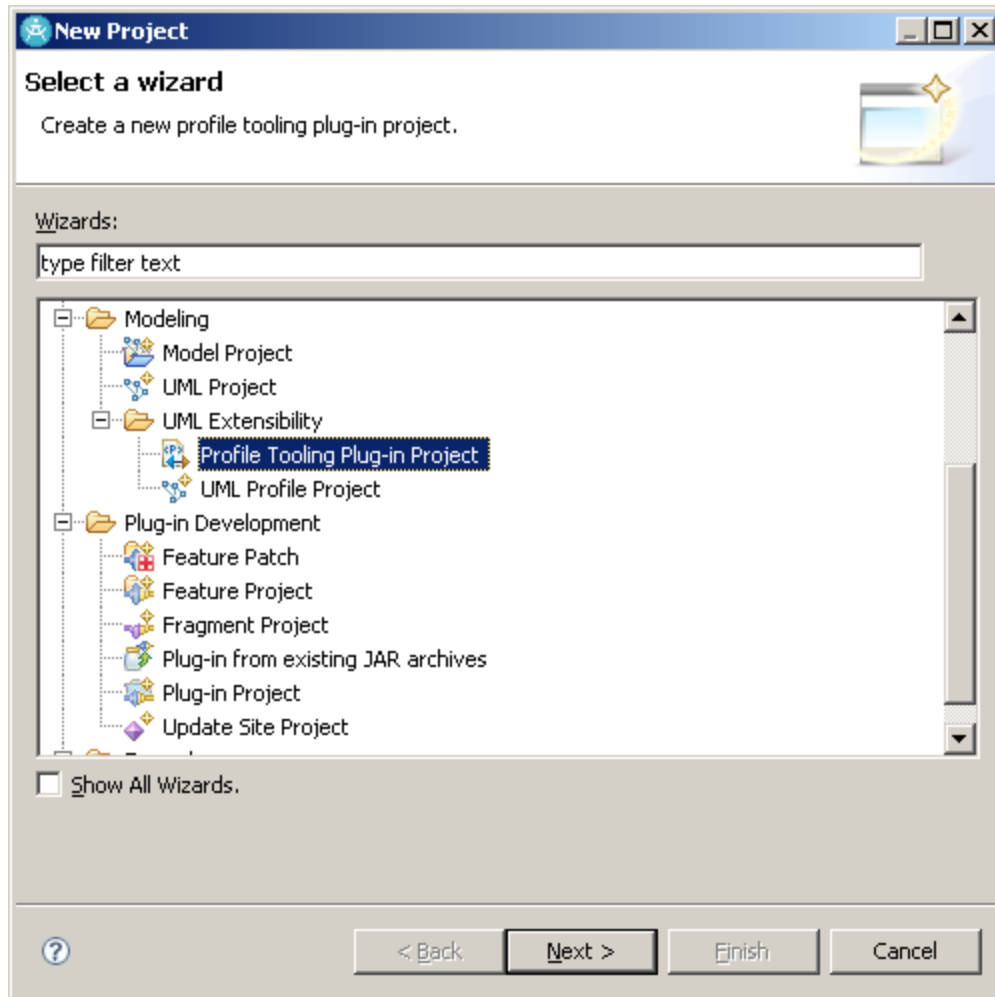


Figure. Invoking the Profile Tooling Wizard from the Eclipse New wizards.

1.1 Starting the Profile Tooling Wizard

Here's how to start using the Profile Tooling Wizard.

1. Choose **File, New, Project...**
2. Expand the **Modeling** category and expand the **UML Extensibility** subcategory.
3. From there, choose **Profile Tooling Plug-in Project** and press **Next**.
4. Give your project a meaningful name and press **Next**.
5. Review the contents of the subsequent **Plug-in Content** wizard page (there is no need to change anything), and press **Next** again.

- Now from the list of available templates, select **Profile Tooling Plug-in Project**. The **Next** button will become enabled. Press **Next** again, and now we are able to configure the options specific to profile tooling.

Tip: Alternatively, you may choose **Plug-in Project** from the **Plug-in Development** category, then go through all of the steps and pick **Profile Tooling Plug-in Project** at the last step.

1.2 Choosing a profile

If you have used a previous version of the Profile Tooling Wizard, one of the first things you'll notice is that you are no longer limited to choosing a profile from the workspace. In 7.5, if you press the **Browse...** button, you will find that in addition to being able to select a profile from the workspace, you can use a deployed profile or browse to an .epx profile outside of the workspace. Alternatively, you can even create a new profile by pressing the **New...** button.

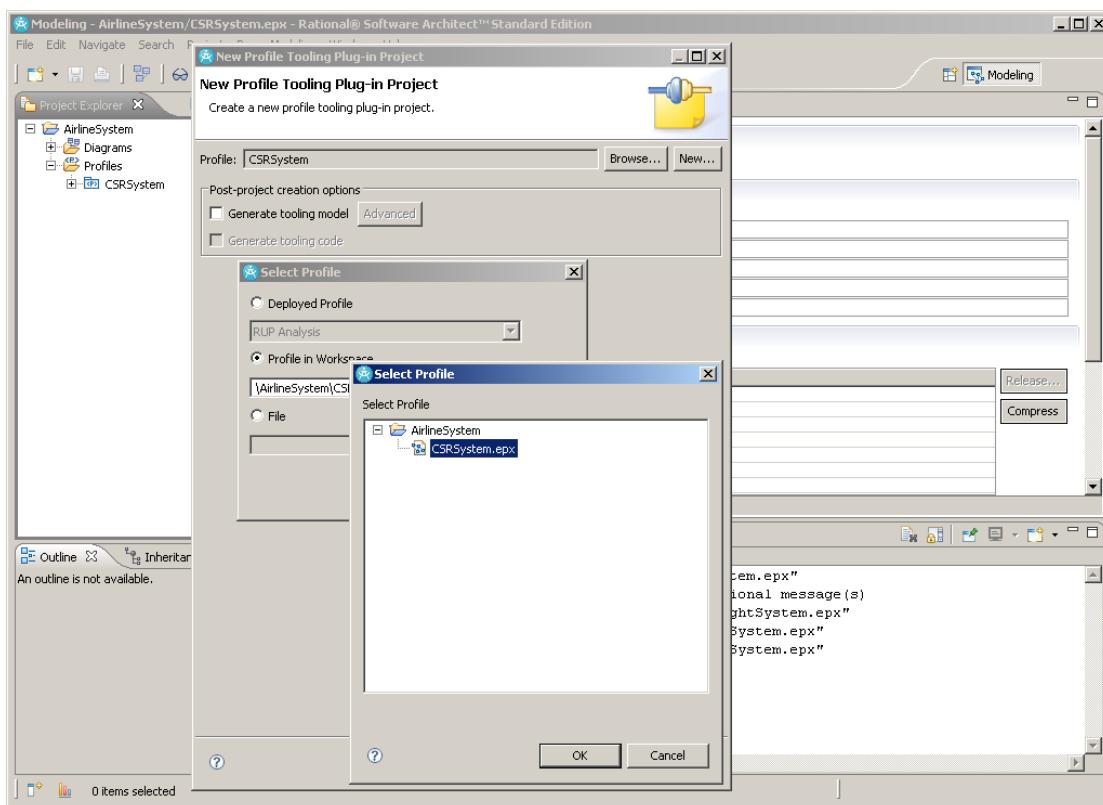


Figure. Selecting a profile.

Note: Although the Profile Tooling Wizard supports generating tooling from unreleased profiles, you will receive a warning. If you are generating tooling for your own model and, especially, if it will be deployed to other domain modelers, you are strongly advised to release your profile first. Compatibility between unreleased profiles is not guaranteed. To release your profile: Bring up the context menu of the profile in the **Project Explorer** and choose **Release**. The profiles shipped with the Rational modeling products have been released. If you do not have a well-developed profile available, you can follow along by choosing one of the deployed profiles in the dialog.

1.3 Basic tooling options

The **Post-project creation options** group contains the following two options: **Generate tooling model** and **Generate tooling code**.

- **Generate a tooling model** generates a UML model, that is an emx file, which describes how the code will be generated. We will learn how to customize this model later.
- **Generate tooling code** actually generates the code based on the parameters specified in the tooling model. Thus, if the option to generate the tooling model is not checked, this option will logically not be available.

The above options allow you to choose what the generated plug-in will contain. In certain scenarios, you do not need to generate the tooling model and the tooling code.

1.3.1 When the tooling model should not be generated

If you choose an existing profile in your workspace or in the file system, you have the option of not generating the tooling model. This is useful if you simply want to deploy your profile in a plug-in. Without this wizard, you would have to create the plug-in and manually copy the profile into the new plug-in. Then you would have to add the dependencies in the plug-in manifest and manually write the code to register the profile. With this wizard, it becomes a lot simpler. You can run through the wizard, select the profile to deploy, and all of the necessary steps will be done automatically.

1.3.2 When the tooling code should not be generated

Since code is generated based on the parameters specified in the tooling model, you may wish to tweak the model first before generating the code. This is why you may not want to immediately generate tooling code.

1.4 Advanced tooling options

Selecting the **Advanced** button in the **New Profile Tooling Plug-in Project** wizard brings up the **Advanced Tooling Model Generation Properties** dialog. From here, you can specify the type of tooling to be generated, which elements are to be considered from the profile, and whether custom shapes should be generated.

1.4.1 General Tab

The general tab allows you to define the type of tooling to be generated.

- **Menus and Palettes:** Selecting these options will create custom contributions to add items corresponding to your profile elements to the **Add** submenu and to the diagram palette. The **Add** submenu appears alongside the usual location of the **Add UML** submenu in the context menu of the diagram surface.
- **Properties:** Custom property sheets are shown in the **Properties** view for supported elements.
- **Wizards:** Adds custom templates to the **New Model Wizard**.

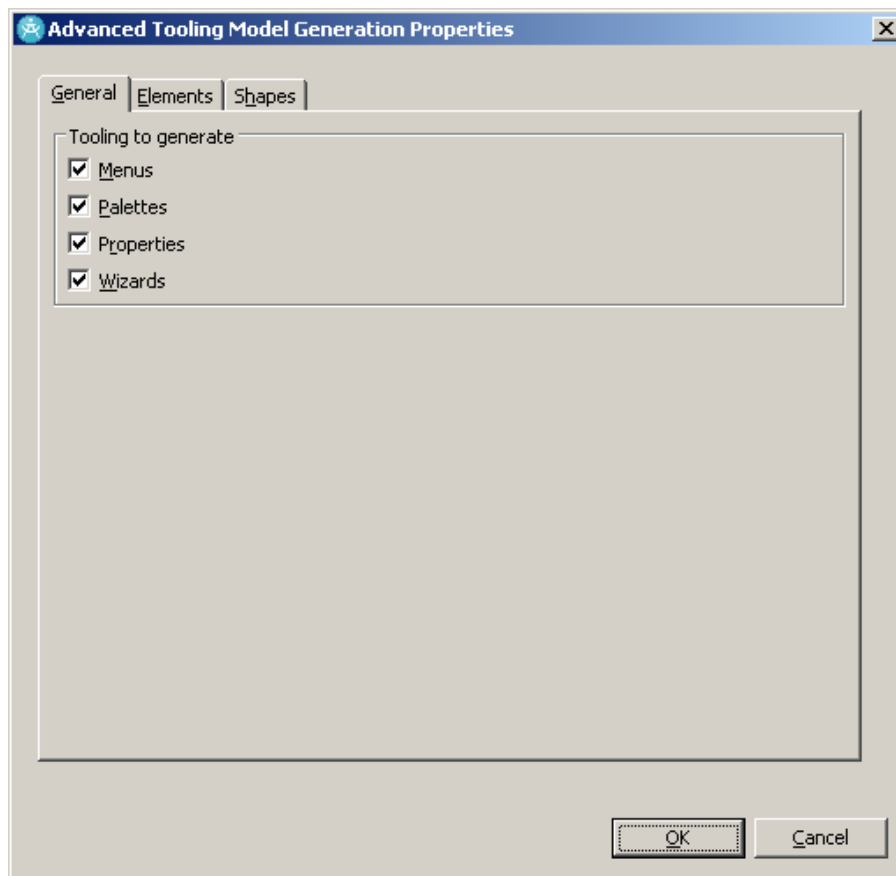


Figure. The **General** tab.

1.4.2 Elements Tab

The elements in this list correspond to the Stereotypes, Stereotype Associations, and Metaclass Associations defined in the selected profile. If you check the checkbox next to an element, the corresponding tooling specified in the **General** tab will be generated.

- The table contains one element per stereotype association and one element per metaclass association. The elements corresponding to stereotypes are added to the table subject to these rules:
- An element corresponding to the stereotype will not be in the table if a metaclass extension is not defined for it;
- If the metaclass extension is to a concrete subtype, one element will be added. For example, if a stereotype named Person extends the UML Actor metaclass, there will be one element in the list corresponding to it (<Person> Actor); AND
- One element per concrete subtype of the metaclass extension will be added. For example, if a stereotype named Requires extends the UML Dependency metaclass, in addition to an element for <Requires> Dependency, all its concrete subtypes will appear in the table: <Requires> Realization, <Requires> Substitution, <Requires> Manifestation, <Requires> Usage, etc.

- An element corresponding to the stereotype will not appear in the table if a metaclass extension to an abstract metaclass is defined for it. However, an element for each concrete subclass of the metaclass extension's metaclass will be added. For example, an element corresponding to the Classifier metaclass will not appear in the table. Even outside the realm of profile tooling it does not make sense to create Classifier on the diagram surface, although it would be valid to create Class or Interface, which are concrete subtypes of the abstract metaclass Classifier. In this case, the table shows all the concrete subclasses of Classifier – those are the classes where it does make sense to generate tooling.

