ProCom model-driven code generation
ProgressIDE Manual

Version 0.3

Revision History

Date	Version	Description	Author
2009-09-09	0.1	Initial Draft	Ana Petričić
2009-09-14	0.2	Finished section 3	Ana Petričić
2009-09-21	0.3	PG-IDE will not be distributed as an Eclipse product so I had to change section 2.	Ana Petričić

Doc. No.:

1. Introduction

1.1 Purpose of this document

This document describes the first steps to start working with ProgressIDE (PG-IDE) and the naming convention that must be used to create the various files (resources) needed in the project.

2. Installation and starting ProgressIDE

NOTE: This software is on-going development, and therefore bugs can appear in the environment. In such case, please raise the issues encountered and provide me an exact description of the situation to make it reproducible.

Requirements:

- Java SE Runtime Environment (JRE): 6.0 (or above). Note that having the SDK only is not enough
- Eclipse distribution prepared for you (“*eclipsePG-IDE”*)
- *“PG-IDEproject*” prepared for you

[1] Extract “*eclipsePG-IDE*” rar file to a specific directory (e.g., “C:\ProgressIDE\eclipsePG-IDE”).

Note 1: Unzip the distribution to a directory with a short directory name, e.g. 'C:\eclipse', otherwise some file names might be too long for the file system.

Note 2: Do not use spaces in the path and file name to avoid future problems. Replace space with underscore, e.g., 'my file.ext' should be named 'my_file.ext'.

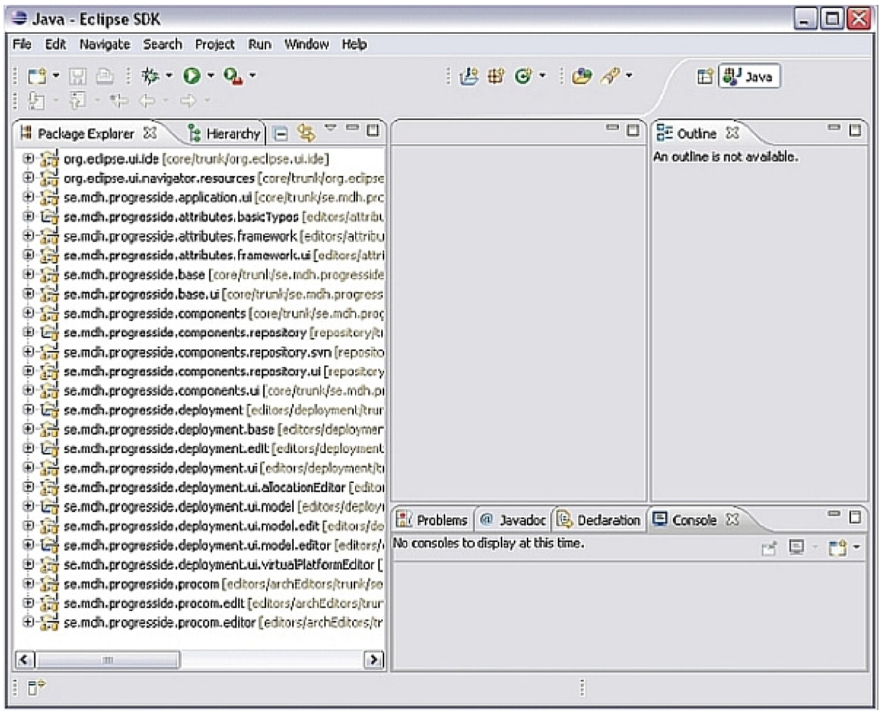
[2] Extract “*PG-IDEproject*” rar file to a specific directory. (See notes 1 and 2 above.)

[3] Start Eclipse by double clicking on “eclipse.exe” in the directory you have chosen in the step [1] (e.g., “C:\ProgressIDE\eclipsePG-IDE”).

Note 1: if you work with Windows a security warning might appear. Ignore it and proceed with starting of Eclipse.

[4] Choose prepared “*PG-IDEproject”* folder (prepared in step [2]) as workspace location.

