Papyrus Guidelines

TR-515 v1.3-info
July 2018

Disclaimer

Open Networking Foundation
1000 El Camino Real, Suite 100, Menlo Park, CA 94025
www.opennetworking.org

Important note

This Technical Recommendations has been approved by the OIMT Project TST but has not been approved by the ONF board. This Technical Recommendation has been approved under the ONF publishing guidelines for 'Informational' publications that allow Project technical steering teams (TSTs) to authorize publication of Informational documents. The designation of '-info' at the end of the document ID also reflects that the project team (not the ONF board) approved this TR.

Table of Contents

List of Figures

List of Tables

Document History

Version	Date	Description of Change
1.0	March 13, 2015	Initial version
1.1	Nov. 30, 2015	Version 1.1 A summary of main changes between version 1.0 and 1.1 is contained in section 10.1.
1.2	Sept. 20, 2016	Version 1.2 A summary of main changes between version 1.1 and 1.2 is contained in section 10.2.
1.3	July 2018	Version 1.3 A summary of main changes between version 1.2 and 1.3 is contained in section 10.3.

1 Introduction

This Technical Recommendation has been developed within IISOMI (**I**nformal **I**nter-**S**DO **O**pen **M**odel **I**nitiative) and is published by ONF.

IISOMI is an open source project founded by UML model designers from various SDOs like ETSI NFV, ITU-T, MEF, ONF and TM Forum.
The goal is to develop guidelines and tools for a harmonized modeling infrastructure that is not specific to any SDO, technology or management protocol and can then be used by all SDOs.
The deliverables are developed in an open source community under the "Creative Commons Attribution 4.0 International Public License".

This document defines the guidelines that have to be taken into account during the creation of a protocol-neutral UML (Unified Modeling Language) information model using the Open Source tool Papyrus. The Guidelines are not specific to any SDO, technology or management protocol. References are written in an SDO-neutral form, enclosed by "{ }"; each SDO will have an SDO-specific "mapping table" for these placeholders:

- XxxModel
  Any Papyrus UML model.
- {XxxModel}
  Identifies the URI of the GitHub repository containing the XxxModel.

2 References

[1] Papyrus Eclipse UML Modeling Tool (https://www.eclipse.org/papyrus/)

[2] Eclipse (https://eclipse.org/)

[3] Unified Modeling Language™ (UML®) (http://www.uml.org/)

[4] IISOMI 514 "UML Modeling Guidelines"https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/UML_Modeling_Guidelines_Version_1-1.pdf
Last published version → v1.3, July 2018

[5] IISOMI Wiki (https://wiki.opennetworking.org/display/OIMT/IISOMI)

[6] Open Source SDN Project EAGLE (https://github.com/OpenNetworkingFoundation/EAGLE-Open-Model-Profile-and-Tools)

[7] OpenModelProfile (https://github.com/OpenNetworkingFoundation/EAGLE-Open-Model-Profile-and-Tools/tree/ToolChain/UmlProfiles)

[8] Eclipse Gendoc (https://www.eclipse.org/gendoc/)

3 Abbreviations

API	Application-Programming-Interface
ARO	Association Resources Online™
ASCII	American Standard Code for Information Interchange
DS	Data Schema
IDE	Integrated Development Environment
IISOMI	Informal Inter-SDO Open Model Initiative
IM	Information Model
IMP	Information Modeling Project (ONF Services Area)
ITU-T	International Telecommunication Union – Telecommunication Standardization Sector
JSON	JavaScript Object Notation
NBI	NorthBound Interface
OF	Open Flow
OT	Optical Transport
RSA	Rational Software Architect (UML tool from IBM)
SDO	Standards Developing Organization
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XML	Extensible Markup Language
WG	Working Group

4 Documentation Overview

This document is part of a suite of guidelines. The location of this document within the documentation architecture is shown in Figure 4-1 below:

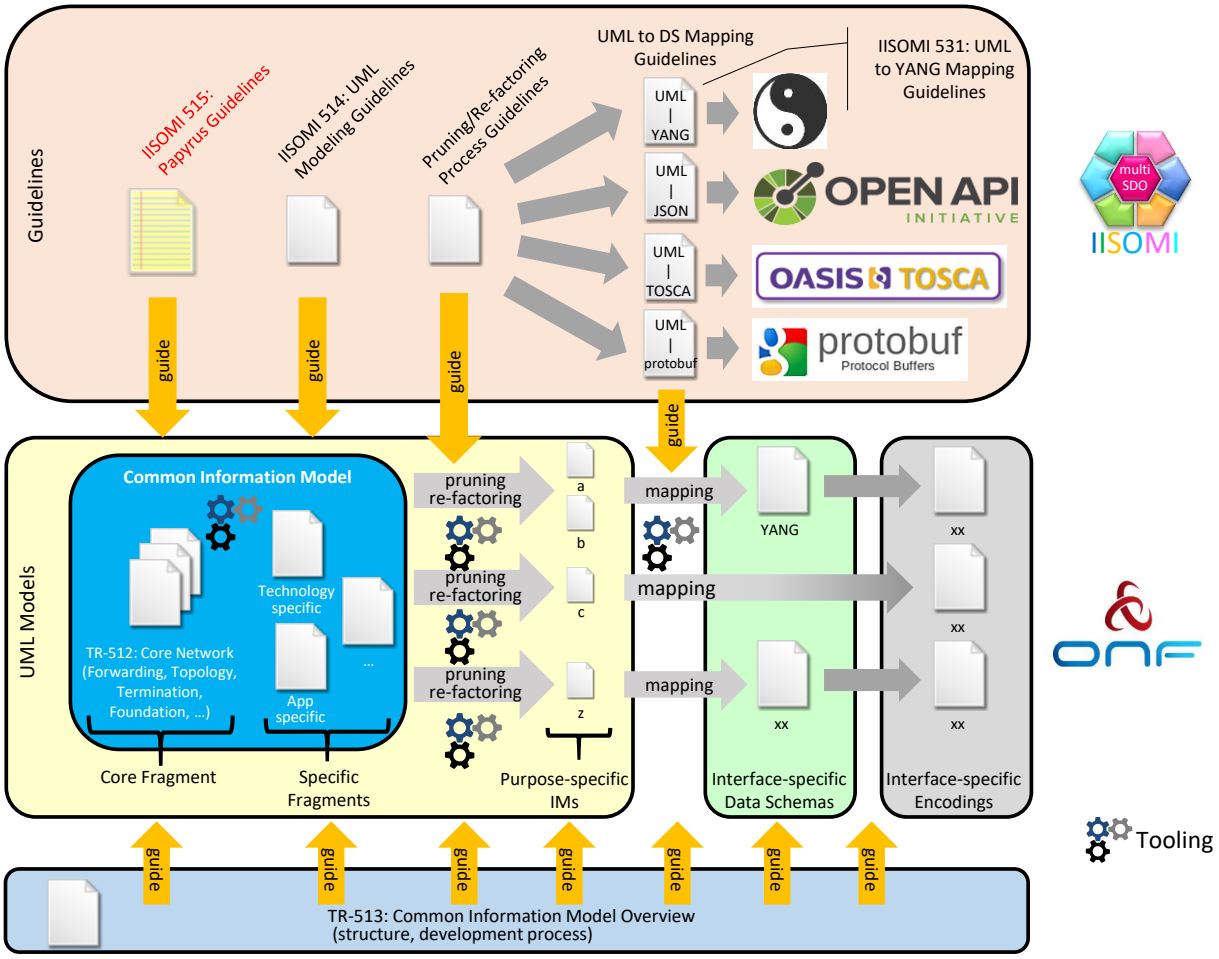


Figure 4-1: Specification Architecture

5 Getting Papyrus Running

5.1 Applied Tool Versions

The Open Source UML tool Papyrus is a plug-in for the Open Source integrated development environment (IDE) Eclipse.

Applied tool versions for this guideline:

- Eclipse version 4.7.2 “Oxygen”
- Papyrus version 3.2.0 RC4
- Gendoc version 0.6.0

This clause explains how to get Papyrus running on the target PC and how to import a model. Working with sub-models is described in clause 6.

5.2 Downloading Papyrus

5.2.1 Downloading Latest Version

This is the easiest way of getting Papyrus. In this download approach, the Papyrus package already includes Eclipse.

Installing the latest available versions of Eclipse and Papyrus is **not recommended** since this is not tested and the migration of models created in previous versions may cause errors.

Note: This download approach provides the nightly builds of Papyrus version 3.x.x which is "For advanced users and pioneers …" and not the version that is used at present!