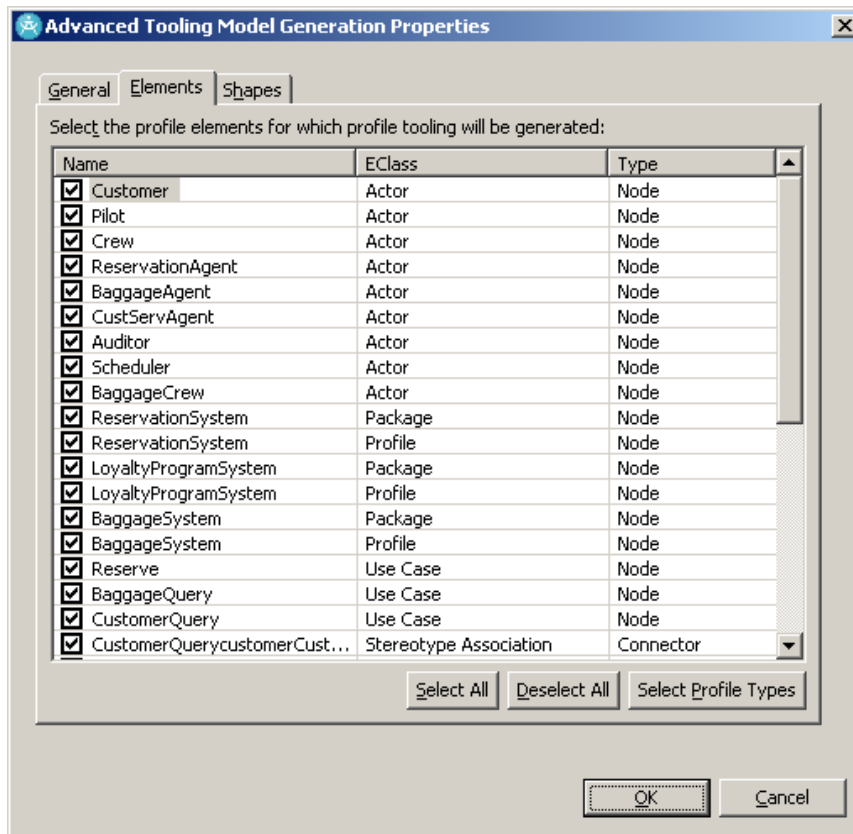


Figure. The **Elements** tab.

Select All and **Deselect All** are obvious, but **Select Profile Types** requires some explanation. **Select Profile Types** selects elements that directly correspond to the metaclass extensions of the stereotypes in the profile. Elements corresponding to subtypes added by the wizard (but not part of the original profile) are not selected.

As an example, suppose your profile contained three stereotypes, ToActor, ToUseCase, and ToClassifier. The first one has a metaclass extension of Actor, the second one has a metaclass extension to Use Case, and the third has a metaclass extension to Classifier. According to the UML spec, Classifier is an abstract type – **Select Profile Types** will select the Actor stereotyped as ToActor and the UseCase stereotyped as ToUseCase, as those were explicitly defined in the profile. It would not select the concrete subtypes of Classifier displayed in the wizard.

1.4.3 Shapes Tab

The elements in this table correspond to the checked elements in the **Elements** tab, with two exceptions which will be explained shortly. All the checkboxes are unchecked by default, and this is for a good reason.

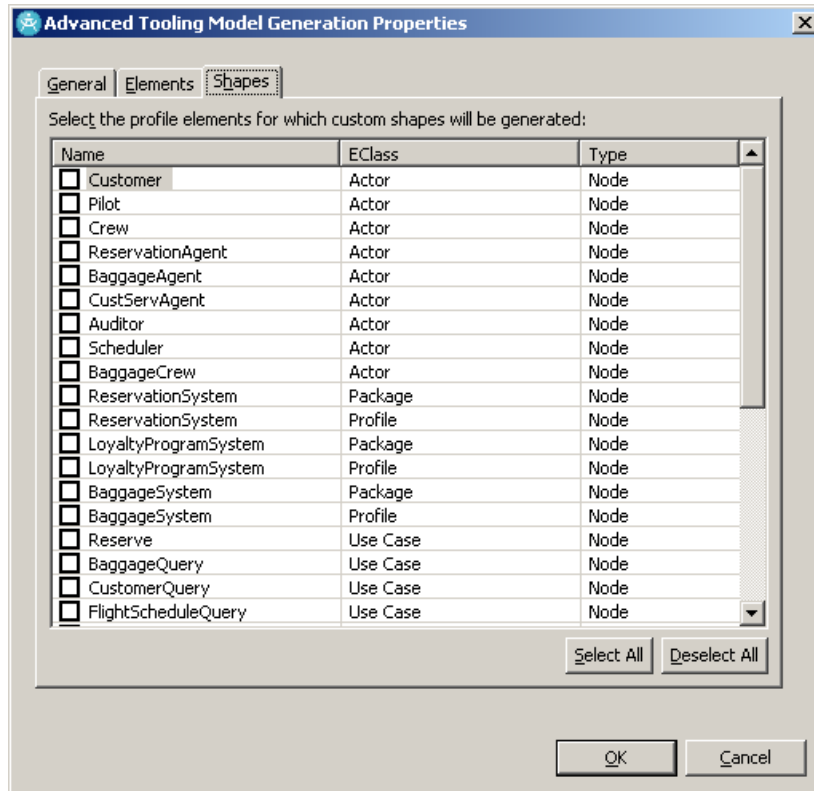


Figure. The **Shapes** tab.

When this is the case, the default UML shape notation is used: the reason is that shape customization is for advanced users. It is not possible to reasonably guess what type of custom shape is desired. Therefore, when generating custom shapes for non-relationship edit parts, a simple edit part, figure, and view in the shape of a standard rectangle is generated. For relationship edit parts, code for a simple connector and label are generated. It is up to the toolsmith to add compartments to it or perform additional customization.

Most of the time, the default edit part will meet the requirements, because the default edit part can easily display a custom graphic, such as the stereotype icon. But if you want custom compartments or other special shapes, you can check the checkboxes to generate the custom edit parts. You can then edit the generated code later on.

Stereotype associations and metaclass associations are never displayed in this table, even when the corresponding element has been checked in the **Elements** tab. A custom shape is generated automatically if the corresponding element has been checked in the **Elements** tab.

2 The generated plug-in

In version 7.0.5, the result of completing the wizard was your profile would be copied over, the source code would be generated if you checked the option, and a couple of models based on GMF code generation models would be generated. In 7.5, the **Generate a tooling model option** will generate a tooling model in the new plug-in. Rather than GMF tooling-based models being generated, this new version generates a single standard UML model with the emx extension. The DSLToolProfile – that is, the Profile Tooling Profile – has been applied to it.

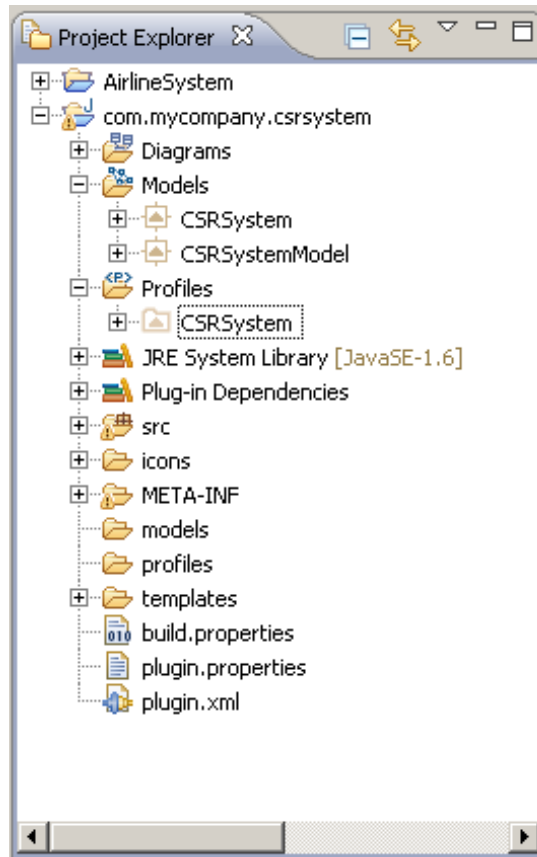


Figure. The two models in the **Project Explorer**.

If you look at the newly created project in the **Project Explorer**, you will see two models. The model residing in the project's models folder is the tooling model, while the model in the templates folder is for a template. The tooling model contains the settings that will describe how the plug-in will be generated. The template model will be used by the **New Model Wizard** when the tooling is deployed. So for example, if you want your template to have a particular capability, you would open that model, set the capability, and save it. Alternatively, if you wanted the user's model to contain a few default elements, you could add them to the template model and save it.

Tip: You can differentiate between the two models immediately because the template model has the suffix Model (e.g. CSRSystem**Model**) and the profile tooling model does not (e.g. CSRSystem).

3 The tooling model

To get an overview of the tooling that would be generated, you can either expand the tooling model in the **Project Explorer** and have a look at the elements in the model or see a visual representation through the use of diagrams. Because the generated tooling model is a standard UML model, you can add diagrams to it. To see a visual representation of the types of tooling that will be generated, perform the following steps:

1. Right click on the tooling model from the **Project Explorer**.
2. From the context menu, choose **Add Diagram, Freeform Diagram**.
3. Drag and drop the tooling model from the **Project Explorer** onto the diagram surface.
4. Right click on the model in the diagram surface.
5. From the context menu, choose **Filters, Show Related Elements...**
6. If necessary, press the Details button to expand the dialog.
7. Choose to show the owned elements of the model.
8. Press the **OK** button.

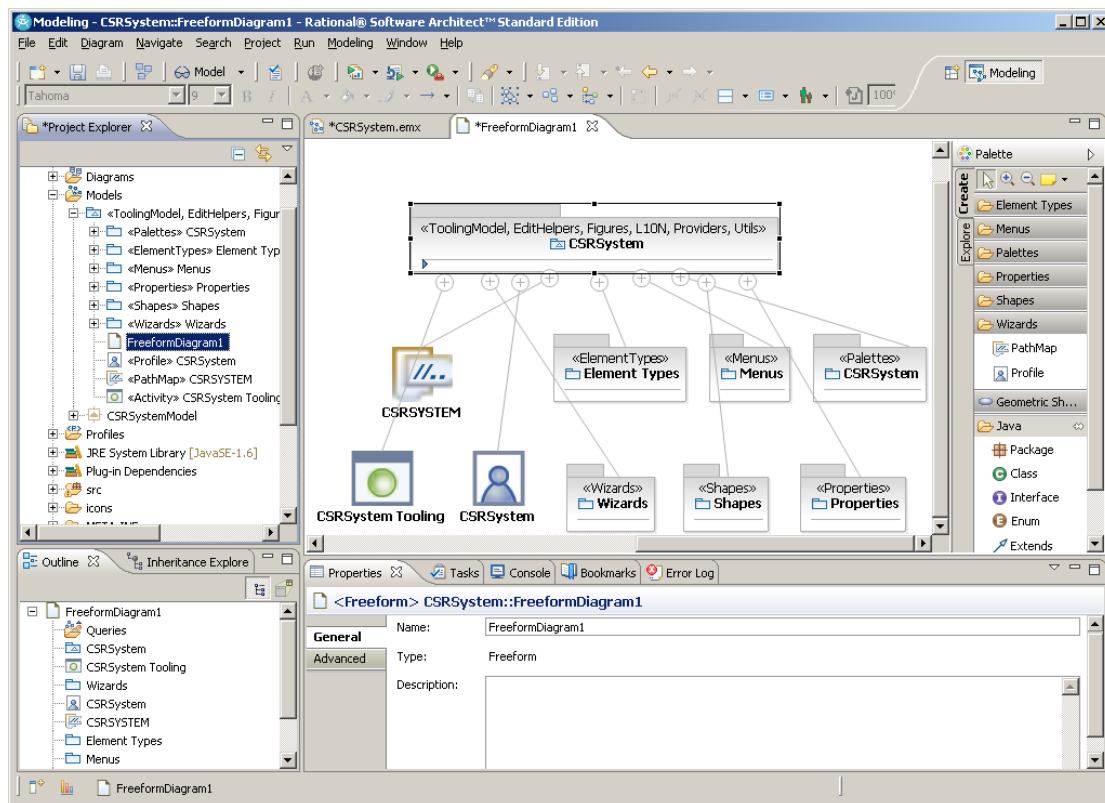


Figure. The typical results of Show Related Elements for a tooling model.

Let's go through the elements of the diagram. In addition to the main tooling model, you will see some stereotyped classes and some packages. A separate package is generated for each type of tooling. In general, settings are adjusted in two ways:

1. by renaming the name of the element;
2. by modifying the properties of the element from the **Properties** view.

The tooling model's properties contain wide-ranging settings that affect various aspects of your tooling. The stereotyped classes are used to control aspects of the plugin.xml that will be generated, while the packages loosely correspond to actual Java code that will be generated.

Customizing the tooling

3.1 The tooling model

The tooling model includes properties such as the Java class names for the providers, the details about your profile, the plug-in project name, and the application name.