[5] Eclipse should now start and you should obtain the following window (Fig. 1):



```
org.eclipse.ui.ide [core/trunk/org.eclipse.ui.ide]
org.eclipse.ui.navigator.resources [core/trunk/org.eclipse
se.mdh.progresside.application.ui [core/trunk/se.mdh.prc
se.mdh.progresside.attributes.basicTypes [editors/attribu
se.mdh.progresside.attributes.framework [editors/attribu
se.mdh.progresside.attributes.framework.ui [editors/attri
se.mdh.progresside.base [core/trunk/se.mdh.progresside
se.mdh.progresside.base.ui [core/trunk/se.mdh.progress
se.mdh.progresside.components [core/trunk/se.mdh.proc
se.mdh.progresside.components.repository [repository/tı
se.mdh.progresside.components.repository.svn [reposito
se.mdh.progresside.components.repository.ui [repository
se.mdh.progresside.components.ui [core/trunk/se.mdh.pı
se.mdh.progresside.deployment [editors/deployment/trur
se.mdh.progresside.deployment.base [editors/deploymer
se.mdh.progresside.deployment.edit [editors/deployment
se.mdh.progresside.deployment.ui [editors/deployment/tı
se.mdh.progresside.deployment.ui.allocationEditor [editoı
se.mdh.progresside.deployment.ui.model [editors/deployı
se.mdh.progresside.deployment.ui.model.edit [editors/de
se.mdh.progresside.deployment.ui.model.editor [editors/ı
se.mdh.progresside.deployment.ui.virtualPlatformEditor [
se.mdh.progresside.procom [editors/archEditors/trunk/se
se.mdh.progresside.procom.edit [editors/archEditors/trur
se.mdh.progresside.procom.editor [editors/archEditors/tr
```

An outline is not available.

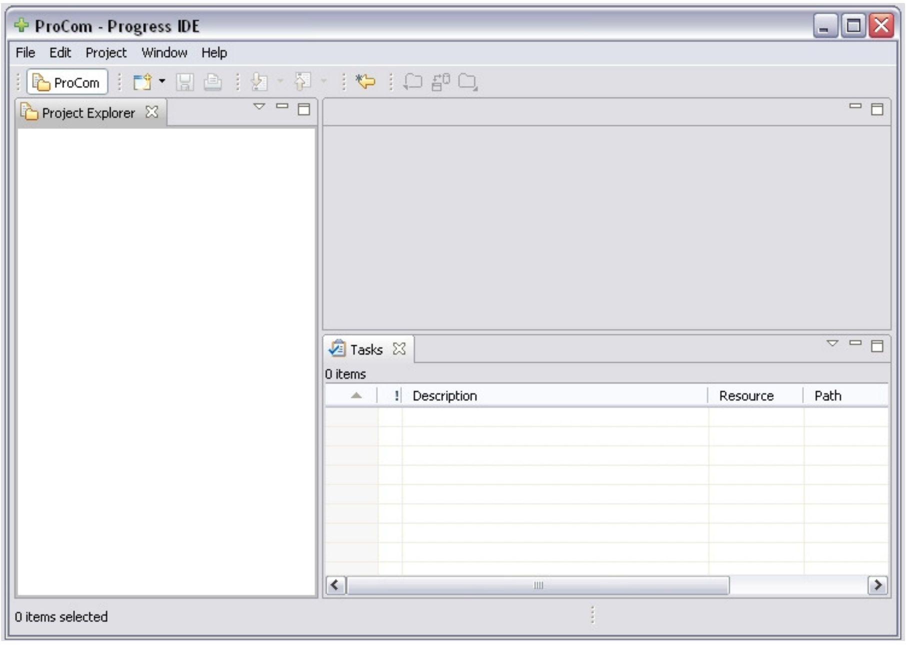
No consoles to display at this time.

Figure 1. Starting Progress-IDE, step 1

NOTE: Unfortunately, due to some "not-so easy to solve ☹" problems, I had to provide you the source code of PG-IDE. <u>This source code is not allowed to modify or copy for any other purposes than for DSD course.</u>

[5] In order to start ProgressIDE, you need to run projects as an Eclipse application. There is already prepared run configuration for you. Go to Run → Run Configurations, then choose "ProCom Application" (under Eclipse Application).

[6] ProgressIDE should now open and you should get an empty workspace as it is shown in the figure below:



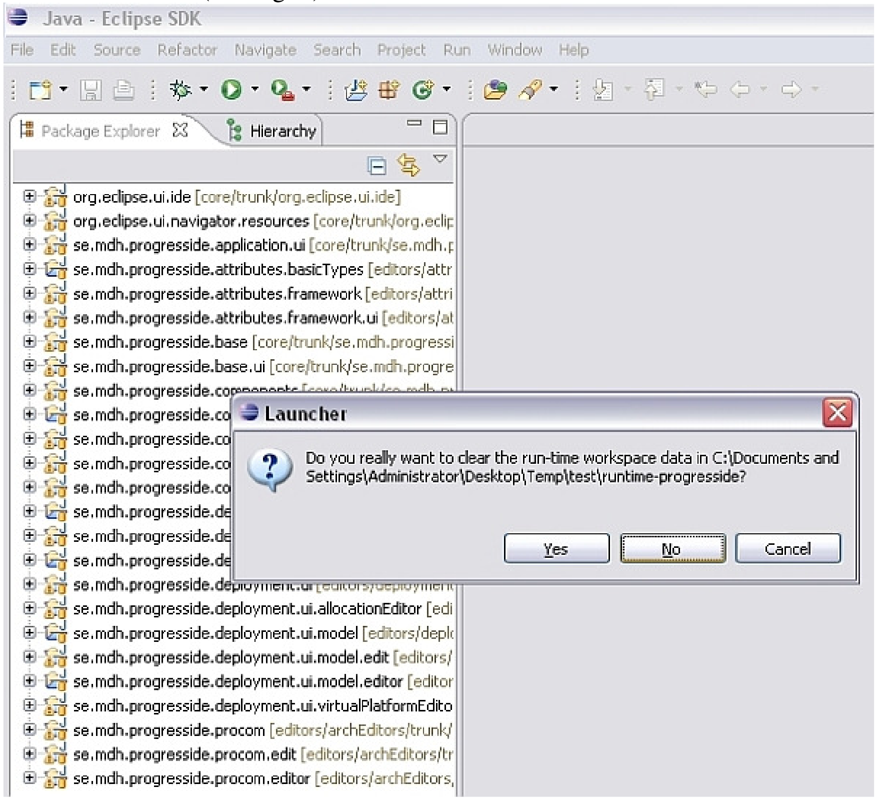
0 items

!	Description	Resource	Path

0 items selected

Figure 2. Progress-IDE workspace

NOTE: Every time you run PG-IDE (the way it is described above), you will get a question do you want to delete contents of run-time folder (see Fig. 3):


Java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
Hierarchy
Launcher
Yes
No
Cancel

Do you really want to clear the run-time workspace data in C:\Documents and Settings\Administrator\Desktop\Temp\test\runtime-progresside?

Figure 3. Restarting ProgressIDE

If you have saved your previous work (ProCom project and models of components) in default folder (which is actually the run-time folder) then choose <u>not to delete it</u> (so that you don't loose your previous work). Otherwise you can choose "yes", and then you need to import the existing projects manually into the workspace.

3. Using Progress-IDE

This section contains basic instructions for using ProgressIDE considering creating of ProCom project, adding new components to the project and their development using available tree editor.
Please read carefully ProCom reference manual in order to understand all concepts used in IDE.

3.1 Creating new ProCom project

- Go to File → New → ProCom project.
- Enter desired project name and click Finish
- New project should appear in Project Explorer window, like in picture below (Fig. 4):

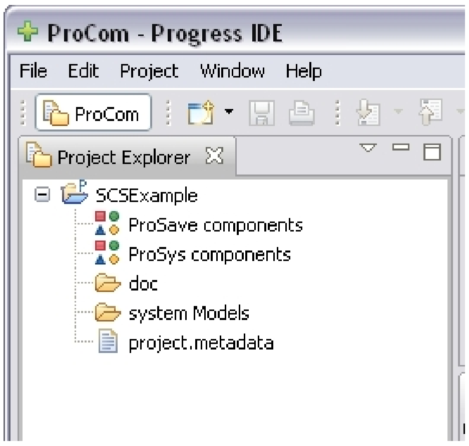


Figure 4. Structure of ProCom project

- Every ProCom project has pre-defined structure: *ProSave components* folder which holds all ProSave components in the project, *ProSys components* folder which holds all ProSys components and *doc* folder for system documentation. Other elements of the project are out of the scope of this document.

3.2 Creating new ProCom components

- Right click on *ProSave components* folder in Project Explorer.
- Choose New → ProSave Component from context menu.
- Enter desired Id and Name for new component.
- New component will appear in Project Explorer window, as it is shown in figure below (Fig. 5)

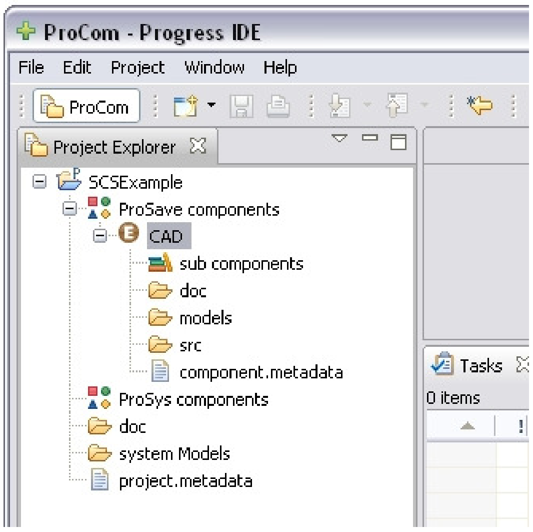


Figure 5. ProCom component file structure

- Every component has pre-defined directory structure. Folders that will be of your interest are *models* folder which contains a model of the component and *src* folder which holds files that contain component implementation (only in case the component is primitive ProSys subsystem or primitive ProSave component).
- Analogously you can create new ProSys component, which has to be placed in *ProSys Components* folder.