The latest version of Papyrus can be downloaded as a complete software package from the Papyrus homepage: https://eclipse.org/papyrus/download.html. The software package contains Papyrus and also the corresponding Eclipse software; i.e., Eclipse need not be available in advance on the target PC.

Papyrus is available for various platforms:

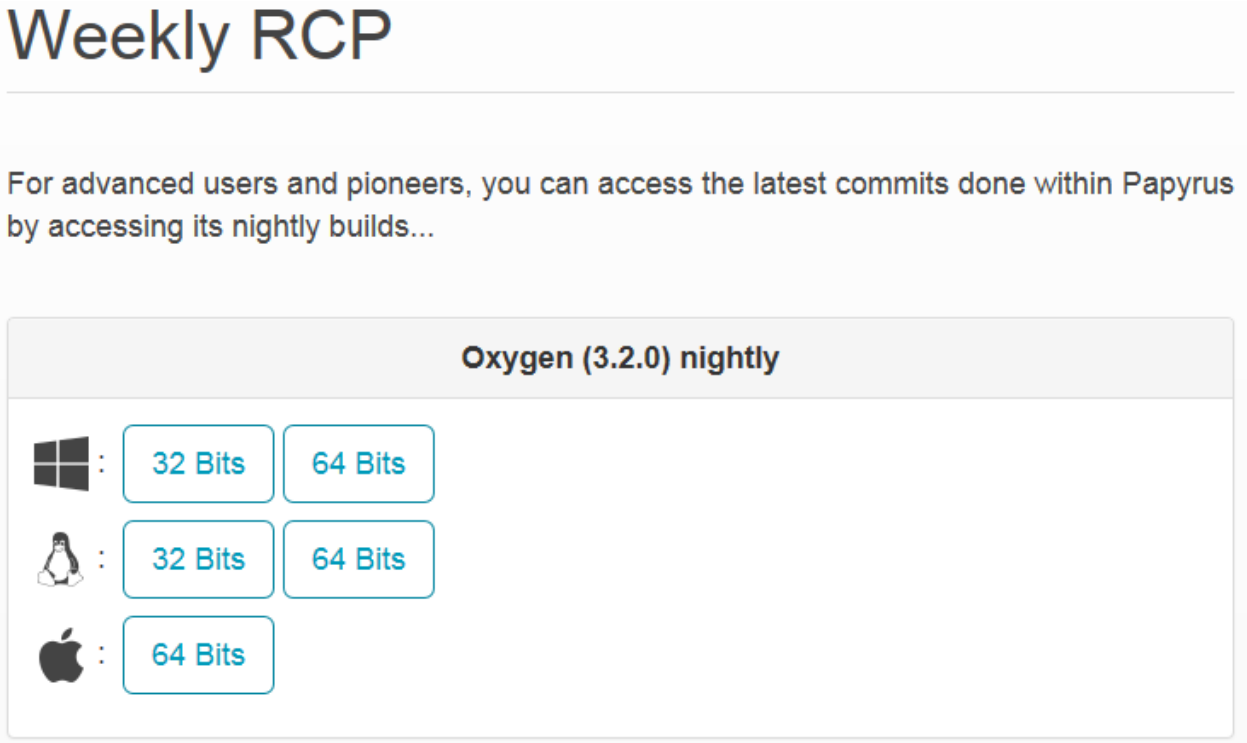


Figure 5-1: Papyrus Download Page

Papyrus release (3.x.x) requires a 1.8 compatible JVM.

Once downloaded, just extract the downloaded zip-file (i.e., Papyrus (Eclipse) need not be "installed" on the target PC):

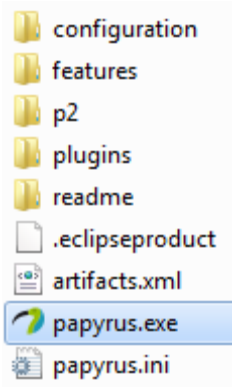
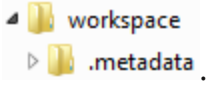


Figure 5-2: Content of the Papyrus Folder after Extracting the Zip-file

To launch Papyrus, double-click on the papyrus.exe file.

Figure 5-3: Initial Papyrus Welcome Icon

After launching Papyrus, a default workspace folder is created in the home directory (…/users/<users name>/). The workspace configuration information is contained in the .metadata folder: (which is automatically created):



Any empty (need not be empty but is recommended) folder - anywhere - can be used as a workspace-folder. The workspace can be selected during the launch of Papyrus.

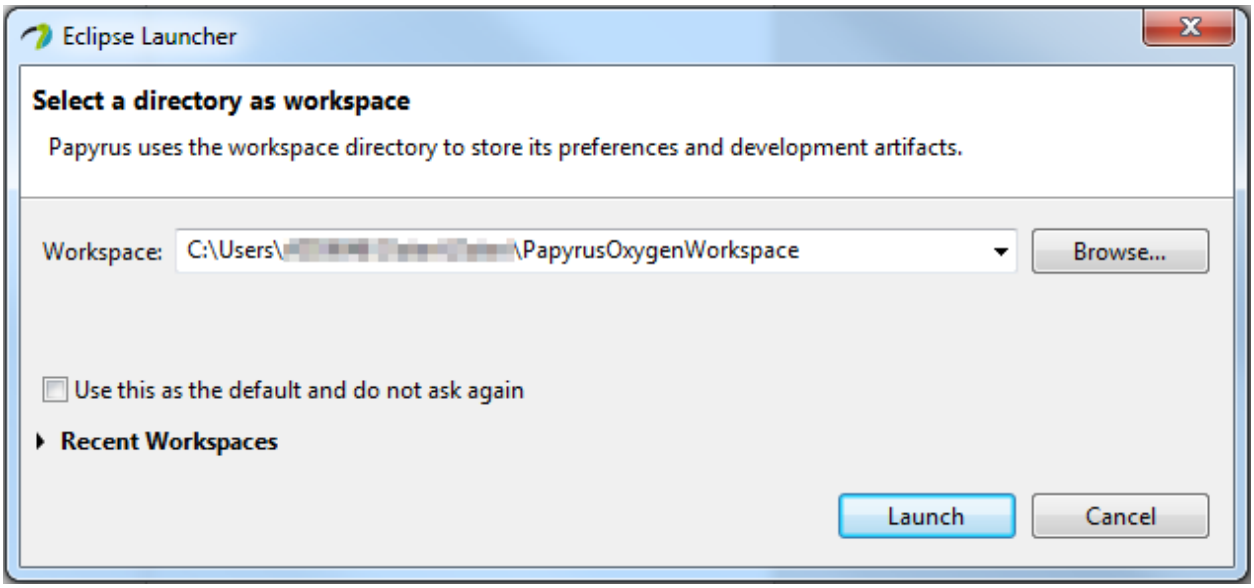


Figure 5-4: Selecting a Workspace

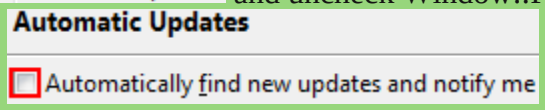
Continue at clause 5.3.

5.2.2 Downloading Applied Version

In this approach, one needs to download and launch Eclipse and then add the Papyrus plug-in.

This is the recommended approach since it makes sure that the agreed version of Eclipse and Papyrus get "installed" on the PC.

Warning:
Avoid to use the Help:: Check for Updates and uncheck Window::Preferences::Install/Update



which may install other versions of Eclipse and Papyrus!

Eclipse "Oxygen" Modeling Tools package version 4.7.2 can be downloaded from here: https://www.eclipse.org/downloads/packages/release/oxygen/2.

Download the "Eclipse Modeling Tools" package.
I.e., not the Standard version.

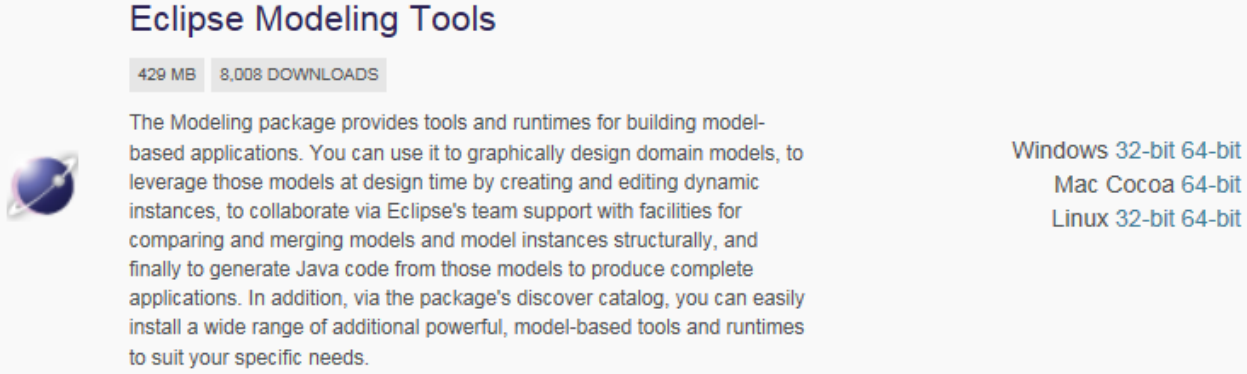


Figure 5.5: Eclipse Oxygen Modeling Tools Download Page

Eclipse Oxygen requires a 1.8 compatible JVM.