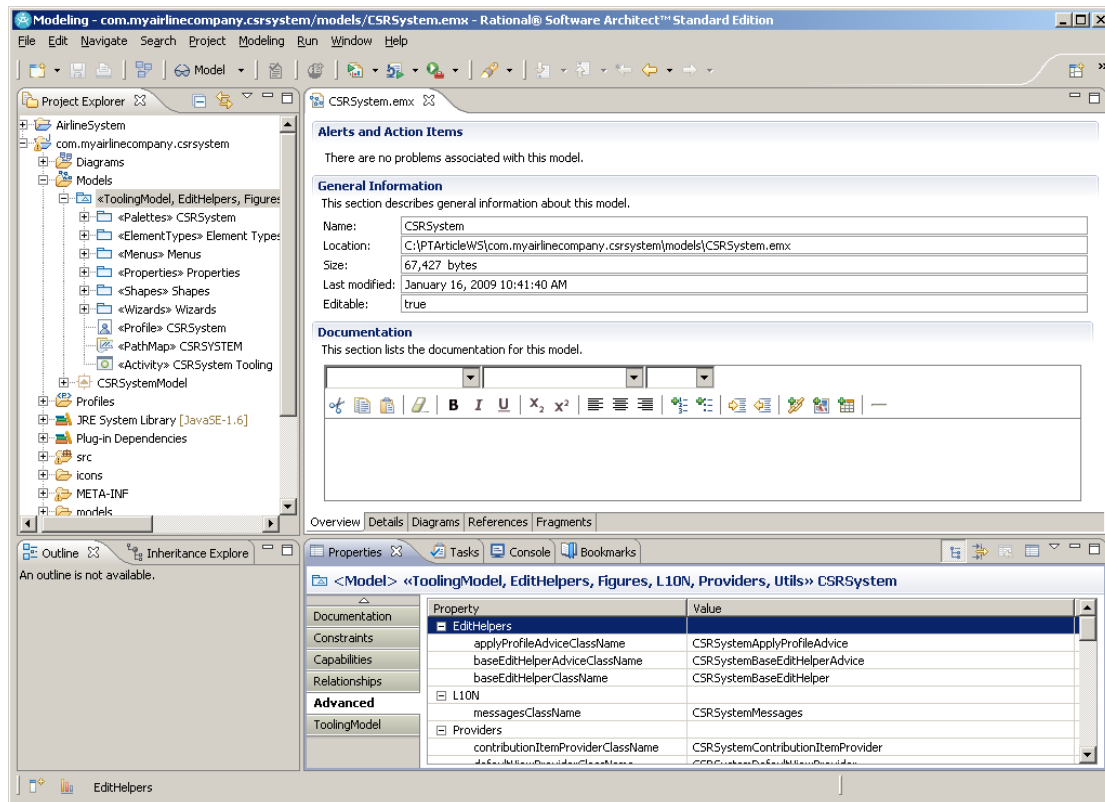


Figure. Showing the properties of the tooling model.

If you are familiar with GMF, you will find that the properties used to define the names of the generated classes are self-explanatory. For more information, please refer to the [Resources](#) section for GMF examples.

Here are some properties which may require clarification.

applicationName: This is typically the name of the domain for which you are generating tooling. By default, it is the same as your profile's name. It is a good idea to give your profile a meaningful name before generating the tooling!

deploymentFolder: If you are not using a deployed profile, this is the folder where your profile has been copied to. By default, the value is profiles. Although it is possible to change the value, it is general convention to leave the deployed profile in a folder called profiles. This value has no meaning when you are generating tooling for a deployed profile.

rsmVersion: This is the version of Rational Software Modeler (or Rational Software Architect) that you are targeting: that is, the version of the software in which you intend your tooling to be deployed.

Tip: If you have generated tooling and verified that it works, yet the domain modeler asserts that it does not work for them, please verify the `rsmVersion` property. Different code is generated based on what the `rsmVersion` is.

3.2 The Class stereotyped as Profile -

If you are deploying the profile in your plugin, this class controls settings for the profile for which tooling is to be generated. When profiles are registered as deployed profiles, you can specify its id (a unique identifier), whether or not the profile is required, and whether or not it is visible in the UI. If a profile is required, all models created using the application's UI will have this profile applied automatically. If a profile is visible, it will be shown to the user in the application UI, such in the dialog where you chose the profile to generate tooling. The profile tooling wizard will use default settings: that is, the profile is not required to be applied and is visible in the UI.

Note: It is strongly recommended that the required flag is left at its default value of false. Unless there is a very good reason, a profile should not be applied by default to every model that is created!

To modify these settings, show the **Properties** view for the element and activate the **Profile** property tab.

If you are using a using a deployed profile, then this setting has no meaning (because the generated plug-in will not contain duplicate XML to redefine the already-deployed profile). In that case, you will have to accept with the original settings used to deploy the profile.

3.3 The Class stereotyped as Pathmap -

The recommended way of deploying a profile is by using a pathmap. If you are deploying a profile in your generated plug-in, it controls the directory to your profile. In the same manner as the settings for the class stereotyped as Profile, if you are generating tooling for a profile that is already deployed, there is no need to modify this.

To modify this directory location, show the **Properties** view for the element and activate the **PathMap** property tab.

3.4 Class stereotyped as Activity -

Activities are used to control UI reduction – that is, minimizing irrelevant UI from the domain modeler's environment. For example, you may not want the New Model Wizard to show your model template or you may not want the palette to contain drawers and tools for your domain unless a certain activity is enabled. Activities are bound to a unique identifier. Normally, you can leave this identifier at its default value, unless you wish to change it so it corresponds to an activity

ID used elsewhere. You are also able to set a textual description of the activity ID.

To modify these settings, show the **Properties** view for the element and activate the **Activity** property tab.

3.5 Element Types package

This package contains classes that correspond to the elements that you have chosen to generate tooling for.

The classes in the **Element Types** package may be stereotyped as one of the following:

- **StereotypeSpecializationElementType:** These represent stereotyped elements with a metaclass extension to a metaclass that is not a relationship.
- **StereotypeLinkSpecializationElementType:** These represent stereotyped elements with a metaclass extension to a metaclass that is a relationship.
- **LinkSpecializationElementType:** These represent elements derived from stereotype associations.
- **MetaclassLinkSpecializationElementType:** These represent elements derived from metaclass associations.

3.6 Menus package

This package contains classes used to define the elements that can be created using the context menu. There must be a corresponding element defined in the **Element Types** package. It is only possible to create elements that are not relationships from a context menu (i.e. the corresponding element must be stereotyped as `StereotypeSpecializationElementType`).

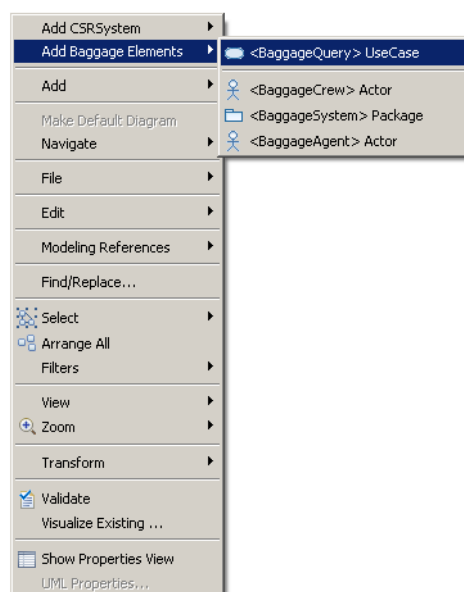


Figure. The **Add** context menu.

The figure above illustrates how the context menus may be customized. If no customizations were performed, all the menu items would appear in the same **Add CSRSys** flyout menu (submenu). To make the menu less cluttered, certain menu actions have been separated into an **Add Baggage Elements** flyout menu, and the **<BaggageQuery> UseCase** action has been separated into its own group.

The classes in the **Menus** package may be stereotyped as one of the following:

- **ContextMenu:** This is the root class for the popup menu definitions. This can be left alone.
- **FlyoutMenu:** This defines a submenu for adding the elements from your profile's domain to the model, much like the standard **Add UML** submenu available from the context menu. A typical customization is changing the name of the submenu. To do that, simply change the name of the class in the **Properties** view or rename the class from the **Project Explorer**. This name will be added to a properties file in order to support localization when the code is generated.
- **MenuGroup:** This is used to group the elements in the FlyoutMenu submenu. Elements in different groups are divided by a separator. By default, all elements for which you have chosen to generate tooling will be added into a single menu group with a default id of defaultGroup. If you wish to group the elements logically, you can do so by creating additional groups. For example, you may want to group all the actions used to create non-relationship domain elements together and have them separate from the actions used to create relationship domain elements.
- **MenuCreationAction:** This creates the specified element type. The element type should be defined in the **Element Types** package discussed above.

Several of the elements pertaining to the menu and the palette support custom icons. However, it is not necessary to specify an icon for an icon to actually appear. If an icon is not specified, the stereotype icon will be used if it is defined. If a stereotype icon is not defined, the corresponding UML metaclass' icon will be used instead.

To summarize:

- a class stereotyped as ContextMenu references classes stereotyped as FlyoutMenu;
- a FlyoutMenu references classes stereotyped as MenuGroup;
- a class stereotyped as MenuGroup references classes stereotyped as MenuCreationAction and MenuSeparator.

Tip: In the summary above, observe the word "references." This is **not** the same as containment! Simply adding a class stereotyped as MenuCreationAction along the other ones inside the **Actions** package is not sufficient. You will need to access the properties of the class stereotyped as MenuGroup, and then add the new action in the children properties. Moreover, if you choose to copy and paste, please be sure to change the id property of your item! Multiple items having the same identifier will confuse the code generator.

3.7 Palettes package

This package contains classes used to define the elements that can be created using the diagram palette. In version 7.5 of the product, if you have chosen to generate tooling for both menu items and palette entries, the palette will contain entries for relationship elements in addition to all the menu actions generated for the context menu (non-relationship elements). There must be a corresponding element defined in the **Element Types** package.

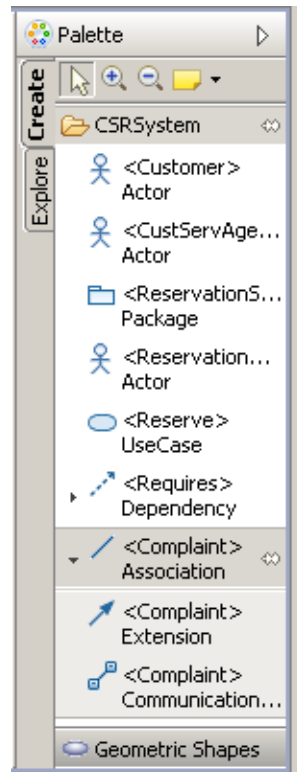


Figure. The diagram palette.

The figure above gives an example of stacks in the palette. The **<Requires> Dependency** element and the **<Complaint> Association** element are in stacks.

The classes in the **Palette** package may be stereotyped as one of the following:

- **Palette:** A palette can contain multiple drawers. For example, the tooling may be grouped by actions performed by different types of customer service representatives. To add an additional group:
 1. Create a new class stereotyped as **PaletteDrawer**.
 2. Show the **Properties** view for the class stereotyped as **Palette** and activate the **Palette** property tab.
 3. Locate the children property and add the new class stereotyped as **PaletteDrawer**.
- **PaletteDrawer:** By default, all items are added into the same palette drawer, and this palette drawer's title is set to the profile's name. To rename a palette drawer, simply change the name of the class in the **Properties** view or rename it from the **Project Explorer**. The palette drawer's title will be added to a .properties file in order to support

localization when the code is generated. The **PaletteDrawer** tab in the **Properties** view also allows you to adjust other settings, including the initial state of the palette drawer (INITIAL_STATE_CLOSED, INITIAL_STATE_OPEN, and INITIAL_STATE_PINNED_OPEN).

- **PaletteCreationToolEntry:** Each class stereotyped as a **PaletteCreationToolEntry** corresponds to an item in the palette drawer in your generated tooling (assuming it is contained by a class stereotyped as **PaletteDrawer**). In a similar way as the **MenuCreationAction**, you can choose the element type to be created by the tool. The element type should be defined in the **ElementTypes** package discussed above.
- **PaletteStack.** A palette stack groups multiple palette tools together so they occupy less space on the palette. To avoid confusing the end user of the tooling, stacked palette tools should be related in some logical manner. Recall that when generating the tooling, the **Elements** tab in the **Advanced Tooling Model Generation Properties** dialog of the **Profile Tooling Plug-in Project** wizard added concrete subtypes. By default, a stack groups the elements created off the concrete subtypes along with the main type defined in the profile (assuming it is concrete). If the intention is that certain domain elements will be created frequently, you can “unstack” them by removing elements from a stack and adding them directly to the palette drawer. To “unstack” an element:
 1. Show the **Properties** view for the class stereotyped as **PaletteStack** and activate the **PaletteStack** property tab.
 2. Locate the children property and remove the class stereotyped as **PaletteCreationToolEntry**.
 3. Show the **Properties** view for the class stereotyped as **PaletteDrawer** and activate the **PaletteDrawer** property tab.
 4. Locate the children property and add the class you removed in Step 2.

To summarize:

- a class stereotyped as **Palette** references classes stereotyped as **PaletteDrawer**;
- a class stereotyped as **PaletteDrawer** references classes stereotyped as **PaletteCreationToolEntry** or **PaletteStack**;
- a class stereotyped as **PaletteStack** references classes stereotyped as **PaletteCreationToolEntry**.

3.8 Shapes

A **Shapes** package is generated regardless of whether you decide to create custom shapes or reuse the default UML shapes. That is because we create classes to represent default edit parts which can be customized by the person generating the tooling. For more information on programmatically customizing the diagram shapes, please see “Product help – Customizing Diagram Shapes” in the [Resources](#) section.

3.8.1 Using default UML notation

By default, the default UML notation is used. If you have not changed these settings, your tooling model will contain classes stereotyped as **DefaultEditPart**, **Style**, and **StyleFeatureValue**. And, when the code is generated, only Java

classes that implement `IViewCustomizer` are generated in the `viewFactories` package.