3.3 Progress-IDE terminology and main ProCom concepts

In this section terminology used in Progress-IDE is explained. Main terms considering ProCom component model are listed in table below.

Keyword	Definitions

ProCom component	ProSys subsystem or ProSave component
ProSys subsystem	ProSys composite subsystem or ProSys primitive subsystem
ProSys composite subsystem	ProSys subsystem whose internal structure is built using ProSys subsystems
ProSys primitive subsystem	ProSys subsystem whose internal structure is built using ProSave components or realized by code
ProSave component	ProSave composite component or ProSave primitive component
ProSave composite component	ProSave component whose internal structure is built using ProSave components
ProSave primitive component	ProSave component realized by code
black-box	Component whose internal structure is at this point undecided (either on the ProSys or ProSave level)

The following figure (Fig. 6) shows hierarchical relation between terms defined in the table.

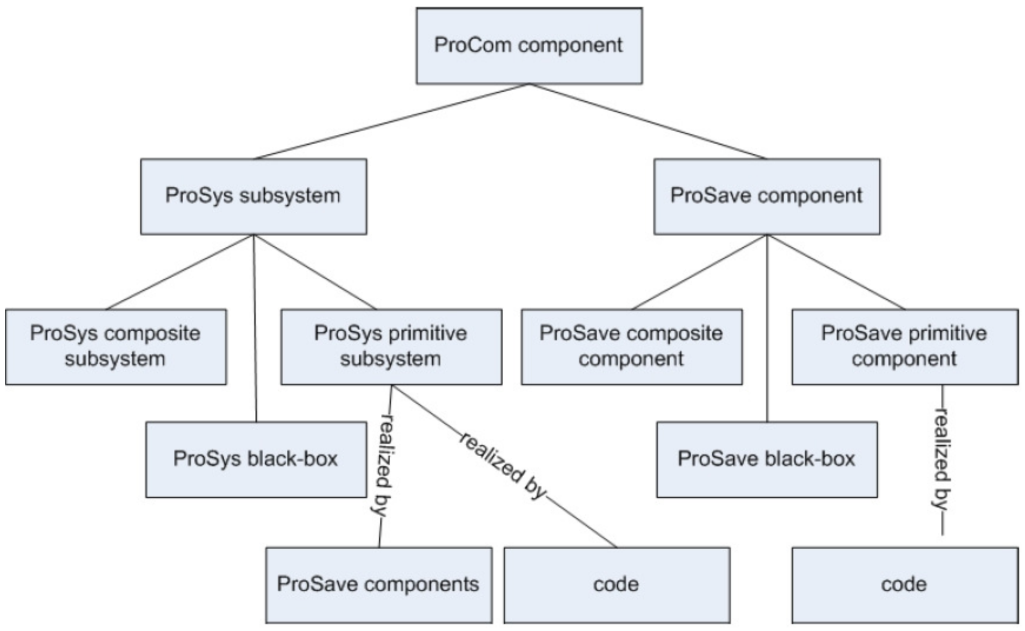


Figure 6. Relations between terms used in Progress-IDE

As it can be seen at Figure 6., ProCom components are divided to components at ProSave or at ProSys level.

Each ProSave component can be realized either as composite component (it is composed out of other **ProSave** components) or as primitive component (realized by source code), also its realization can be undefined (but you will not have to handle this case).

On ProSys level component is named *a subsystem.* Each ProSys subsystem can be realized as composite subsystem (it is composed out of other **ProSys** subsystems) or primitive subsystem, also its realization can be undefined (but you will not have to handle this case). Primitive ProSys subsystems have two possible realizations; they can be realized by code or they can be realized as a composition of **ProSave** components.

The notion of realization of a component must be distinguished from the notion of services and ports. Services and ports present an interface of the component (i.e. how this component is seen from the outside, what functionality it offers). Realization of a component presents its internal organization (i.e. how the specified functionality is achieved).

Another important concept in ProCom is the notion of *instance*. If a component is composed out of other components (ProSys composite subsystem, ProSave composite component or ProSave primitive subsystem), then these sub-components are actually instances. For example ProSys subsystem is actually composed out of instances of other ProSys subsystems. Likewise, ProSave composite component is a composition of instances of some other ProSave components. This is similar to relation between a class and an object in object oriented programming. Objects are instances of classes, so classes can be considered as “a type” of some object. If you use class C1 in class C2, then you actually use an instance of C1, in other words you use an object of type C1.

It is similar in ProCom, if some component C2 contains some other component C1, then it actually contains an instance of C1.

Finally, last important concept in ProCom is that everything is a component. The whole system that is designed in ProCom is considered to be a component composed out of other components.