Once downloaded, just extract the downloaded zip-file (i.e., Eclipse need not be “installed” on the target PC):

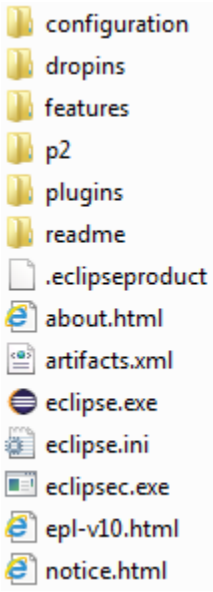


Figure 5.6: Content of the Eclipse Folder after Extracting the Zip-file

To launch Eclipse, double-click on the eclipse.exe file.

Figure 5-7: Initial eclipse Welcome Icon

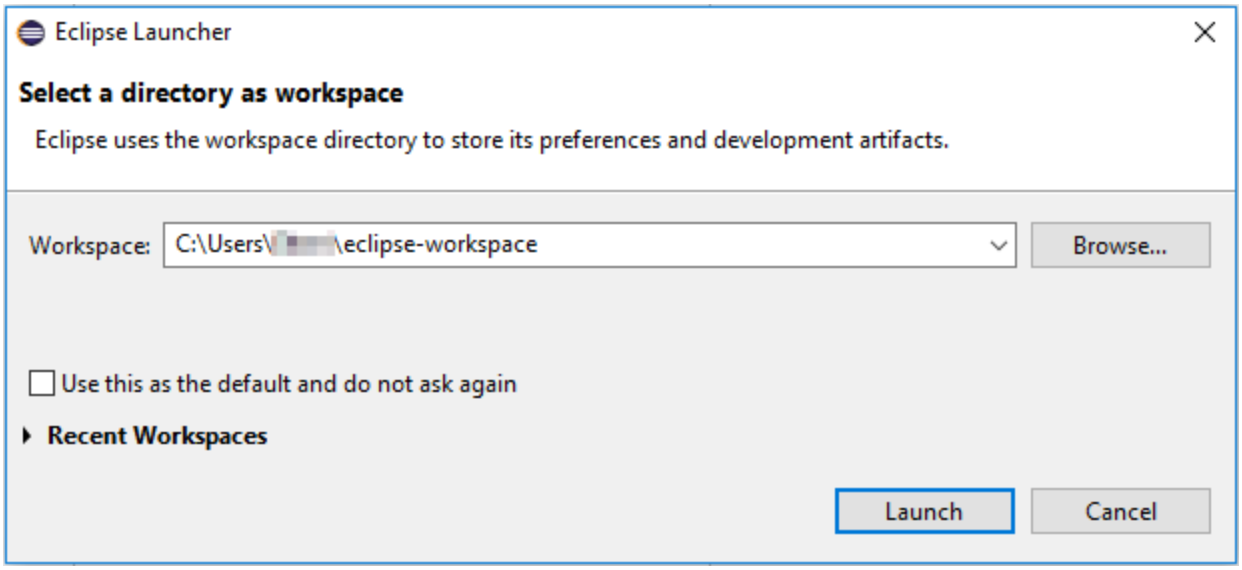
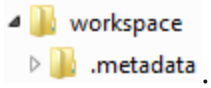


Figure 5-8: Selecting a Workspace

After launching Eclipse, a default eclipse-workspace folder is created in the home directory (…/users/<users name>/). The workspace configuration information is contained in the .metadata folder:



Any empty folder (need not be empty but is recommended) - anywhere - can be used as a workspace-folder. The workspace can be selected during the start of Eclipse.

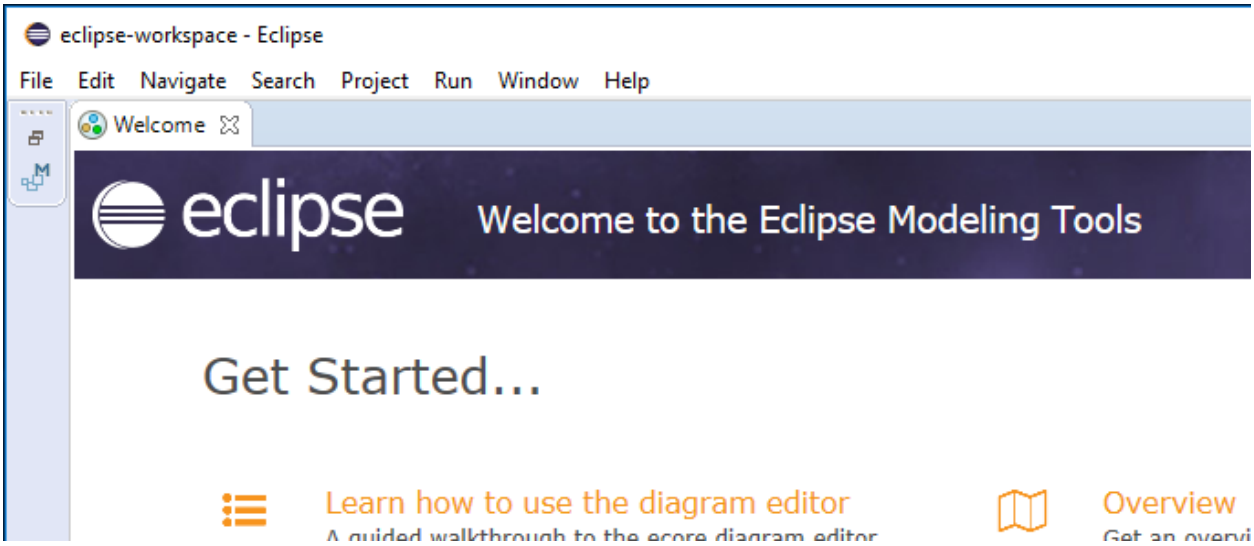


Figure 5.9: Initial Welcome Page of Eclipse

Close the Welcome tab at the upper left corner.

Click Help and select About Eclipse to check that the correct version 4.7.2 is installed:

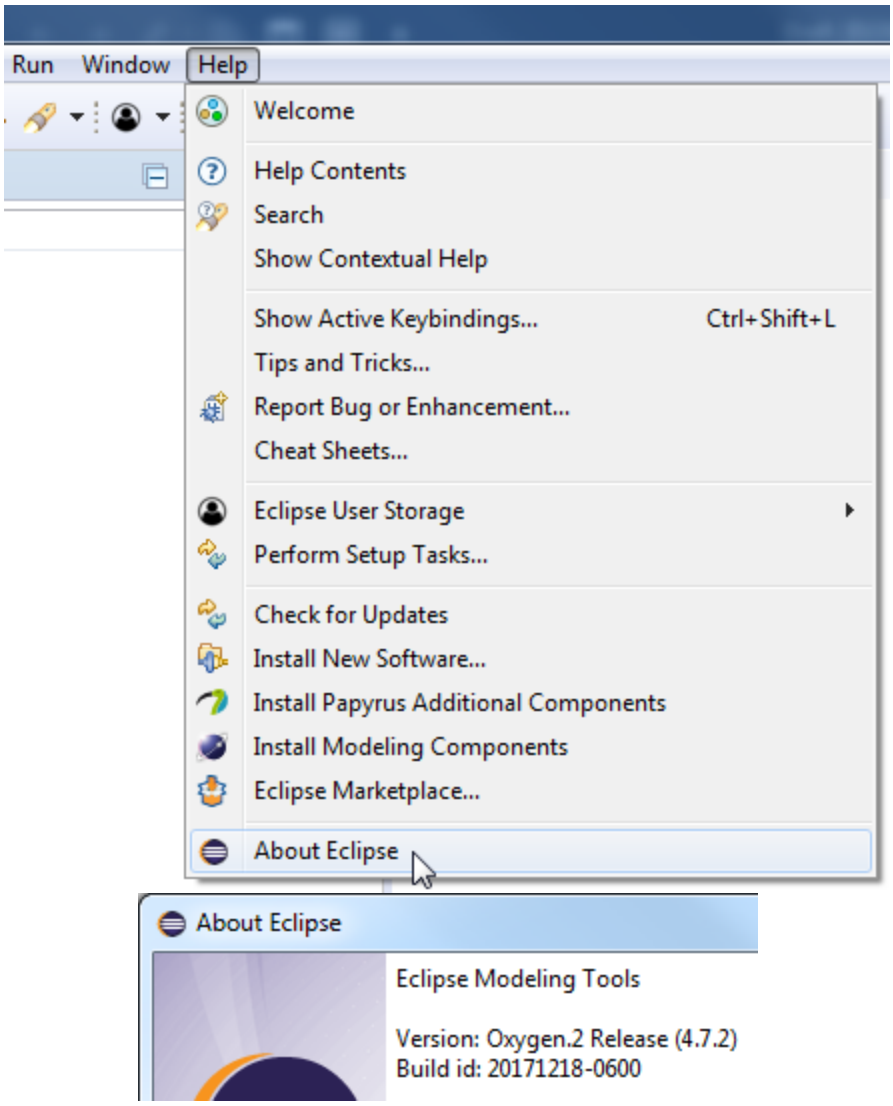


Figure 5.10: Eclipse Version

Eclipse is now ready for use.

To add Papyrus, click menu Help and then Install New Software...:

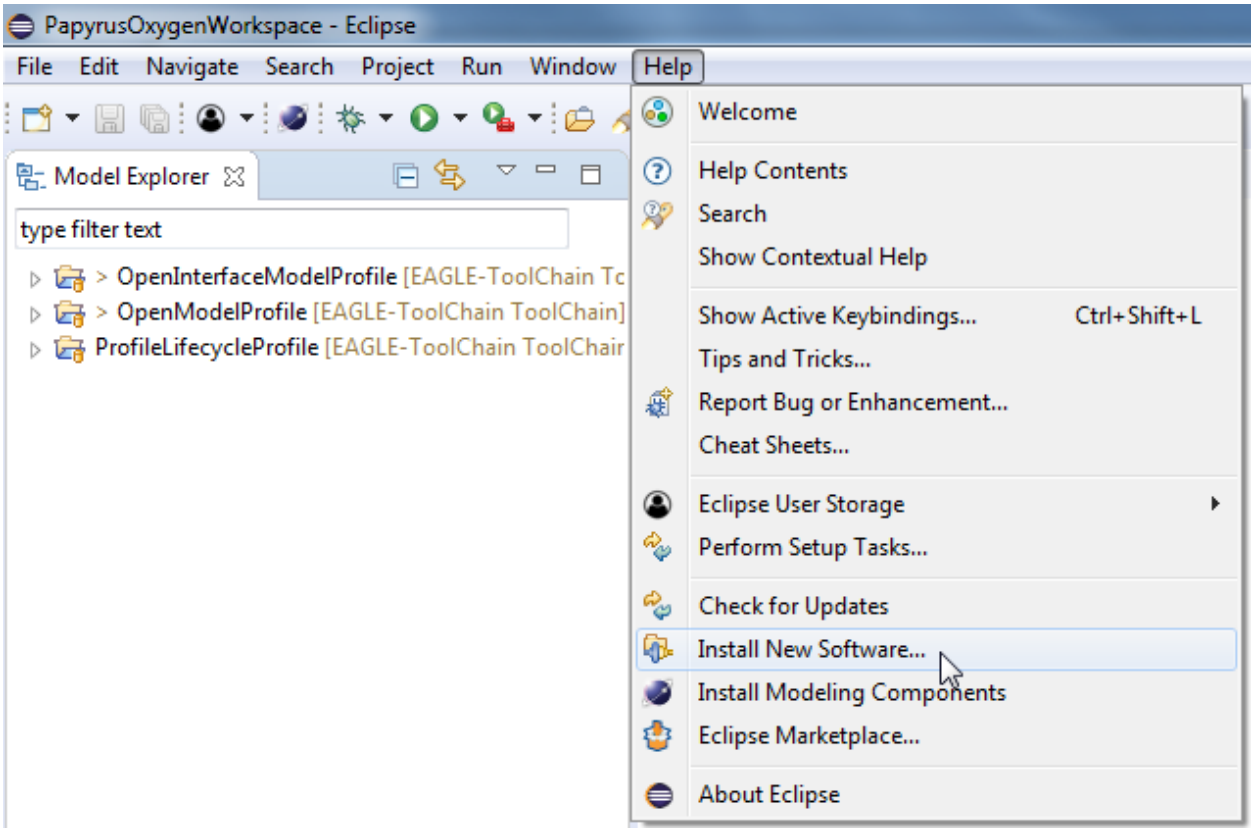


Figure 5.11: Installing Papyrus (1)

Click Add... in the Install window, then add a name and the update site (http://download.eclipse.org/modeling/mdt/papyrus/updates/releases/oxygen) in the Add Repository window:

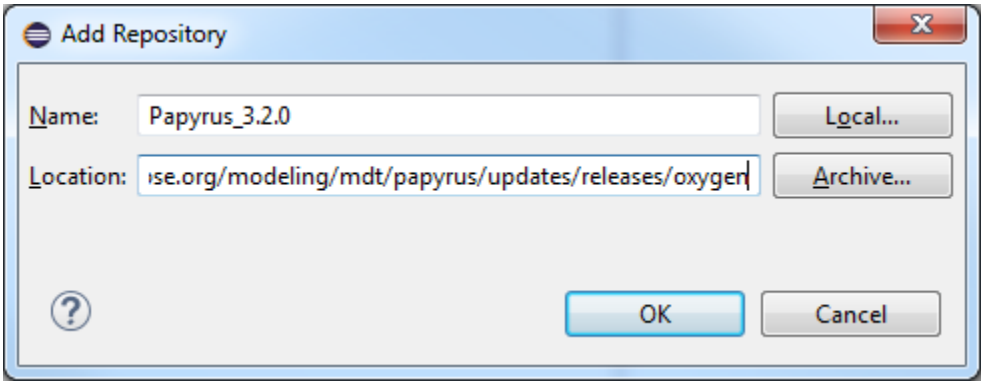


Figure 5.12: Installing Papyrus (2)

Click OK and then make sure you do not check Show only the latest versions of available software and select Papyrus for UML 3.2.0.201712060842 in the Install window:

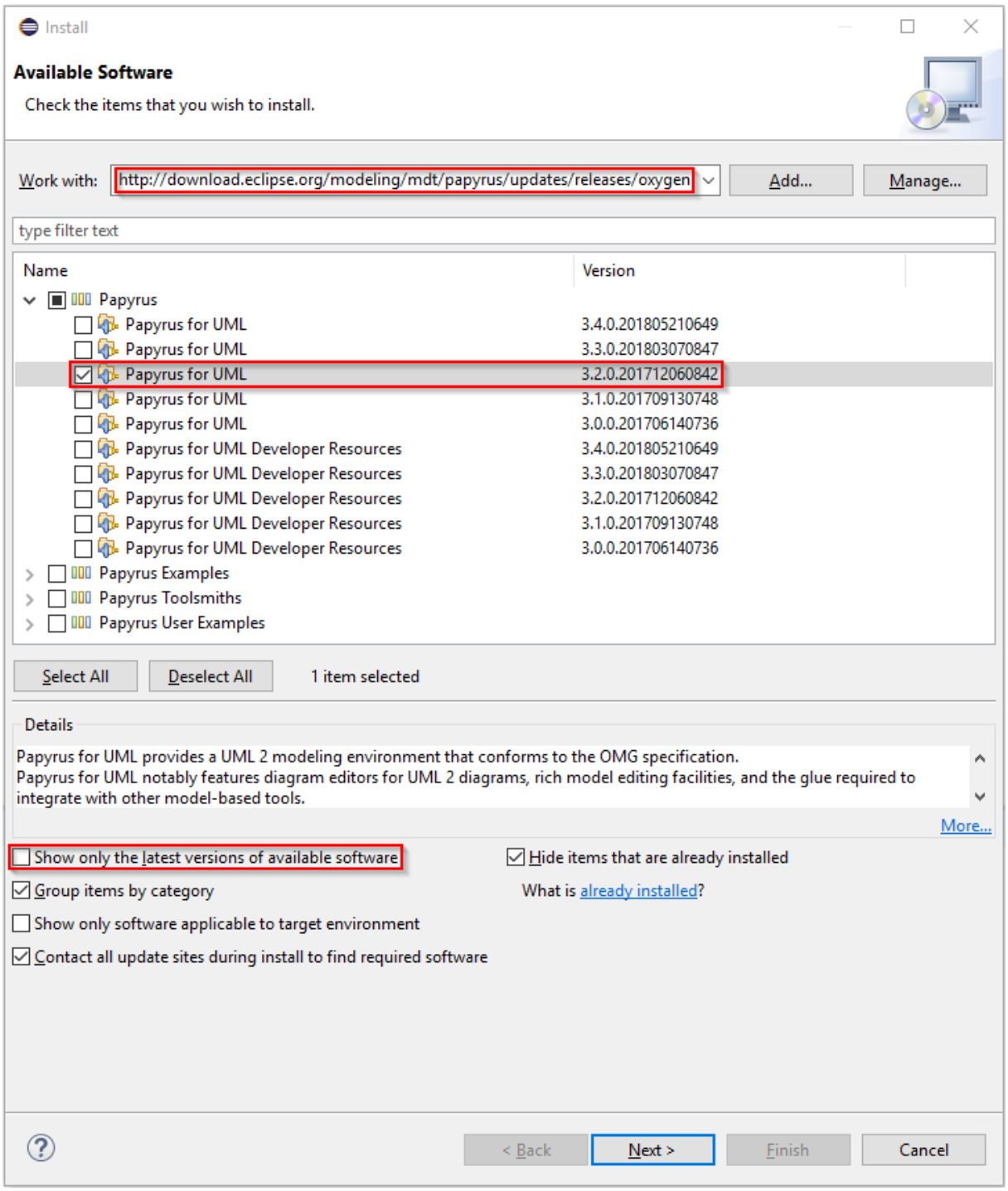


Figure 5.13: Installing Papyrus (3)

Click Next >.

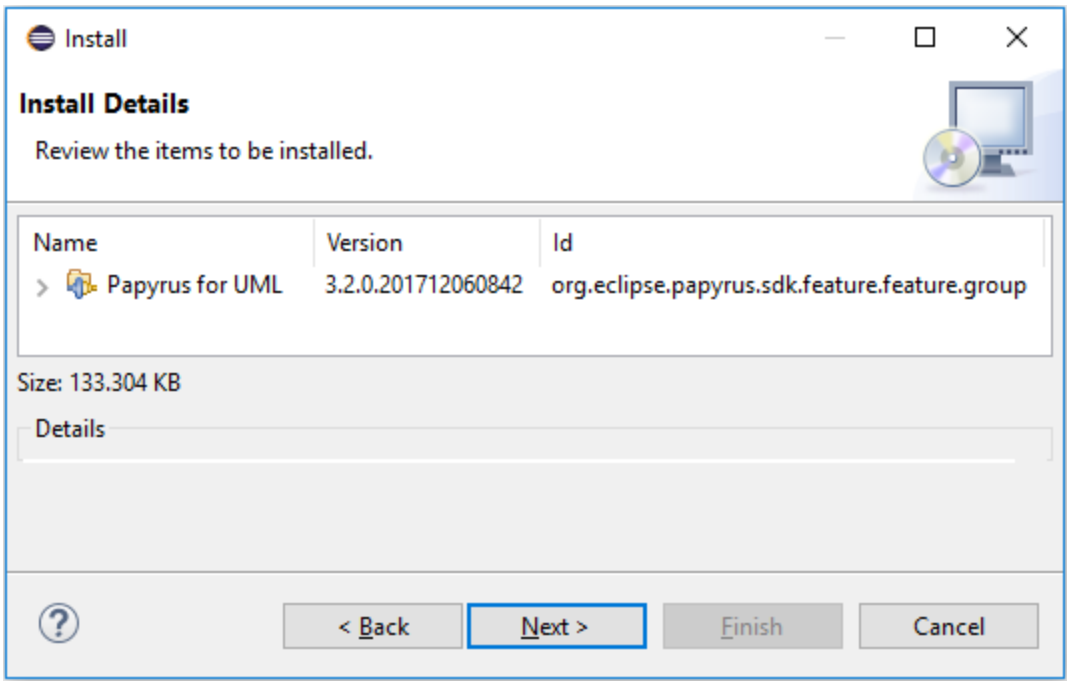


Figure 5.14: Installing Papyrus (4)

The correct Papyrus version 3.2.0.20171206 can be checked in the appearing Install window.

Click Next >, accept the license agreement I accept the terms of the license agreement and then click Finish:

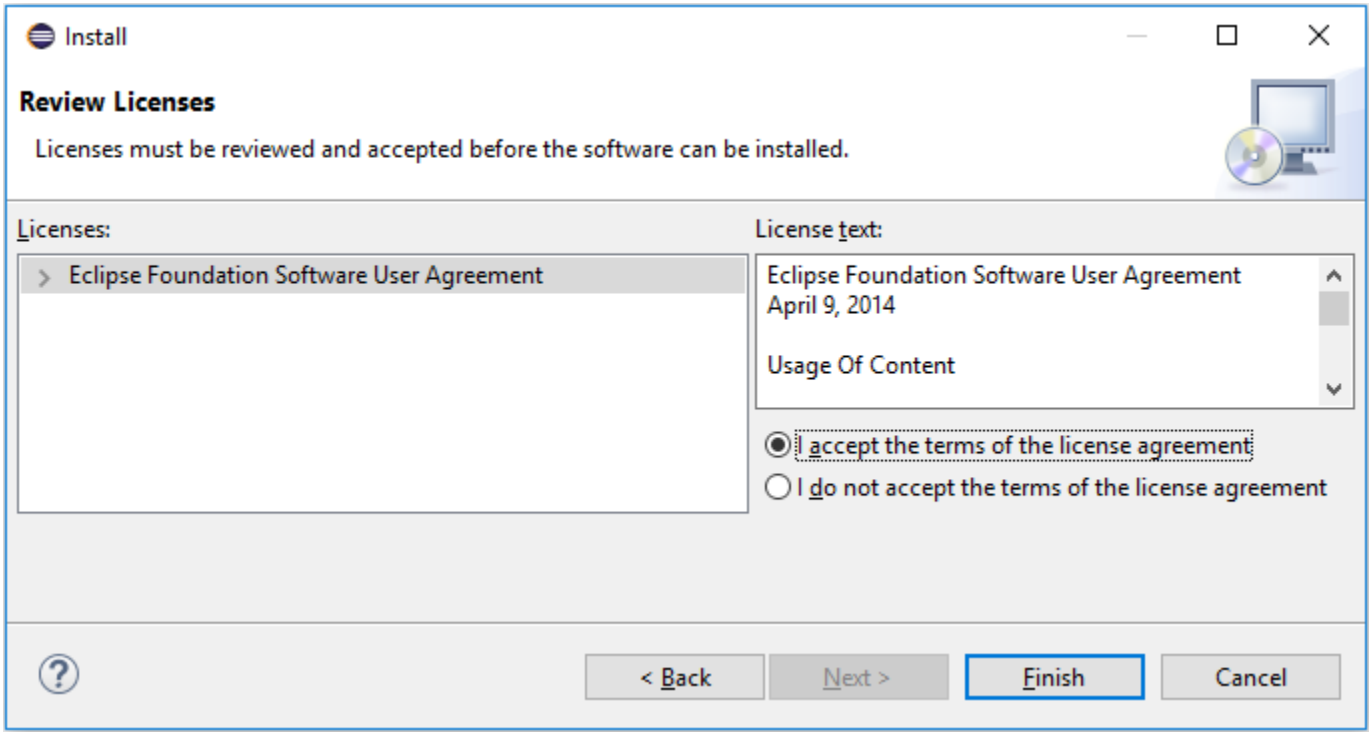
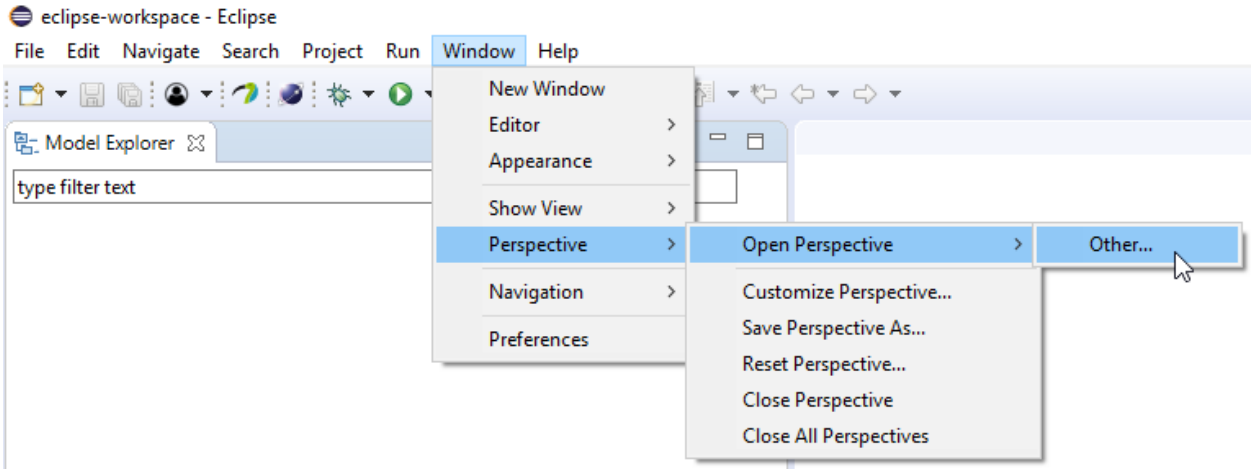