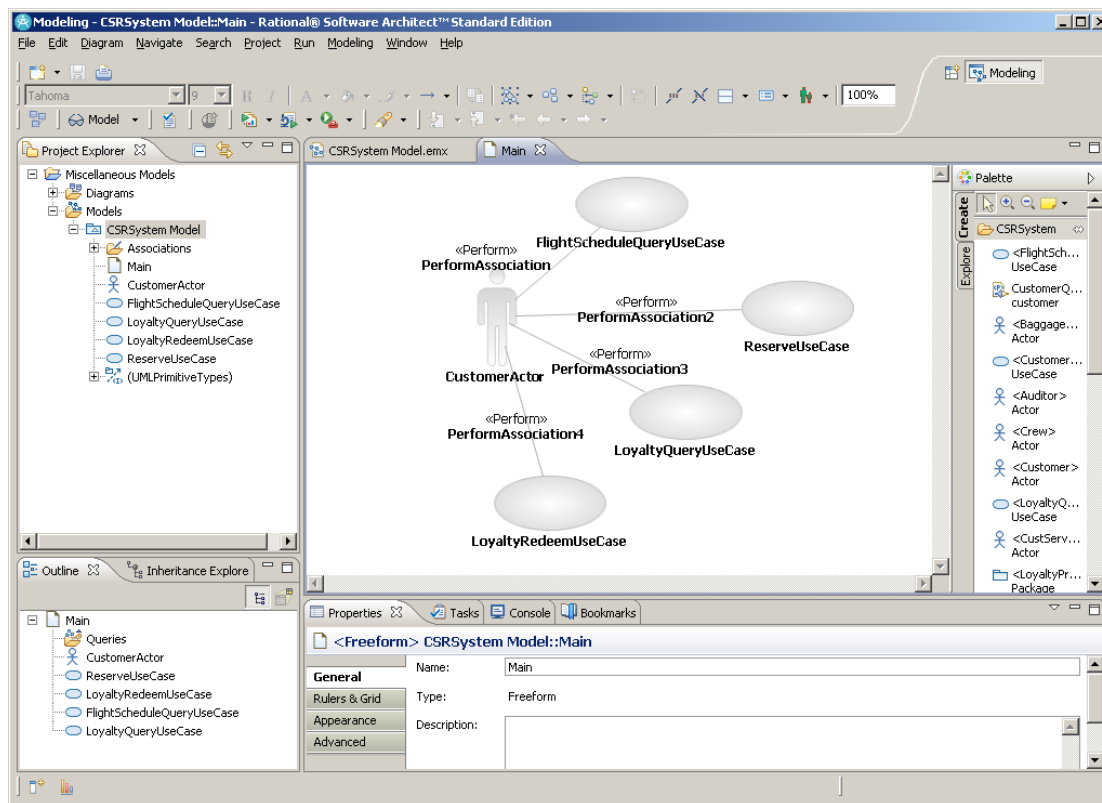


Figure. The shapes in this diagram use the default UML notation.

Note: The customizations made by the view customizer will override the defaults specified in **Windows, Preferences, Modeling, Appearance, Shapes**. You can, however set them back using the **Appearance** tab in the **Properties** view.

When using default UML notation, the classes in the **Shapes** package may be stereotyped as one of the following:

- **DefaultEditPart:** As in the `MenuCreationAction` or `PaletteCreationToolEntry`, the `DefaultEditPart` is associated with an element type. In addition to specifying the element type corresponding to this edit part, you can set the style or the view customizer's Java class name.

A view customizer is a concept introduced by Profile Tooling. It is used to perform customization on views after they have been created. The view customizer (`IViewCustomizer`) is defined in the generated view provider. View customizers are invoked by the view provider to customize the view in the `createNode()` and `createEdge()` methods after the GMF `ViewService` has returned a `Node` or an `Edge`.

- **Style:** The style to be modified. By default, the style class is `UMLShapeStyle` and it brings together the GMF styles `FontStyle`, `UMLListStyle`, `FillStyle`, `LineStyle` and the UML styles `UMLNameStyle`, `UMLParentStyle`, `UMLStereotypeStyle`. For full details, please see the references section. The UML styles are defined as public API in

com.ibm.xtools.umlnotation. Please see “Product help – com.ibm.xtools.umlnotation” in the [Resources](#) section.

- **StyleFeatureValue:** The generated view customizer sets the UMLStereotypeStyle to **Image**. Other possible values for UML stereotype style are **None**, **Text**, **Icon**, and **Label**.

Note: If you have generated elements for stereotype associations or metaclass associations, a custom shape will be generated, as described in the next section.

3.8.2 Generating custom shapes

If you have chosen to generate custom shapes, your tooling model will contain classes stereotyped as NodeEditPart (non-relationship elements) and LinkEditPart (relationship elements). NodeEditPart contains a TextEditPart and a Figure, while a LinkEditPart contains a LabelEditPart further containing a TextEditPart. When code is generated, no view customizers will be generated. Instead, view factories will be generated in the viewFactories package, and corresponding edit parts and figures are generated in the editParts and figures packages.

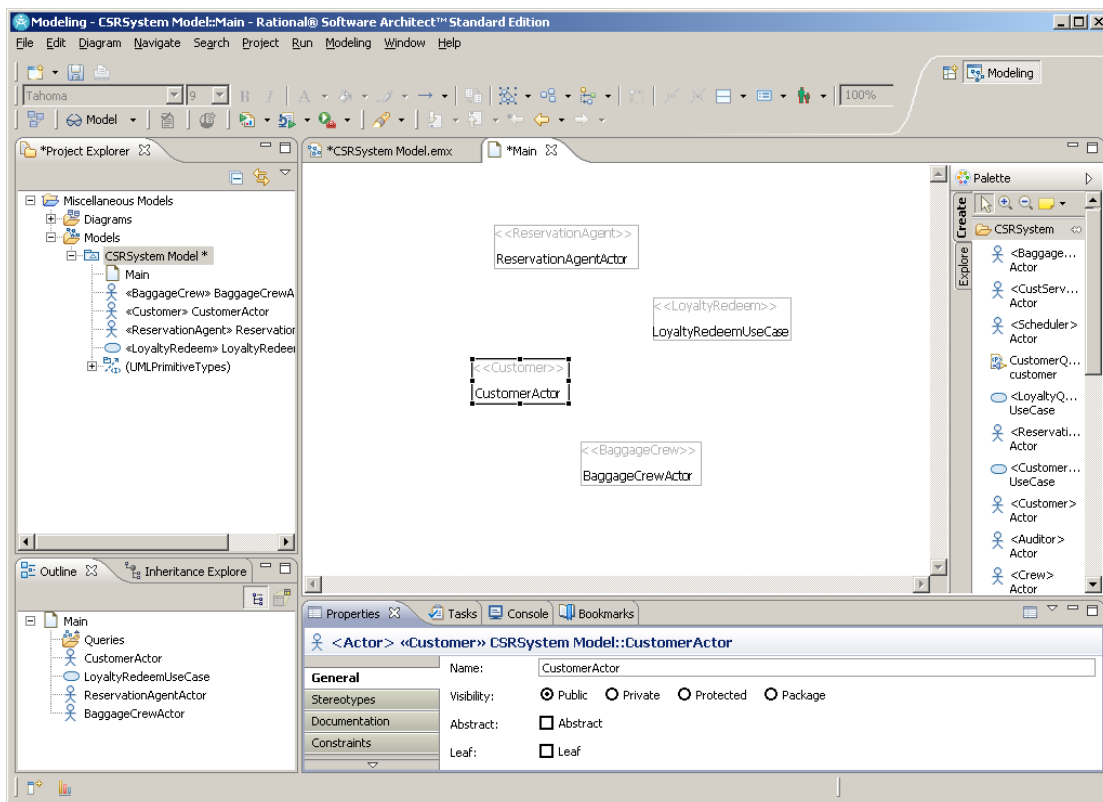


Figure. The shapes in this diagram use custom UML notation.

When generating custom shapes for non-relationship elements, the classes in the **Shapes** package may be stereotyped as one of the following:

- **NodeEditPart:** The NodeEditPart is the basic edit part describing non-relationship elements. Of note, you can customize the names of the generated Java classes and the styles.
- **TextEditPart:** This corresponds to the text box that appears inside of the edit part.

- **Figure:** This corresponds to the figure for the edit part.

When generating custom shapes for relationship elements, the classes in the **Shapes** package may be stereotyped as one of the following:

- **LinkEditPart:** The LinkEditPart is the basic edit part describing relationship elements. Of note, you can customize the names of the generated Java classes and the styles.
- **LabelEditPart:** This corresponds to the edit part that contains the TextEditPart. Of note, you can customize the anchor location.
- **TextEditPart:** As for non-relationship elements, this corresponds to the text box that appears inside of the edit part.

To summarize:

- a class stereotyped as DefaultEditPart is used when default UML notation is to be used;
- a class stereotyped as NodeEditPart contains a class stereotyped as TextEditPart and a class stereotyped as Figure;
- a class stereotyped as LinkEditPart contains a class stereotyped as LabelEditPart;
- a class stereotyped as LabelEditPart contains a class stereotyped as TextEditPart;
- a class stereotyped as Style contains a class stereotyped as StyleFeatureValue.

Note: There is no Figure generated for relationship elements because the generated edit parts extend GMF's ConnectionNodeEditPart which already defines a default figure. However, the non-relationship elements extend GMF's ShapeNodeEditPart, which does not define the default figure.

3.9 Properties

In version 7.5, it is possible to generate a custom tab in the **Properties** view corresponding to your particular element. By default, this generated tab contains the stereotype specific properties that are normally shown in the **Advanced** properties tab and in the **Stereotype Properties** section of the **Stereotypes** properties tab. Please see “Eclipse tabbed properties” in the resources section for more details on how to write code to expand on the generated properties.

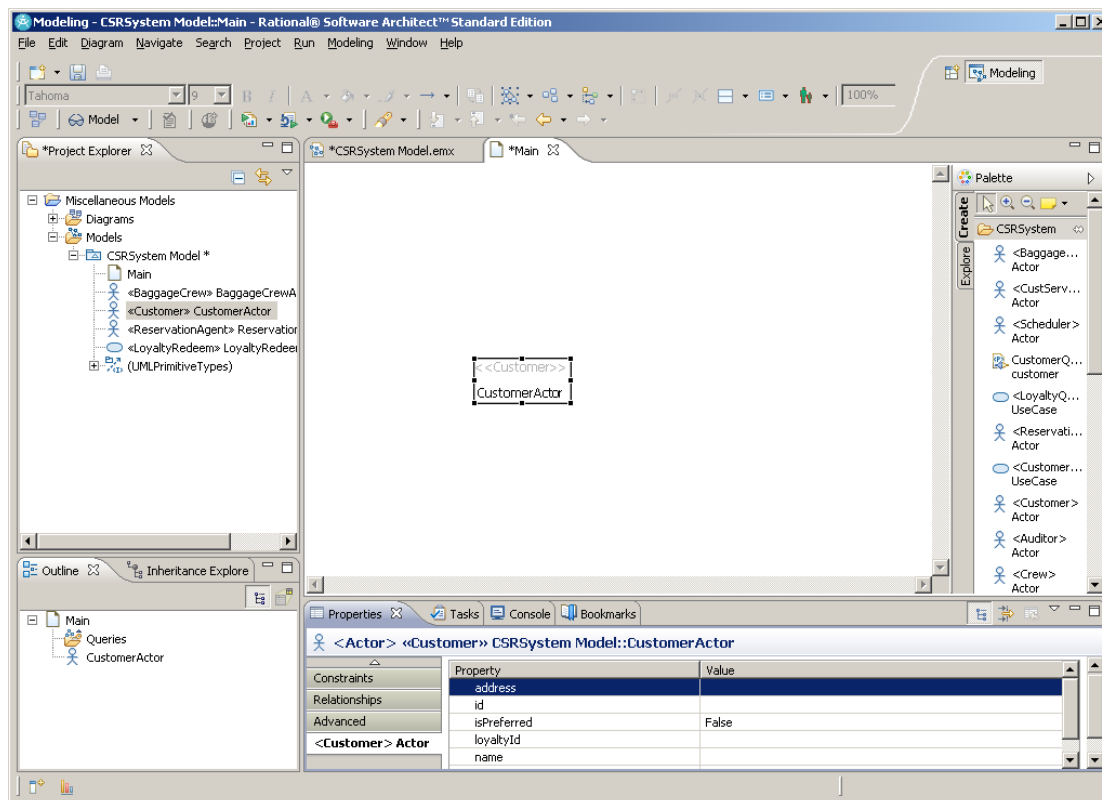


Figure. The Properties for the <<Customer>> actor are shown.

The classes in the **Properties** package may be stereotyped as one of the following:

- **PropertyCategory:** A PropertyCategory references the classes stereotyped as PropertyTab that custom properties will be generated for.
- **PropertyTab:** This corresponds to the custom property tab.
- **PropertySection:** A section in the tab. Each section references an element type. By default, one section is generated per tab.

To summarize:

- a class stereotyped as PropertyCategory references classes stereotyped as PropertyTab;
- a class stereotyped as PropertyTab references classes stereotyped as PropertySection.

4 Wizards

The classes in the Wizards package provides the ability to customize the contributions to the **New Model Wizard**.