3.4 Creating models of components

In order to use Progress-IDE tree-editor to edit some component, you have to create model files for that component. You can do this with following steps:

- In Project explorer right-click on “models” folder which belongs to desired component (Figure 7).

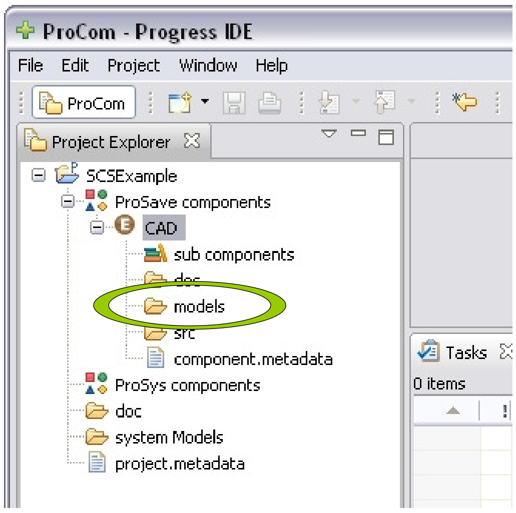
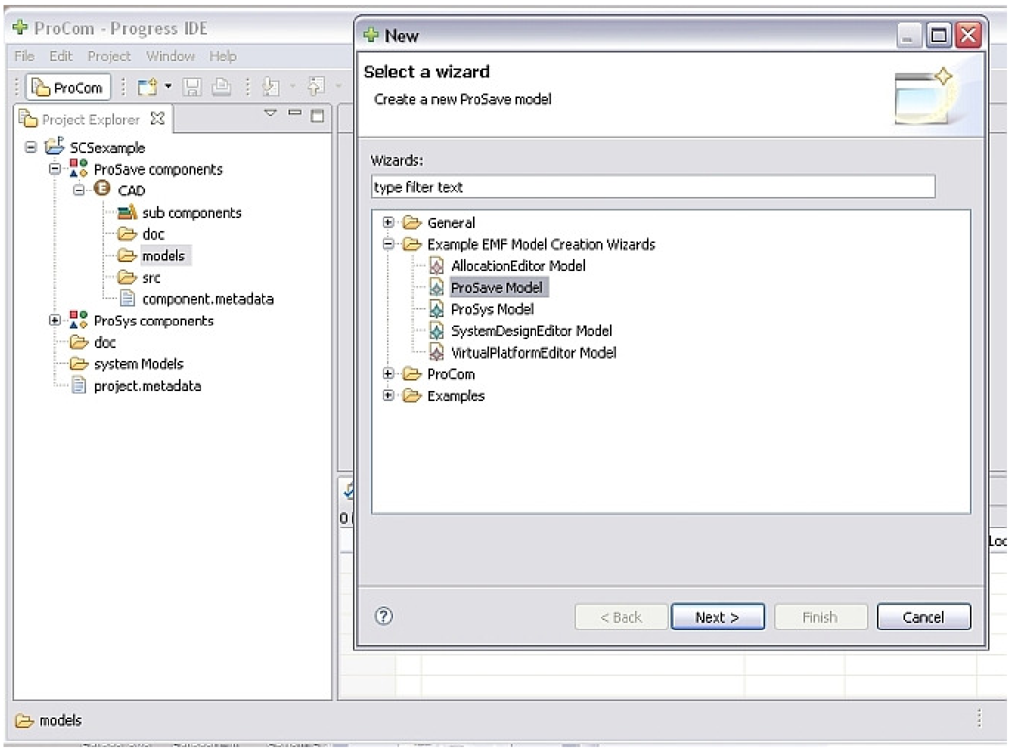


Figure 7. Creating model of component, step 1

- In pop-up menu, go to New → Other. A dialog like the one in Figure 8., below should appear.



ProCom - Progress IDE

Project Explorer

```
SCSexample
  ProSave components
    CAD
      sub components
      doc
      models
      src
      component.metadata
  ProSys components
  doc
  system Models
  project.metadata
```