Figure 5.15: Installing Papyrus (5)

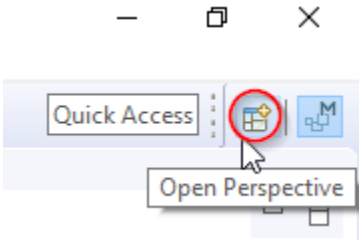
After Papyrus is installed successfully, Eclipse need to be restarted.

After restarting Eclipse switch to the *Papyrus Perspective* (if not already displayed) by

- either going via menu Window, *Perspective, Open Perspective, Other...*:



- or by clicking the Open Perspective-button ( ) at the top right side of the screen:



and then selecting Papyrus:

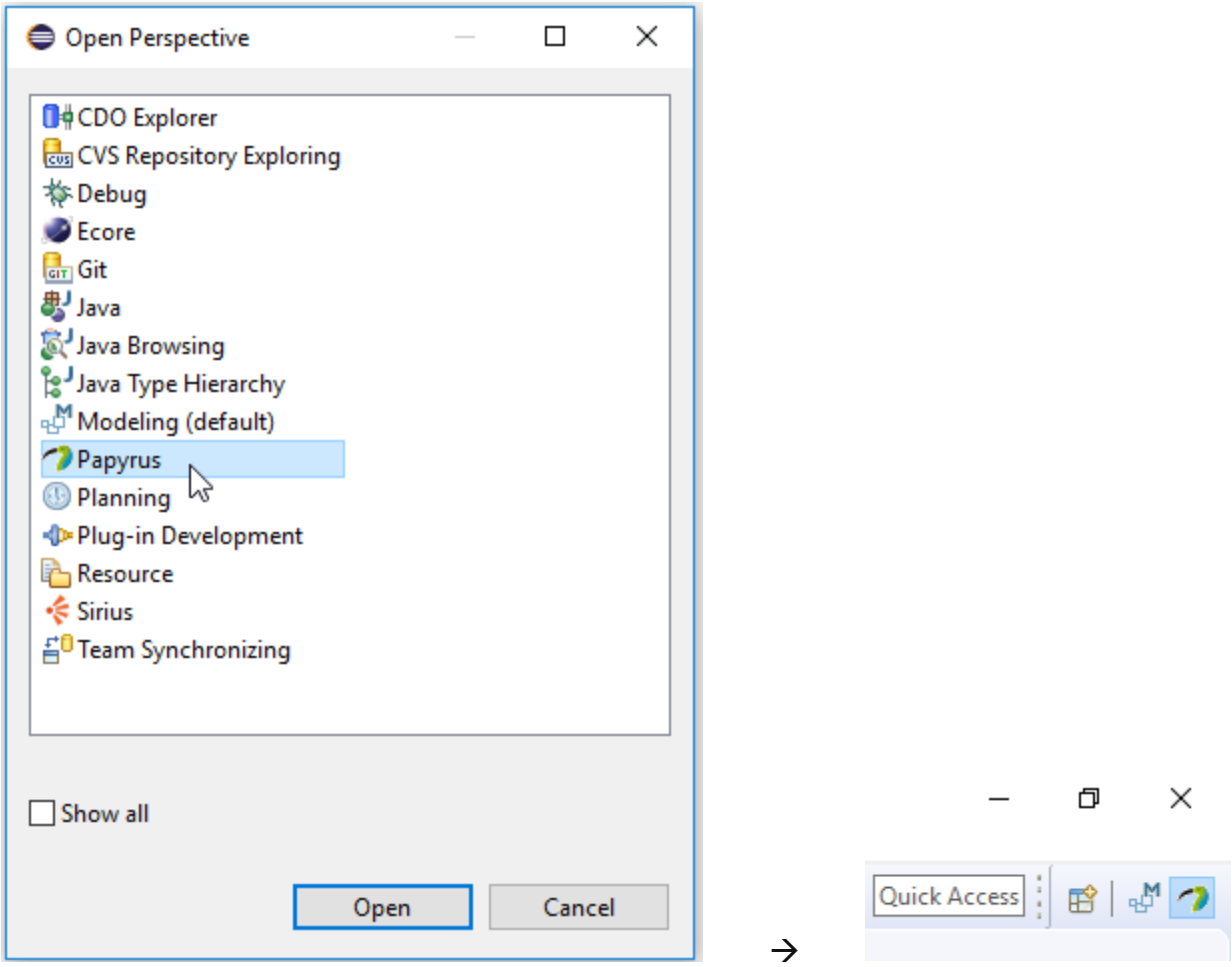


Figure 5.16: Open Papyrus Perspective

Papyrus content can only be viewed properly if the computer display is set to 100 %.

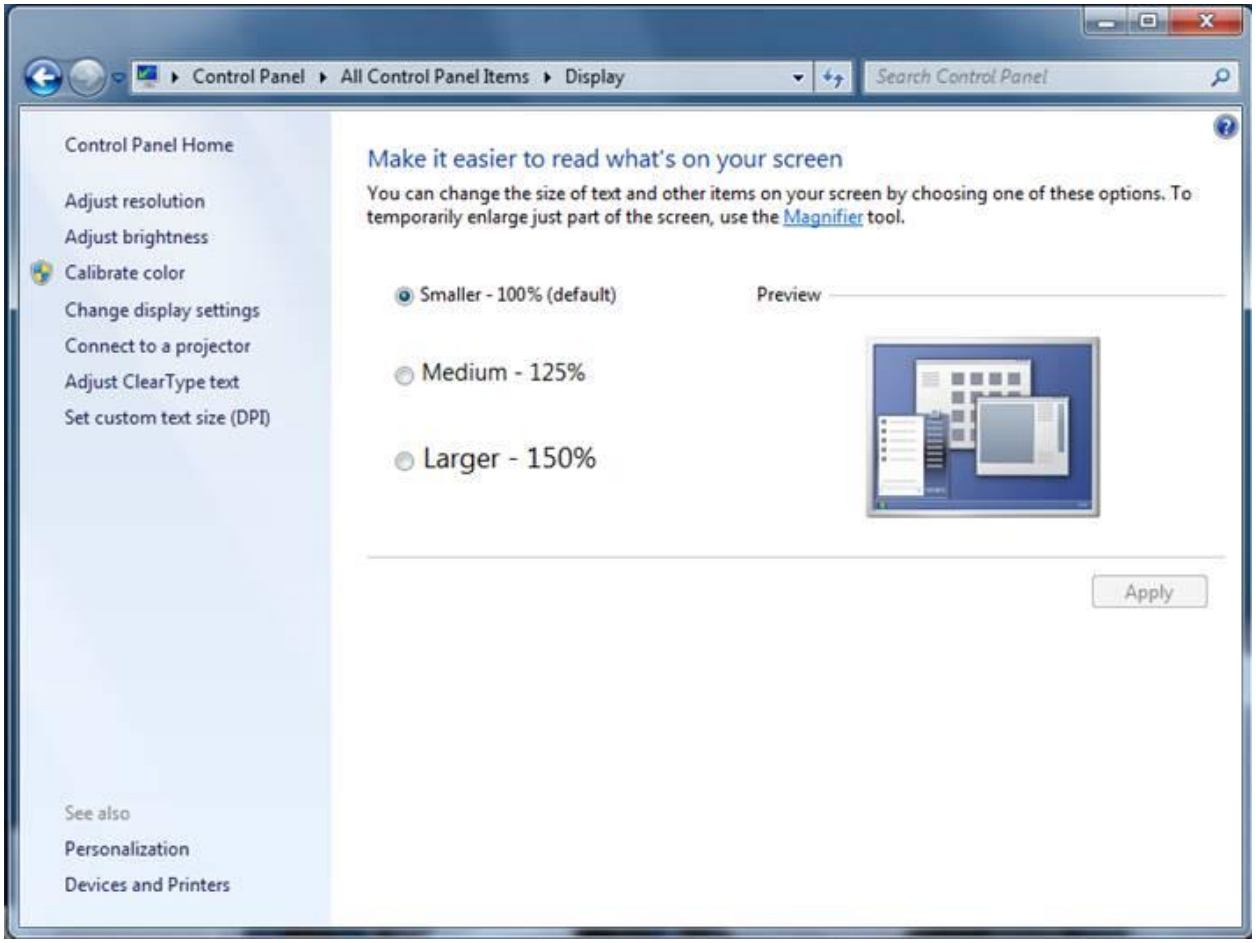


Figure 5-17: Required Display Setting

5.3 Papyrus Overview

The outline of the Papyrus Perspective presents different panes/panels and toolbars:

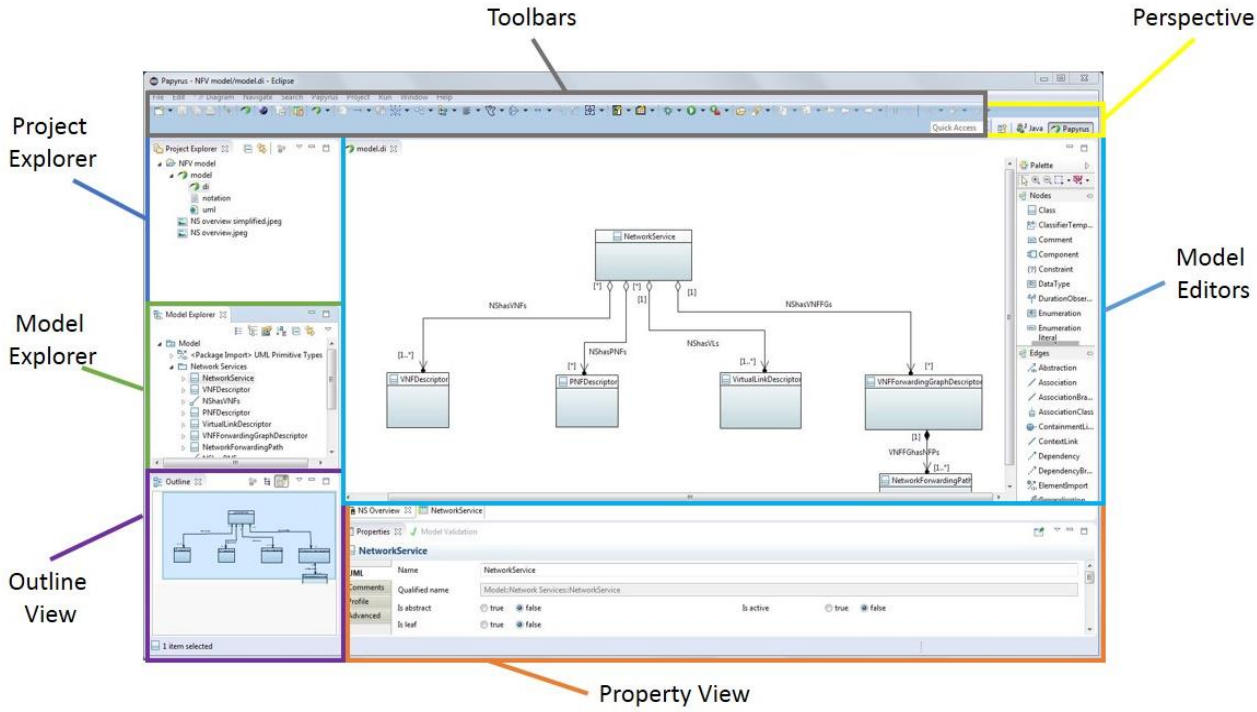


Figure 5.18: Outline of Papyrus Perspective

- **Perspective:** it provides the modeling context and the layout of the windows, as well as the definition of the menus and toolbars. For using Papyrus, it shall always be set to "Papyrus".
- **Project Explorer:** it is used to manage Papyrus projects at system level. It provides a view on the model files in the workspace folder.
- **Model Explorer:** it provides the internal view of the model selected in the Project Explorer. It is a tree-based model editor for the whole model. If the Project Explorer contains several models, only one at a time can be selected to be edited in the Model Explorer.
- **Model Editors:** it allows graphic edition of the model via diagrams. Class diagrams are the only type of diagram mandated.
- **Property View:** it is a form-based editor allowing to view and edit the detailed property of a given element.
- **Outline View:** it provides a read-only view of the model presented in the Model Editor.

5.4 Gendoc Plugin Installation

The Gendoc plugin is used in conjunction with a document template. The template contains instructions that enable generation of a Microsoft Word document. The document can include extracts from the model such as diagrams, class definitions, attribute definitions along with their stereotypes etc. as well as figures and text directly entered into the template. This clause provides instructions on how to install Gendoc followed by guidance on construction of Gendoc templates along with example fragments of templates.

Click menu Help and then Install New Software...:

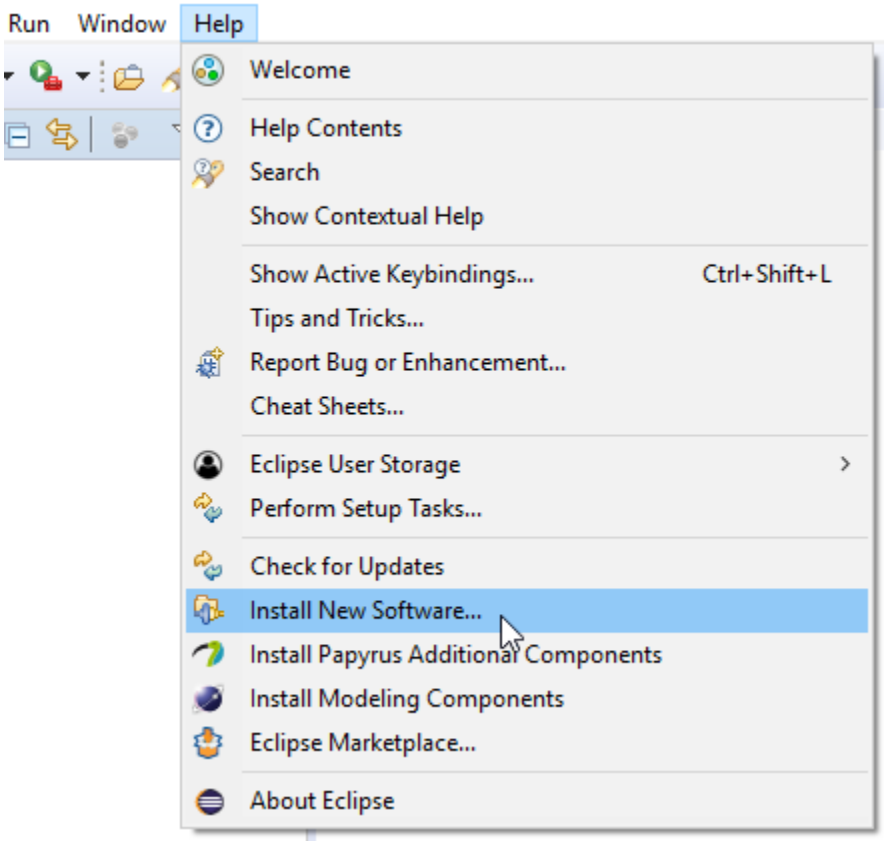


Figure 5-19: Installing Gendoc (1)

Click Add... and enter the Gendoc 0.6.0 update site:
http://download.eclipse.org/gendoc/updates/releases/0.6.0/

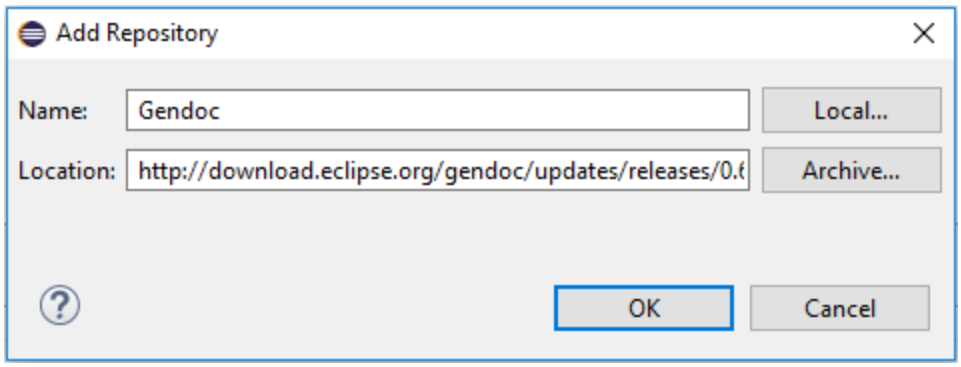


Figure 5-20: Installing Gendoc (2)