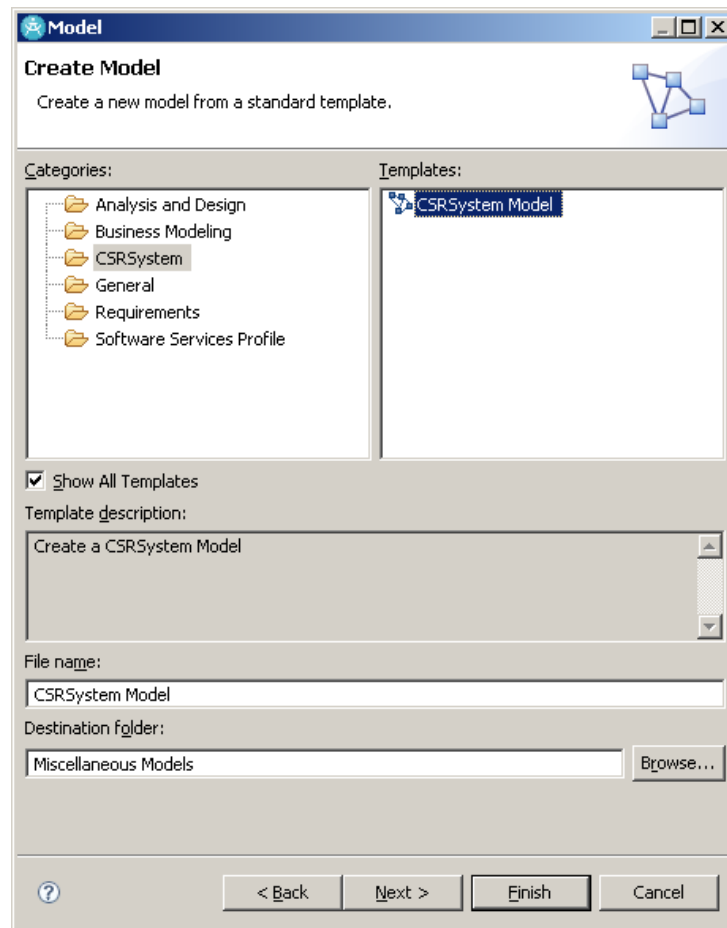


Figure. The New Model Wizard.

The classes in the Wizards package may be stereotyped as one of the following:

- **TemplateContribution:** A TemplateContribution references classes stereotyped as Activity, TemplateCategory, and Template. You can also specify the template directory and the generated Java class name of the template handler.
- **Activity:** We discussed the use of activities earlier, and how to customize the activity that controls the tooling. By default a separate activity is generated for the wizards.
- **Template:** This allows customization of the template description, template id, model name, and template file.
- **TemplateCategory:** Here it is possible to set the ID of the category the template should appear in the **New Model Wizard**.

5 Dynamic Profile Tooling diagrams

Apart from using Show Related Elements to visualize the related tooling elements at a high level, you can use dynamic Profile Tooling diagrams to show the relationships of a certain tooling class.

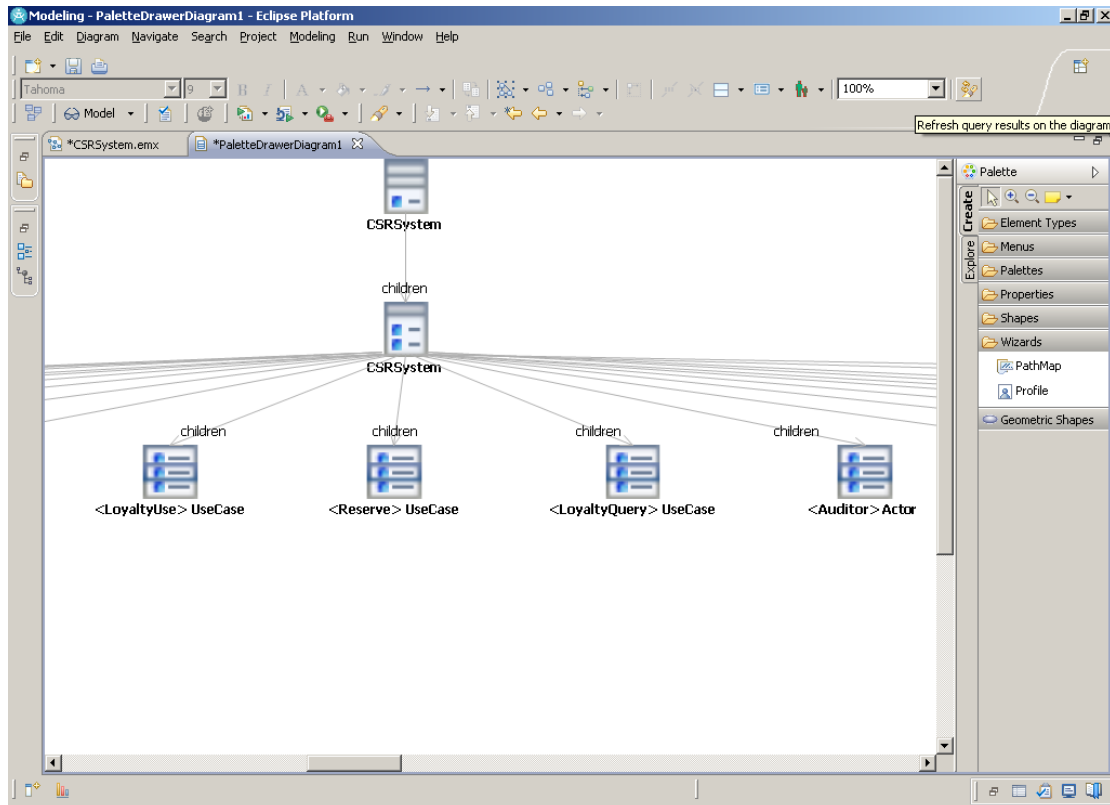


Figure. A dynamic PaletteDrawer diagram. Notice the button to refresh the diagram in the top right corner.

Note: Dynamic Profile Tooling diagrams are supported for most classes with a Profile Tooling stereotype applied, but not packages. You can easily use Show Related Elements to visualize the contents of packages or simply examine the containment hierarchy in the **Project Explorer**.

To create a Profile Tooling diagram:

1. Right click on the element which has an applicable Profile Tooling stereotype applied (such as PaletteDrawer).
2. From the context menu, choose **Add Diagram, Add [ProfileToolingStereotype] Diagram**, where *[ProfileToolingStereotype]* is a profile tooling stereotype, such as PaletteDrawer. A dynamic palette drawer diagram shows how the elements in the palette are related.

Because this is a query based diagram, it will update based on the elements in the tooling model. To force a manual refresh, press the **Refresh query results on the diagram** button.

Tip: The **Refresh query results on the diagram** button may be hidden in the toolbar area, so you may have to move the toolbars around to see it.

6 Change management

Recall the **Profile Tooling Plug-in Project** wizard gave the options to **Generate a tooling model** and to **Generate tooling code**. Both the tooling model and the tooling code must be kept in sync with changes to either the profile or the tooling.

6.1 Responding to changes in the profile

If you have made changes to a profile not deployed in the workspace, you can regenerate the tooling model to update the tooling model with the changes in your profile. For example, if you add or remove a Stereotype with a metaclass extension to a concrete subtype, you may want to add or remove the corresponding element to or from your tooling.

To update your tooling model:

1. Right click on the model in the **Project Explorer**.
2. From the context menu, choose **Save**.
3. Right click on the tooling model again.
4. From the context menu, choose **Update Tooling Model**.
5. A dialog appears informing you that the changes will be merged into the model. Press the **OK** button to proceed.
6. A dialog allowing you to perform a visual merge appears. The left side of the dialog represents the tooling model generated from the latest version of the profile. The right side of the dialog represents your existing tooling model. Check the checkboxes of the changes you wish to accept and press the **OK** button to update your tooling model.
Note: Be careful not to accidentally accept the changes to delete your existing diagrams. Since no diagrams are generated by default in the tooling model, the lack of a diagram is considered as a change.

Alternatively:

1. Right click on the profile in the **Project Explorer**.
Note: Be sure to right click the instance of the profile that was copied into the generated plug-in. If you have made changes to some other instance of the profile, you will need to replace the profile copied into the generated plug-in.
2. From the context menu, choose **Generate Profile Tooling Model**.
3. Continue through Steps 3 and 4 as shown above.

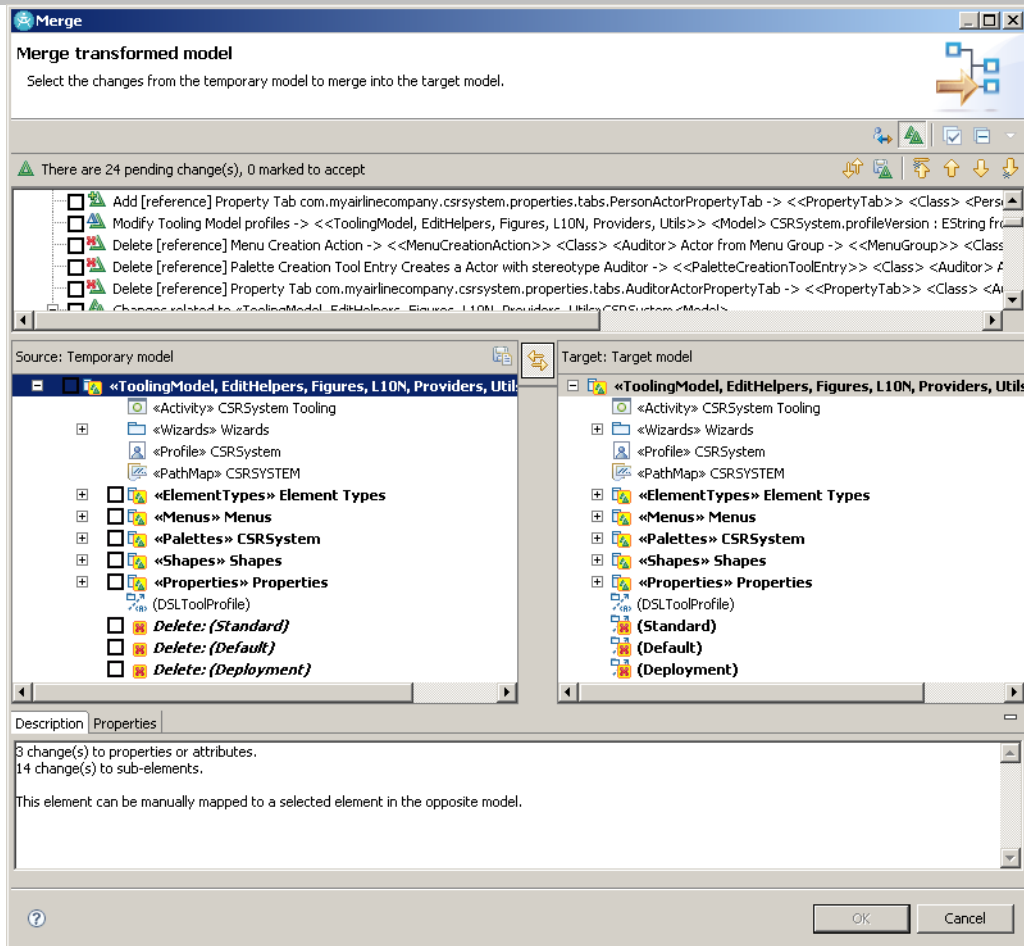


Figure. A visual merge.

6.2 Updating tooling model generation properties

When you used the **Profile Tooling Plug-in Wizard**, you were given the option to choose the type of tooling to generate and the elements for which tooling should be generated. You can adjust those settings without editing the tooling model.

To update the tooling model generation properties:

1. Right click on the tooling model in the **Project Explorer**.
2. From the context menu, choose **Properties**.
3. On the left side of the dialog, choose **Tooling Model Generation**.
4. You are presented with the same properties shown in the **Advanced Tooling Model Generation Properties** dialog. The settings you specified previously are remembered, so you do not have to start from scratch. Make the desired changes and press the **OK** button.
5. At this point, your tooling model has not been updated. Now, save your model and repeat the process to update your tooling model described in the [Responding to changes in the profile](#) section.

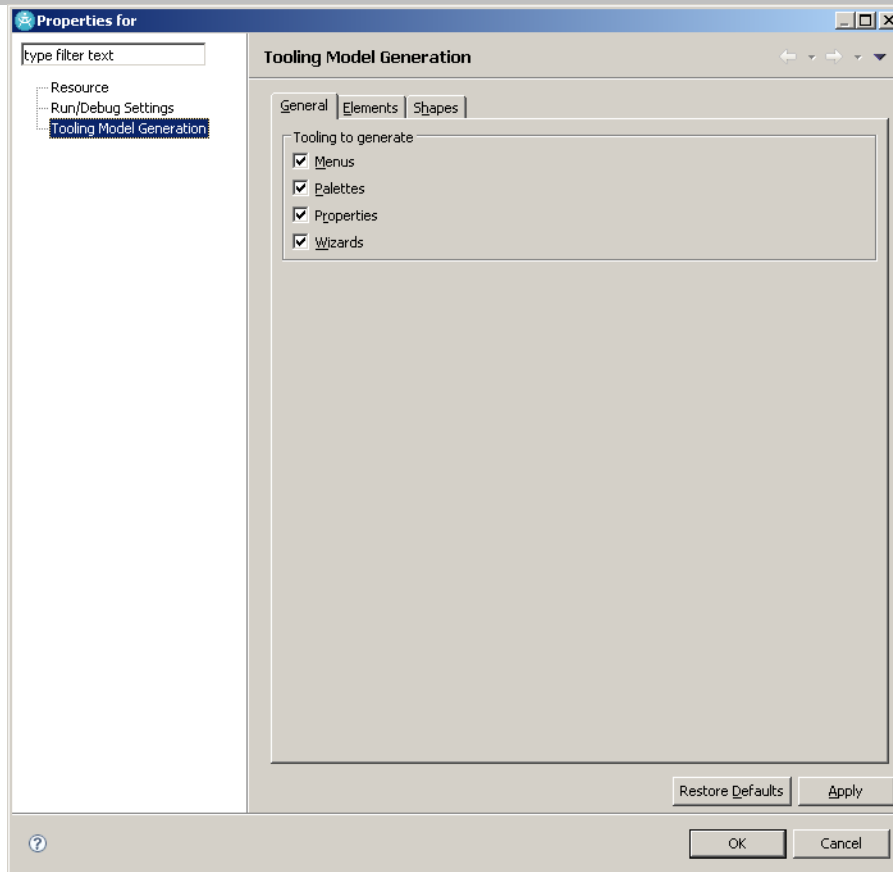


Figure. The **Tooling Model Generation Properties** dialog.