New

Select a wizard

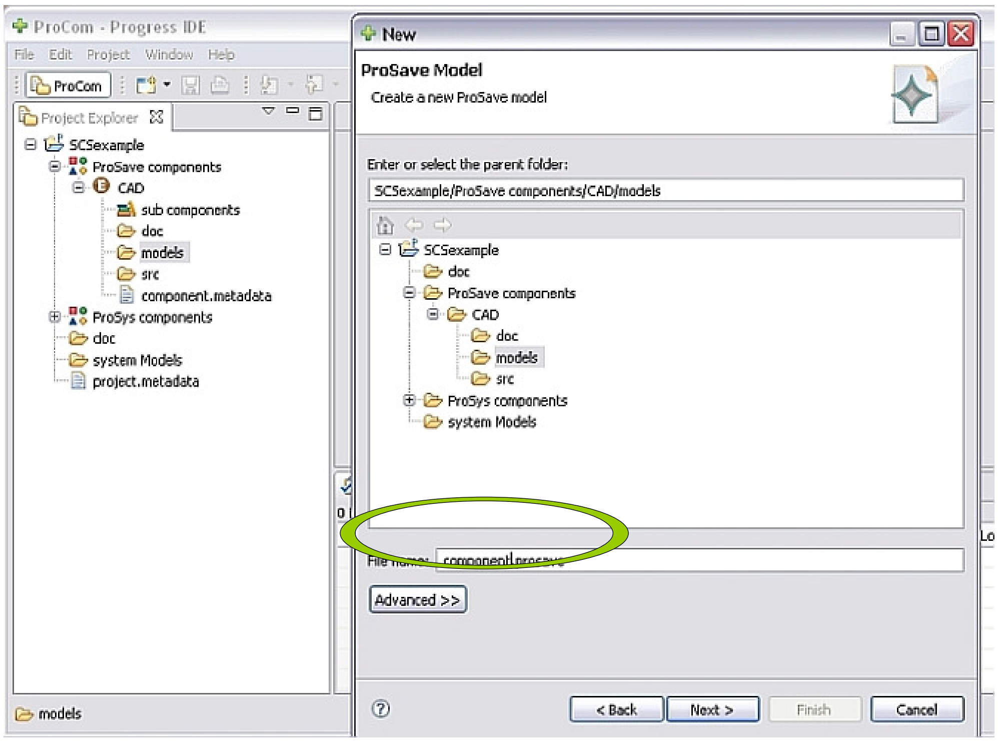
Create a new ProSave model

Wizards: type filter text

```
General
Example EMF Model Creation Wizards
  AllocationEditor Model
  ProSave Model
  ProSys Model
  SystemDesignEditor Model
  VirtualPlatformEditor Model
ProCom
Examples
```

Figure 8. Choosing model type

- Under “Example EMF Model Creation Wizards”, choose ProSave Model or ProSys Model. It depends whether you are creating a model of ProSave or ProSys component. In figure above I choose ProSave model because CAD component is an ProSave component (you can see it in Project explorer on the left).
- Click Next
- Under file name, the file **must** be called “component”. So the name is *component.prosave* or *component.prosys*. It can be seen in example below (Figure 9.):



ProCom - Progress IDE

New

ProSave Model

Create a new ProSave model

Enter or select the parent folder: SCSexample/ProSave components/CAD/models

```
SCSexample
  doc
  ProSave components
    CAD
      doc
      models
      src
  ProSys components
  system Models
```

File name: component.prosave

Figure 9. Naming the model file

- Click Next
- Final step is choosing the model object. You have to choose:
    - *Component* – if you are creating a model of ProSave component
    - *Subsystem* – if you are creating a model of ProSys component
- In example below (Figure 10.) I choose *Component* because I am creating a model of an ProSave component.