6.3 Generating code from the tooling model

If **Generate tooling code** was not checked, or if made changes to the tooling model after generating the tooling code, you will need to (re)generate the tooling code. Tooling code is not automatically regenerated after updating your tooling model.

To regenerate the tooling code:

1. Right click on the tooling model.
2. From the context menu, choose **Save**.
3. Right click on the tooling model again.
4. From the context menu, choose **Generate Tooling Code**.

When the code is regenerated, code is merged as much as possible. To enable the code to merge successfully, the generated tooling code contains the `@generated` tag in the Javadoc of the methods. If you have made changes to the generated code, and you wish that those changes will not be overwritten during code regeneration, change the `@generated` tag to `@generated NOT`.

Second, the code regeneration is as non-destructive as possible. If an element type has been removed from your tooling model (either by you or because the tooling model was regenerated to be synchronized with the profile), the previously generated Java file will not be deleted. You should delete those files manually after reviewing them.

Please be aware that non-Java files are not merged upon regeneration: they are overwritten. If you had changes that were overwritten upon regeneration, they can be restored from Local History. To restore your changes from Local History:

1. Right click on the file in the **Project Explorer**.
2. From the context menu, choose **Replace With, Previous from Local History**.

Tip: If you totally mess up a Java file, you can delete it and regenerate the tooling code. Starting over from scratch can be much simpler and less time-consuming than trying to fix problems in the code.

7 Running the Generated Plug-in

The generated plug-in can be tested in your environment prior to deployment. To do this, launch a new instance of the runtime workbench.

7.1 Launching a new instance of the runtime workbench

To perform a quick test in your environment:

1. Switch to the Java perspective using the **Open Perspective** button at the top right of the toolbar.
2. Open the **Debug Configurations** dialog by choosing **Debug, Debug Configurations.....**
3. Double-click **Eclipse Application** on the left side to create a new Eclipse Application configuration.
4. Type a name for your configuration.
5. In the **Main** tab, you can choose a new location (such as C:\my-runtime-workspace).
6. For **Program to Run**, leave the default settings of **com.ibm.rational.rxx.product.v75.ide**
7. Press the **Debug** button.

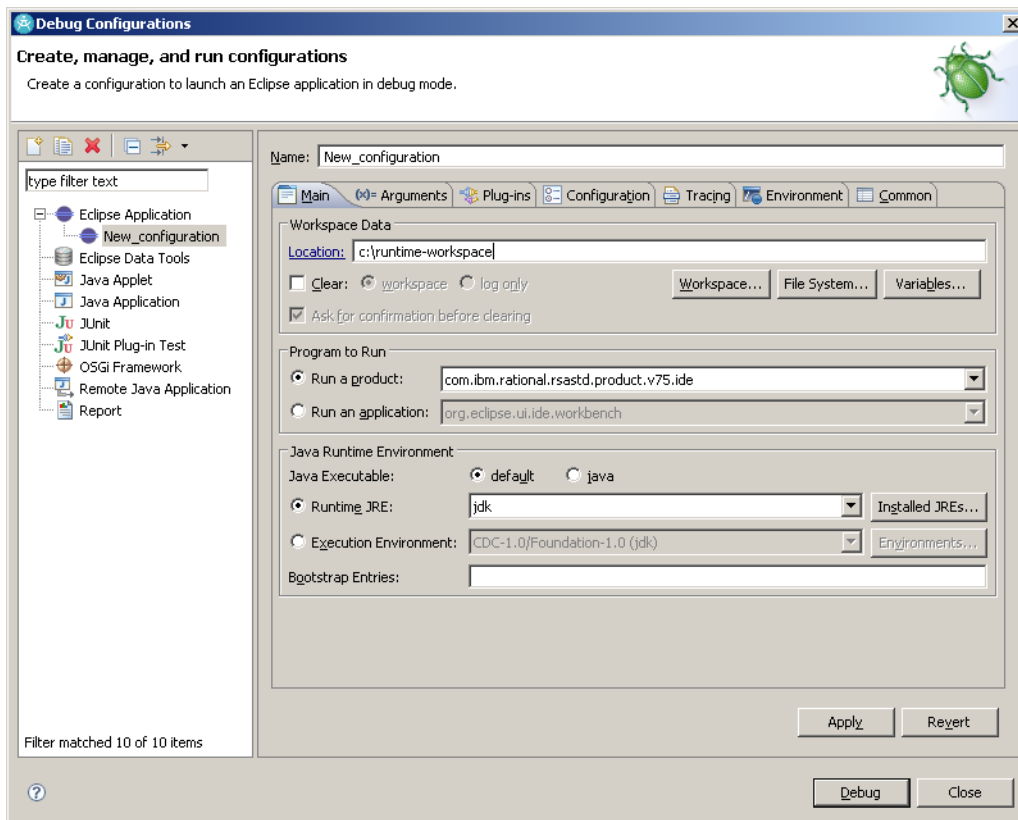


Figure. The **Debug Configurations** dialog.

Note: Under the **Plug-ins** tab, ensure that **Launch with** is set to **all workspace and enabled target plug-ins**. If you must choose **plug-ins selected below**

only, ensure that the entry for the plug-in corresponding to the tooling is checked.

Note: If you plan to modify the generated code, it is a good idea to include `-ea` in the **VM Arguments** box under the **Arguments** tab. Some of the generated code includes assertions, so you may find that enabling the assertions will help you track down programming errors faster.

7.2 The New Model Wizard

After you have launched a runtime instance, you can create a new model which will allow you to model using the generated tooling.

Note: This only applies if you have chosen to generate tooling for **Wizards** in the **Tooling Model Generation** properties.

To create a new model:

1. Choose **File, New, Project...**
2. Expand the **Modeling** category and choose **New Model Project**.
3. Check the **Show all templates** check box to show the template associated with the tooling for your profile.
4. Select that template, and then press the **Next** button.
5. Review the capabilities displayed in the **Model Capabilities** wizard page. If you selected any capabilities in your template model, they will be checked here.
6. Press the **Finish** button to create the new model. If you added any elements to your template model, they will be in this newly created model.

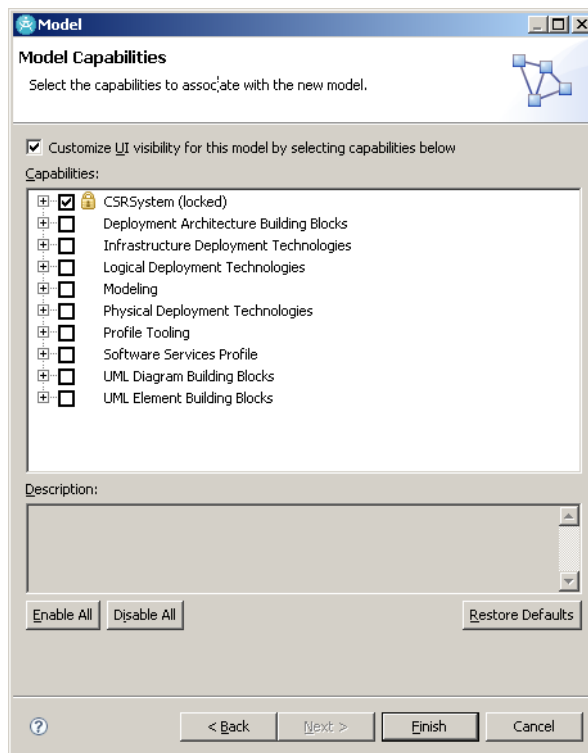


Figure. The **Model Capabilities** page in the **New Model Wizard**.

The next time you use the wizard, you will see the template even when the **Show all templates** check box is unchecked. This is because the corresponding capability has been enabled. To enable or disable capabilities manually:

1. Choose **Window, Preferences**.
2. Choose **Capabilities** from the left side.
3. Press the **Advanced...** button.
4. Locate the capability corresponding to your profile and expand it. The **Core** subcategory controls the wizard, while the **Tooling** subcategory controls the tooling in the diagram. Check or clear the checkboxes to enable or disable the capabilities as appropriate.

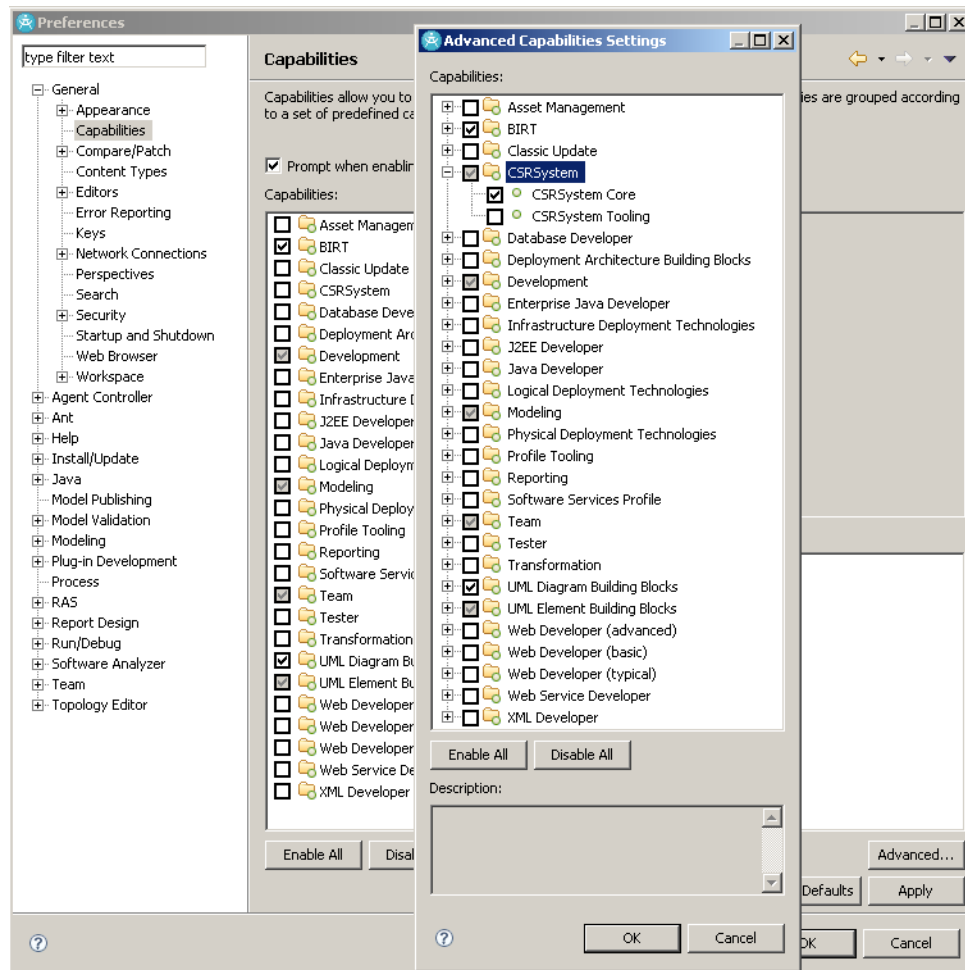


Figure. The **Advanced Capabilities Settings** from the **Preferences** dialog.

Observe that tooling corresponding to the profile elements appears in the palette. Bring up the context menu on the diagram surface, and observe there are now menu items that correspond to the profile elements. The palette entries corresponding to connectors all follow live constraints. That is, they can be connected only in a way that follows UML rules and the tooling's rules. Thus, you would not be able to:

- add an unsupported connector to a node (such as an implements connector to a UML package);
- create a stereotype association connector between objects that do not have the expected stereotype applied;

- create connectors in the wrong direction.

Similarly, it is not possible to add an element onto a diagram that does not support it. For example, the tooling would prohibit adding a lifeline onto a class diagram.

7.3 Enabling additional capabilities

By default, when the domain modeler creates a model from a profile tooling template model, the tooling is restricted to the generated tooling. The modeler can see additional tooling by performing one of the following tasks.

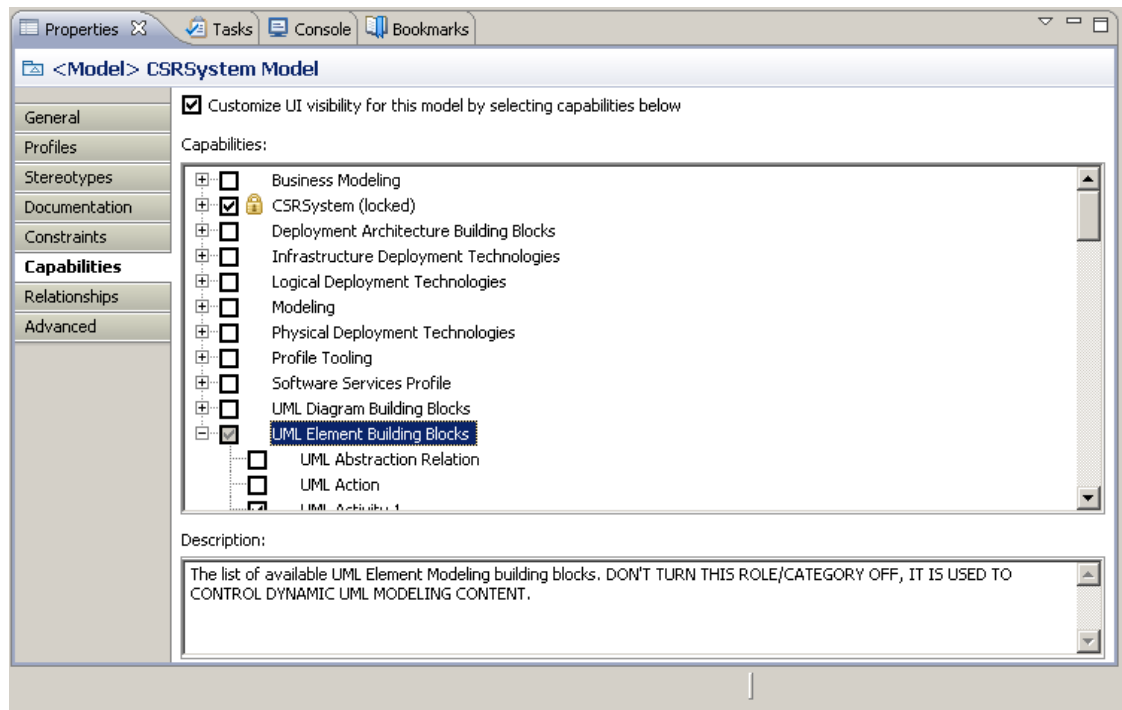


Figure. The **Capabilities** tab in the **Properties** view.

The first method involves disabling the customization of capabilities:

1. Click on the model in the **Project Explorer**.
2. Activate the Properties view and choose the **Capabilities** tab.
3. Clear the checkbox to **Customize UI visibility for this model by selecting capabilities below**.
4. The capabilities will be as defined in the preferences, which we described how to customize [earlier](#).

The second method involves directly defining the capabilities in the model:

1. Click on the model in the **Project Explorer**.
2. Activate the Properties view and choose the **Capabilities** tab.
3. Verify that the checkbox to **Customize UI visibility for this model by selecting capabilities below** is checked.
4. Locate the **UML Element Building Blocks** capability (and optionally the **UML Diagram Building Blocks** capability).
5. Check the checkboxes next to the desired UML elements (and optionally UML diagrams).

Note: The capabilities may also be customized in the **Model Capabilities** page of the **New Model Wizard**.

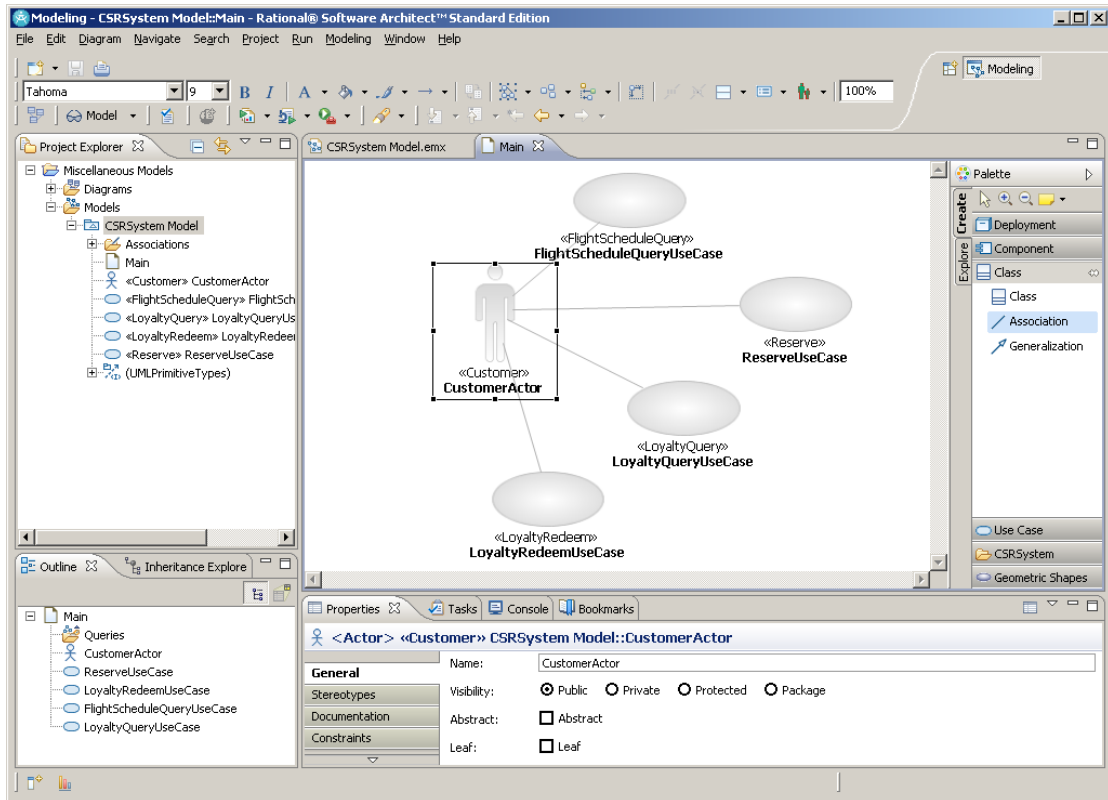


Figure. Modeling with both plain UML elements and domain specific elements.

7.4 Working with existing models

If the domain modeler is working with an existing model, the tooling may not show up unless the capabilities settings are adjusted. The capabilities settings can be adjusted as described above. The generated profile tooling code will apply the profile automatically to the model when an attempt is made to add a domain element onto the diagram using the tooling. However, a profile can also be applied manually.

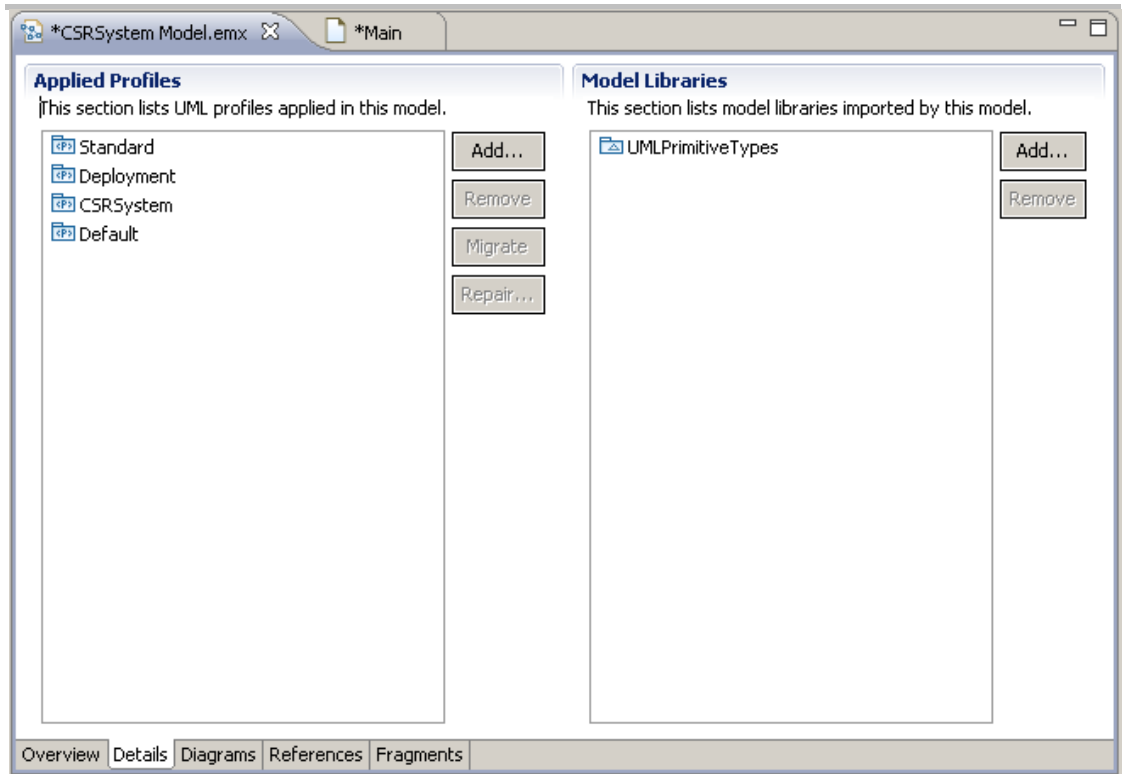


Figure. The **Details** tab in the **Model Editor**.

To manually apply a profile:

1. Double click a model to open the **Model Editor**.
2. In the **Model Editor**, select the **Details** tab and click the **Add...** button in the **Applied Profiles** section. Since the Profile Tooling wizard deploys the profile into the generated plug-in, the profile corresponding to the generated tooling shows in the list of deployed profiles.
3. Select the profile, and then click OK.

Even though the generated tooling applies the profile automatically upon adding a domain element, deleting the element will not automatically unapply the profile.

To remove a profile application:

1. Double click a model to open the **Model Editor**.
2. In the **Model Editor**, select the **Details** tab and click the **Remove** button in the **Applied Profiles** section.

Note: The above steps can also be done in the **Properties** view for the model from the **Profiles** property tab.

8 Deploying the tooling

When you are satisfied with your tooling, you are ready to deploy it to domain modelers.

8.1 Deploying tooling the easy way

The easiest way for the toolsmith is to distribute the generated project to the domain modeler. The modeler must import the project into the workspace by using **File, Import, Existing Projects into Workspace**, and then launch a new runtime workbench (as we did to test the tooling). This method is the worst of all: one missing file and the plug-in ceases to work. Furthermore, requiring another instance of the runtime workbench increases memory consumption. Therefore, this is not a recommended approach. This method should only really be used when you are debugging and developing your plug-in.

8.2 Deploying tooling as a JAR file

A more reasonable step is to generate a Java Archive (JAR) file for the tooling project. The JAR file includes the generated code in its compiled form. To deploy tooling as a JAR file:

1. Right click the generated project that contains your tooling from the Package Explorer.
2. From the context menu, choose **Export, Plug-in Development, Deployable plug-ins and fragments**.
3. Ensure that only the required plug-in is checked and, within the **Destination** tab, enter the directory destination for the JAR output.
4. (Optional) In the **Options** tab, you may find it helpful to check the **Include source code** option if you're planning on customizing the code. Doing so will make it easier to debug.
5. Press the **Finish** button.

Tip: If this fails, remove unnecessary plug-ins from your environment if you added any. Likely, another builder is trying to contribute and it is failing, causing the entire process of generating the jar to fail.

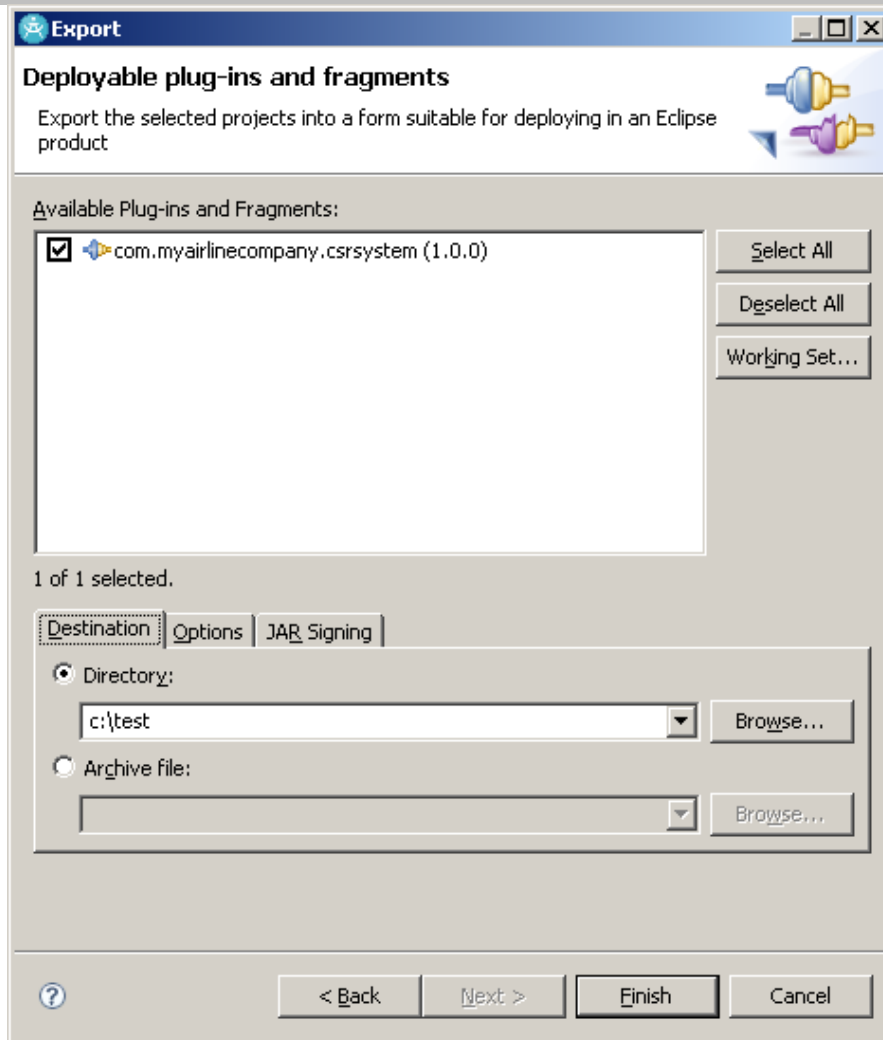


Figure. The **Export** wizard.

8.2.1 Importing the JAR

There are several ways to go from here. A possible way is to give the modeler your jar. The modeler imports it into the workspace using **File, Import, Plug-ins and Fragments**, then launches another instance of the workspace. Again, this has memory implications.

8.2.2 Including the JAR in the plugins directory

A far better way is to copy the jar file into the product install's plugins directory. If the application is installed in the default location, here is where to find the directory, depending on which operating system you are using:

Microsoft® Windows®: C:\Program Files\IBM\SDP\plugins

Linux®: /home/<username>/eclipse/plugins

If the file is copied as part of the product's installation and the generated tooling is not going to be updated, this is a relatively easy approach. The generated tooling will be available the next time the application is launched.

There may be no plan to update the generated tooling, but assuming that there will be no updates is likely unrealistic. What you can do, however, is anticipate the updates and assign versions to your plug-in. By default, the plug-in version is taken from the profile's version. Therefore, if you make changes to your tooling without making changes to the profile, the version number will remain the same. You can overcome this problem by manually updating the version of your plug-in before generating the JAR file. To do this:

Double click the manifest.mf file in the generated plug-in project to open it. On the **Overview** page, increment the version in the **General Information** section (see figure).