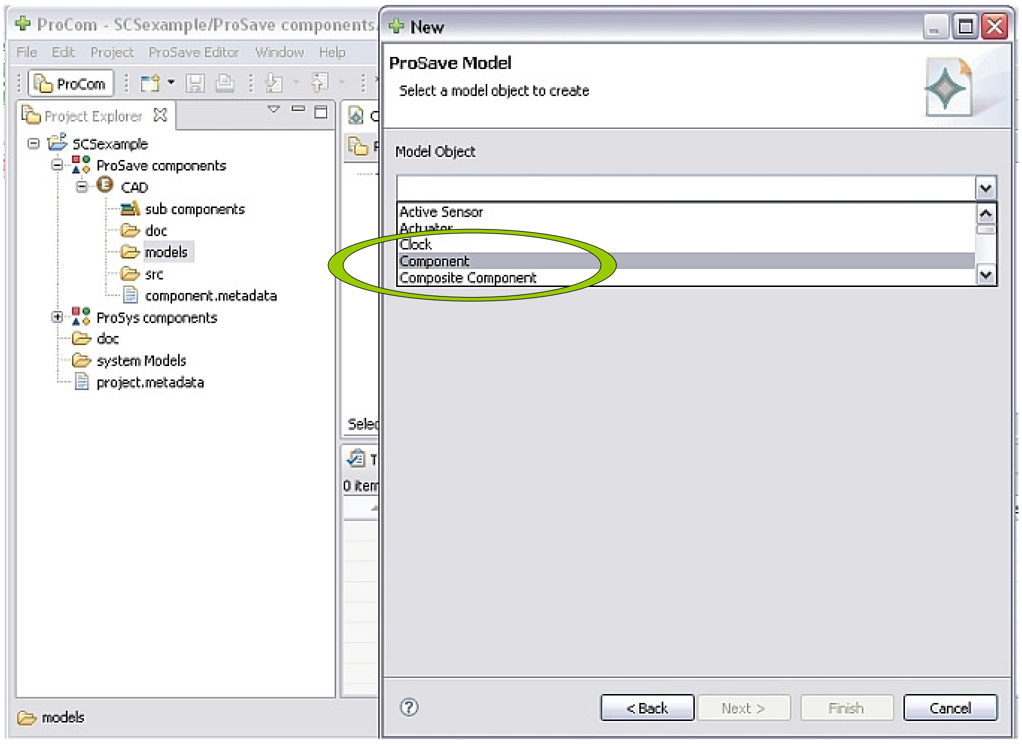


Figure 10. Choosing model object

- Finally, click Finish.
- The newly created model should appear in "models" folder of the component (Figure 11.).

3.5 Modeling a component using Progress-IDE tree-editor

After you have created the model file for a component, you can edit it using Progress-IDE tree-editor. This editor allows you to define:

- Interface of the component by adding ports and services.
- Realization of the component by specifying realization type, adding sub-component instances etc.
- Connections between components.

If you successfully created the model file as it was described in previous section (Section 3.4), then the created model was automatically opened in tree-editor.
Otherwise, to open the file in tree-editor you have to:

- Right-click on model file (Figure 11.).

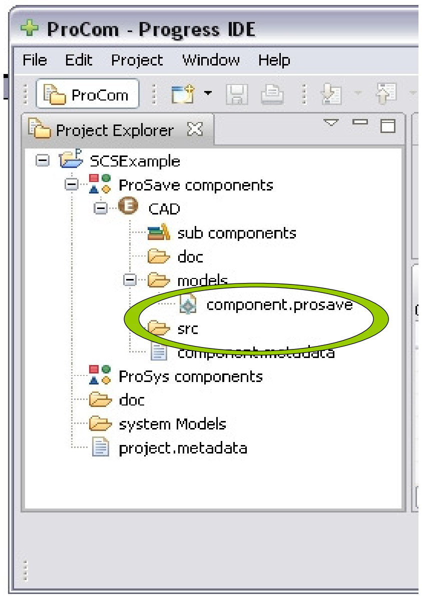


Figure 11. Model file

- From pop-up menu choose Open With → ProSave Model Editor OR ProSys Model Editor (Depends on component type, ProSave or ProSys).

A screenshot of Progress-IDE and tree-editor is shown on Figure 12.

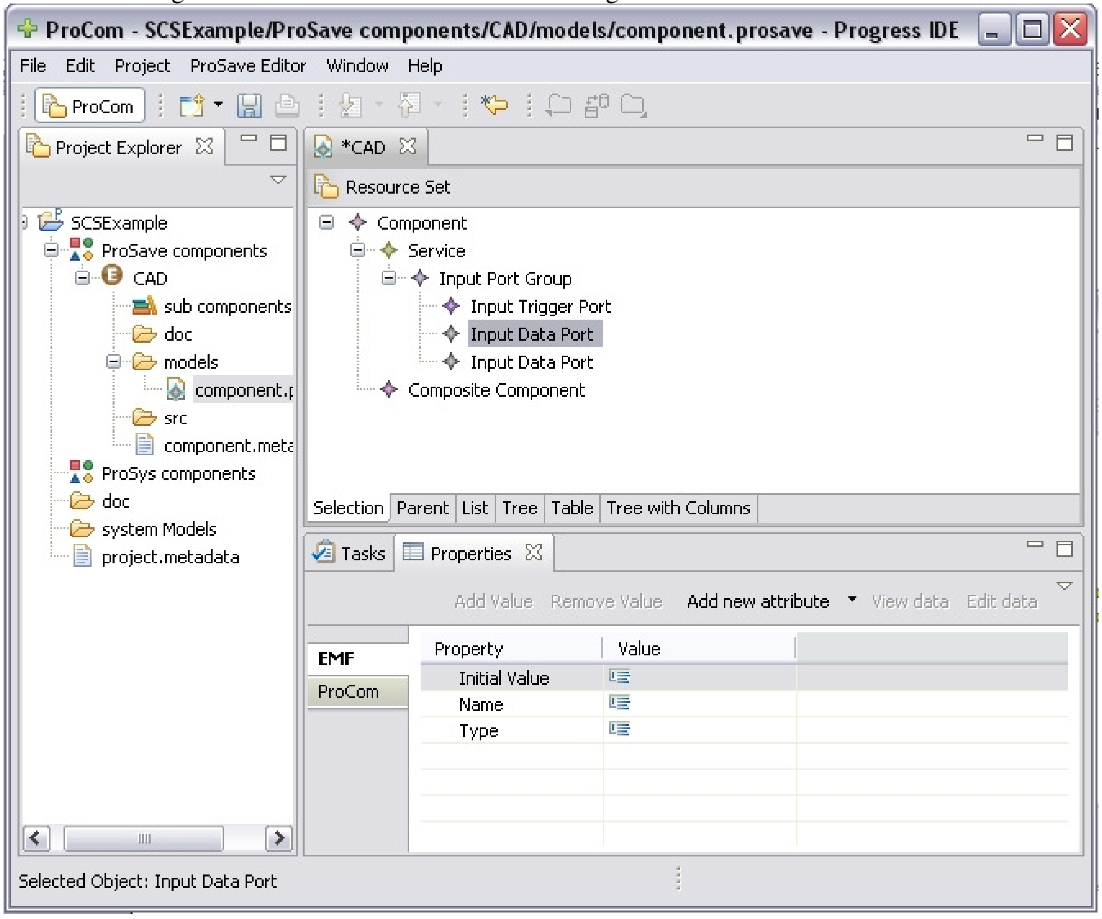


Figure 12. Progress-IDE with tree-editor

In tree editor a component is presented as a set of nodes placed hierarchically. In example at Figure 12., you can see that this component has one service. That service has one input port group, which has one trigger port and two data ports. Also the component is realized as Composite Component.

Tree-editor is used by right-clicking on desired node and choosing what you want to add to the model. For example:

- If you want to add new service to the component, right-click on Component node (top node) and choose New Child → Service.
- If you want to add new input trigger port to some input port group, right-click on desired port group and choose New Child → Input Trigger Port.

Similar is for all other elements that you wish to add. Properties of all elements (for example name or data type of a port) are edited in Properties view in Eclipse (bottom area in Figure 12.).

If you want to add a sub-component to component which is realized as composite you have to click on realization node (in example above, it is *Composite component* node), and then choose New Child → Subcomponent Instance. After that you have to define component that implements this instance (in other words to choose the component that will be instantiated). This is done in Properties view as it is shown in Figure 13., by choosing from a list of all components that exist in the project.

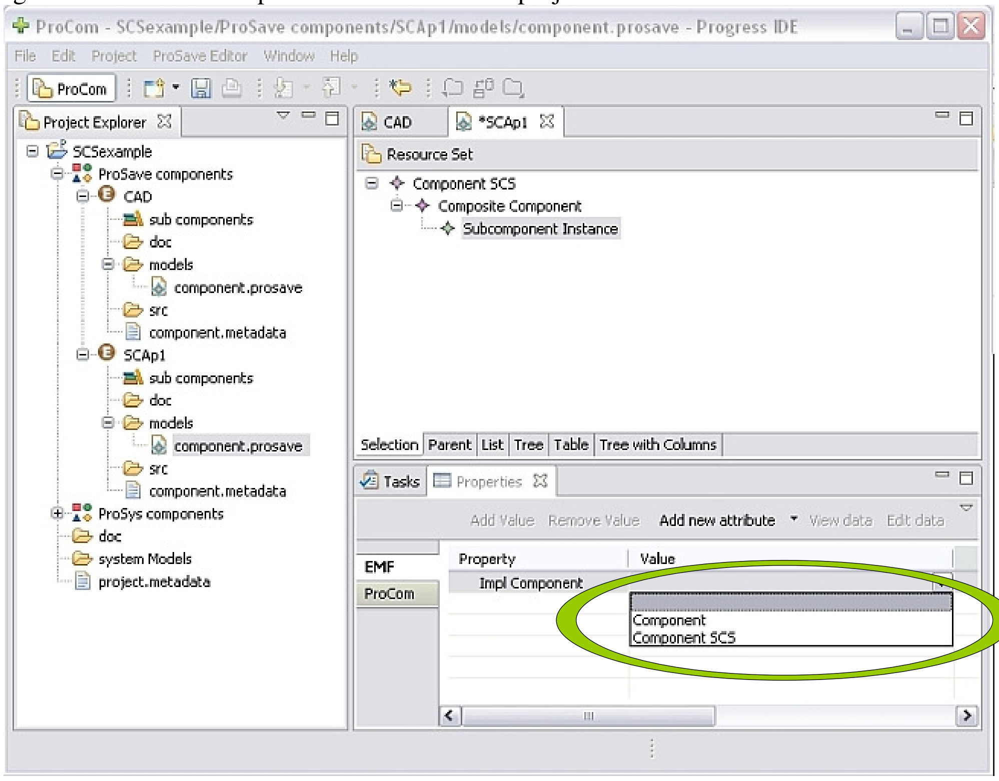


Figure 13. Adding subcomponent instance