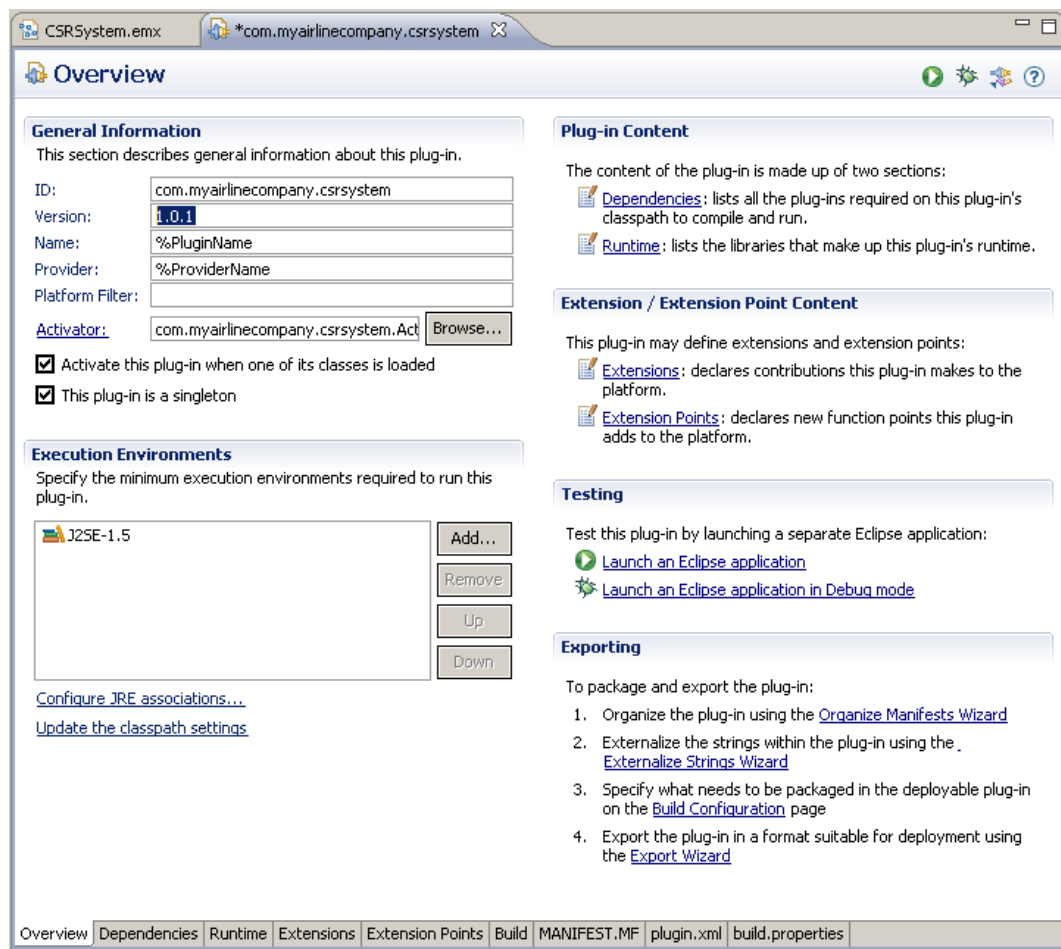


Figure. The **Manifest Editor**.

Now, the next time the tooling is updated, it just needs to be copied into the correct location, and the application needs to be relaunched. Updating the version of the manifest prevents naming conflicts when the JAR file is copied into the plugins folder.

8.3 Deploying tooling using an update site

The final method is the standard Eclipse approach.

To create an update site and publish your plug-in on the site
Select **File, New, Other, Plug-in Development, Update Site Project**, and follow the prompts.

This step is far more involved. See the Resources section.

Troubleshooting tip:

If the JAR file is copied into the plug-in folder of the installation and the tooling fails to appear, first verify that the correct version of the plug-in is recognized.

To do so:

1. Select **Help, About Eclipse SDK**.
2. Click **Plug-in Details**, and verify that the generated plug-in is in the list.
3. If the plug-in is not in the list, restart the application with the `-clean` parameter.
4. If the plug-in is in the list, check the **Error Log** view by selecting **Window, Show View, Other, PDE Runtime, Error Log**, and checking for errors that pertain to the tooling plug-in.

Table 1 summarizes the methods for deploying your tooling plug-in.

Sig...	Provider	Plug-in Name	Version	Plug-in Id
	Eclipse.org	Core Runtime	3.4.0.v2008...	org.eclipse.core.runtime
	Eclipse.org	Core Runtime Plug-in Com...	3.2.0.v2007...	org.eclipse.core.runtime.compatibility
	Eclipse.org	Core Variables	3.2.100.v20...	org.eclipse.core.variables
	IBM	CSH for deployment archit...	1.0.0.v2008...	com.ibm.ccl.soa.deploy.core.ui.cshelp
	IBM	Cshelp Plug-in	1.0.0.v2006...	com.ibm.ccl.f1.mapping.ui.cshelp
	My Company	CSR System Plugin	1.0.0	com.myairlinecompany.csrssystem
	Eclipse.org	CSV Data Extraction Plug-in	2.3.0.v2008...	org.eclipse.birt.report.engine.dataextraction
	Eclipse.org	CVS SSH Core	3.2.100.I200...	org.eclipse.team.cvs.ssh
	Eclipse.org	CVS SSH2	3.2.200.I200...	org.eclipse.team.cvs.ssh2
	Eclipse.org	CVS Team Provider Core	3.3.100.I200...	org.eclipse.team.cvs.core
	Eclipse.org	CVS Team Provider UI	3.3.100.I200...	org.eclipse.team.cvs.ui
	Eclipse.org	Data Core Plugin	1.0.0.v2008...	org.eclipse.datatools.sqltools.data.core
	Eclipse.org	Data Tools Platform Conn...	1.6.1.v2008...	org.eclipse.datatools.connectivity.doc.user
	Eclipse.org	Data Tools Platform Conn...	1.6.1.v2008...	org.eclipse.datatools.connectivity.doc.user.c
	Eclipse.org	Data Tools Platform Conn...	1.0.1.v2008...	org.eclipse.datatools.enablement.jdt.classpa
	Eclipse.org	Data Tools Platform Conn...	1.1.1.v2008...	org.eclipse.datatools.connectivity
	Eclipse.org	Data Tools Platform Conn...	1.1.1.v2008...	org.eclipse.datatools.connectivity.ui
	Eclipse.org	Data Tools Platform Help ...	1.5.0.v2008...	org.eclipse.datatools.help
	Eclipse.org	Data Tools Platform User ...	1.6.0.20080...	org.eclipse.datatools.doc.user
	Eclipse.org	Data UI Plugin	1.1.0.v2008...	org.eclipse.datatools.sqltools.data.ui

Figure. Plug-ins.

Table 1. Deployment methods for your tooling plug-in.

Manual deployment of the plug-in project	<ul style="list-style-type: none"> • Very easy for the tooling developer • User has easy access to source 	<ul style="list-style-type: none"> • Too easy to miss a file, causing the tooling to not function properly • Process to update the tooling may confuse the end user • Requires end user to launch another instance of the workspace (which requires more memory) • User has easy access to source and easy to modify, thus tooling could be out of sync on multiple users' machines
Manual deployment of the JAR file	<ul style="list-style-type: none"> • Very easy for the tooling developer • User has easy access to source code if the source is included in JAR file 	<ul style="list-style-type: none"> • Requires end user to launch another instance of the workspace (which uses more memory) • Process to update the tooling may confuse the end user • User has easy access to source code if the source is included in the JAR file and easy to modify, so tooling could be out of sync on multiple users' machines
Copy JAR file into the plug-in folder of the installation (recommended)	<ul style="list-style-type: none"> • Very easy 	<ul style="list-style-type: none"> • Updates to the tooling require copying the plug-in into the proper folder again
Use update site	<ul style="list-style-type: none"> • Standard Eclipse way of updating • End user can get updates for the profile tooling easily 	<ul style="list-style-type: none"> • More work for the profile developer to maintain the site

9 Upgrading from earlier versions

The GMF-based tooling models from previous versions of the product must be upgraded to standard .emx Modeler models in order to be used in version 7.5 of the product. To upgrade the model:

1. Locate the .epxgen file from the **Project Explorer** and double click on it.
2. An editor will open and prompt you to migrate the models.
3. Press the **Migrate** button.

Note: None of the other models (tool, graph, and map models) will open. You must upgrade the .epxgen file to a tooling model in the new format.

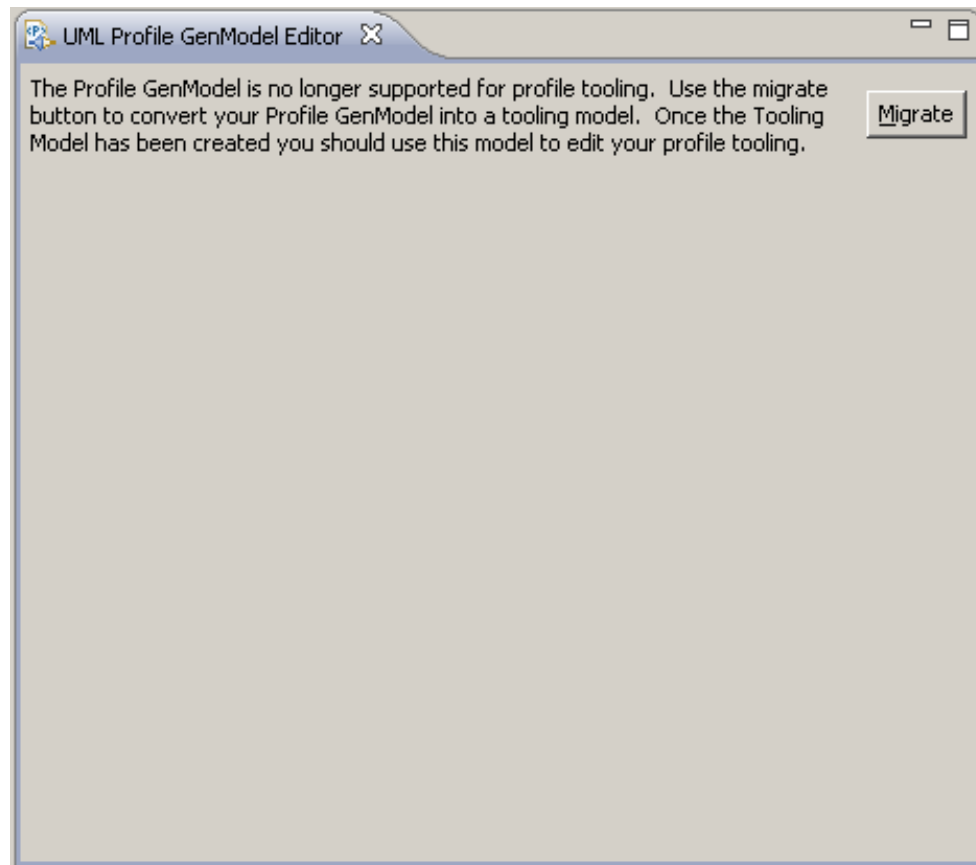


Figure. The Profile Tooling model migration editor.

10 Conclusion

With Profile Tooling, it is possible to become a toolsmith and quickly generate specialized tooling and shapes to allow true custom domain modeling. This tooling can be deployed to end users – domain modelers who might be familiar with a particular domain but unfamiliar with UML modeling. The capabilities settings work in conjunction with this feature to turn off the unnecessary tooling, such as UML palette entries and menu items, thereby simplifying the domain modeler’s workflows.

Profile Tooling enables domain specific modeling by generating custom tooling and custom shapes from an existing UML profile. The new features were intended to enhance the workflow so that working custom tooling can be generated quicker than before. Using the profile tooling feature, you can automatically generate custom code to provide a specialized diagram editor which only models elements from a particular domain defined in a UML profile. In addition, the elements can be shown using notation specific to the particular domain. Profile Tooling drastically reduces the time it would otherwise take to write the editor and shape code by hand, and eliminates the need to learn GMF, GEF, and Eclipse development in depth to add your custom palette, menu, wizard, and properties contributions.

11 Acknowledgements

The author expresses his thanks to Anthony Hunter, Dusko Mistic, Michael Hanner, and Michelle Crane for reviewing this article.

12 About the author



Wayne Diu is a software developer at IBM Rational. He worked on the original implementation of the Profile Tooling feature to allow custom domain modeling. Prior to that, he developed several features for the Rational Modeling Platform, such as the Browse Diagram infrastructure, and he was one of the developers responsible for the platformization of the metamodel integration framework. Wayne is now on the Modeler team, where he works on a diverse collection of features including those involving the Project Explorer, refactoring support, and UML modeling.

13 Resources

- [GMF Resources](#)
- [GMF Runtime Programmer's Guide](#)
- [Deploying the plugin in an Eclipse style update site](#)
- [Eclipse Tabbed Properties View](#)
- [Extending UML Modeler](#)
- Product help – Customizing diagram shapes
Extending product function > Extending the Rational Modeling Environment > Rational Modeling Platform Developer's Guide > Programmer's Guide > Customizing Diagrams > Customizing Diagram Shapes.
- Product help – com.ibm.xtools.umlnotation
Extending product function > Extending the Rational Modeling Environment > Rational Modeling Platform Developer's Guide > Reference > API Reference > UML Modeling Layer > com.ibm.xtools.umlnotation.
- [Authoring UML Profiles: Part 1](#)
- [Authoring UML Profiles: Part 2](#)
- [Custom domain modeling with UML Profiles in version 7.0.5: Part 1](#)
- [Custom domain modeling with UML Profiles in version 7.0.5: Part 2](#)