Select Gendoc:

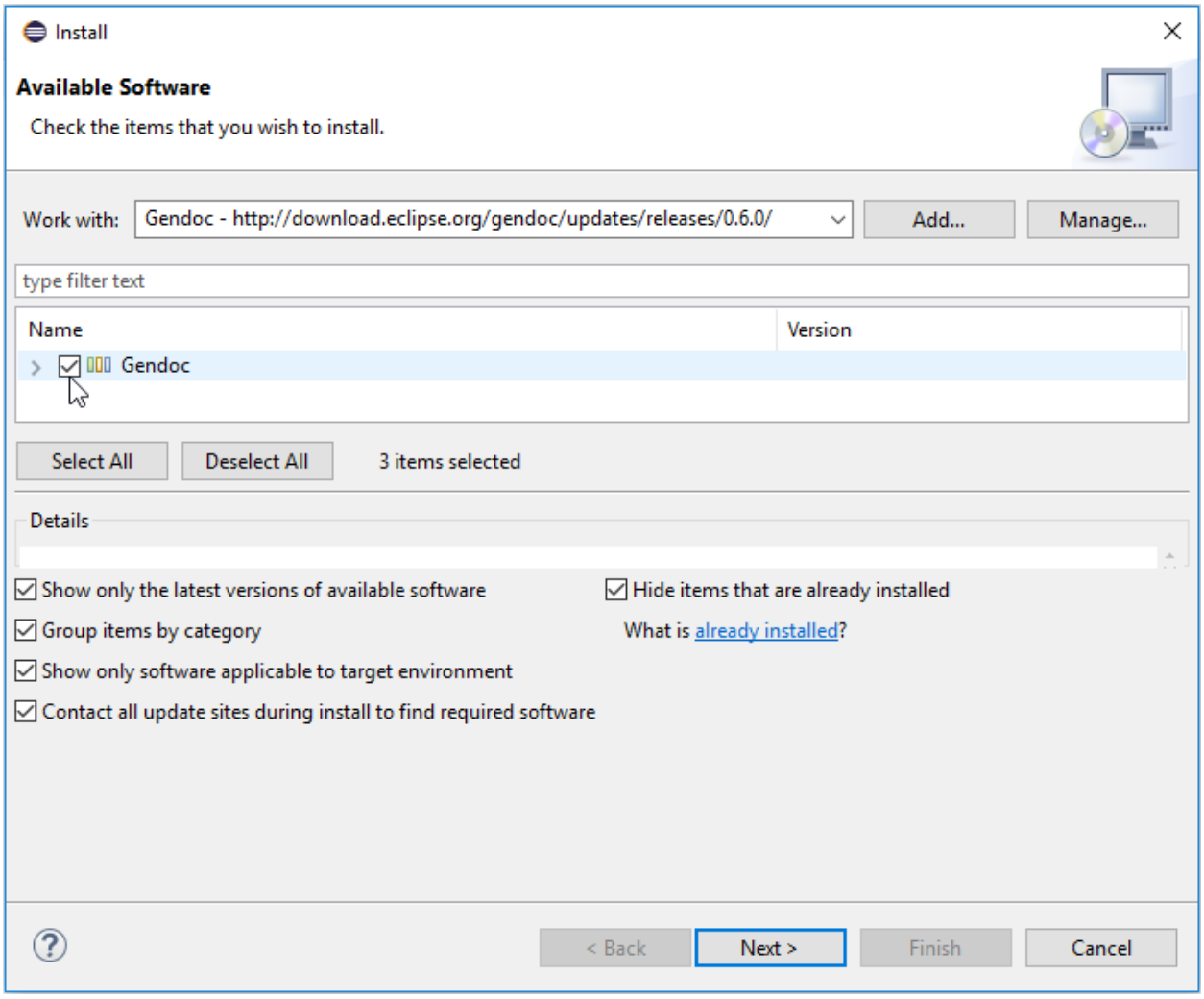
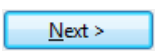


Figure 5-21: Installing Gendoc (3)

Then click  and follow the instructions.

5.5 Importing a Model into Papyrus

The Papyrus Perspective shows a *Project Explorer* and a *Model Explorer*:

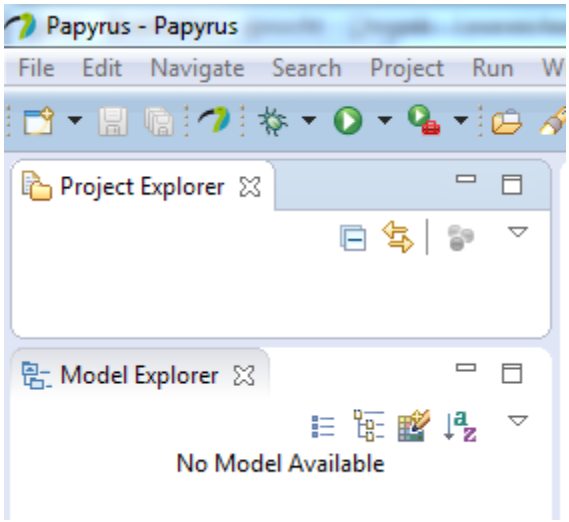


Figure 5-22: Papyrus Project Explorer / Model Explorer

Notes:
Models cannot exist on their own. Every model needs to be contained in a project. A project can contain zero or more models.
The Project Explorer window provides a view on the model files in the workspace-folder.
The Model Explorer window provides the internal view of the model selected in the Project Explorer. The Model Explorer can only handle (edit) one model at a time.

The actual interface specification is contained in the Information Model and the additional properties of the UML artifacts are defined in a Profile Model. It is possible to organize the two models in a single project (*Alternative 1* in the figure below) or in two separate projects (*Alternative 2* in the figure below).

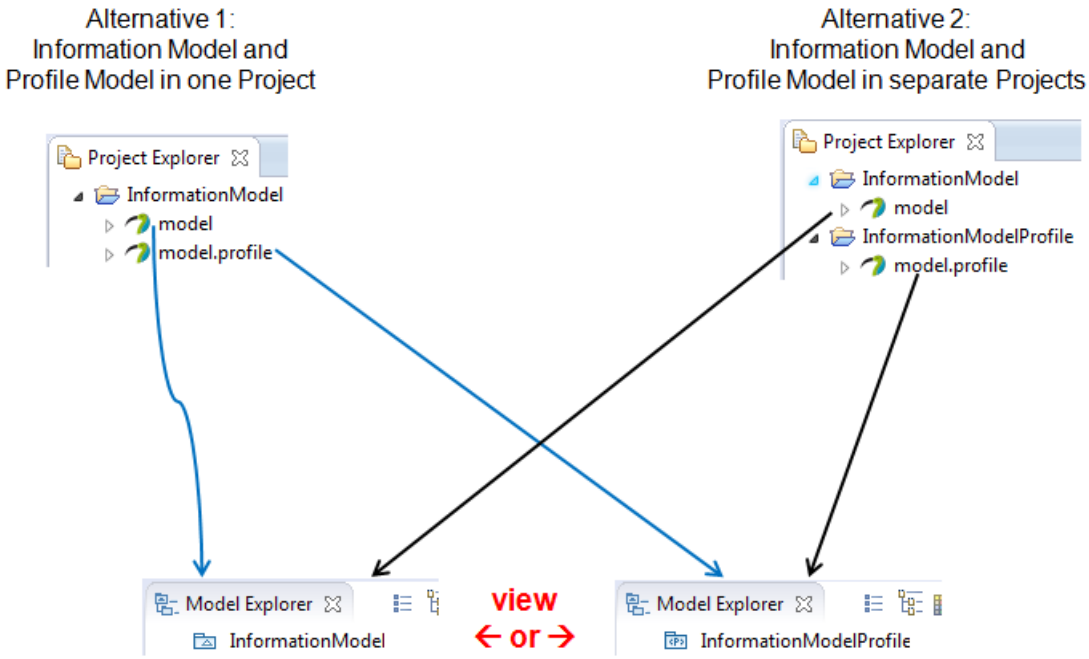


Figure 5-23: Papyrus Model Structure

Clause 6.2 explains how to access the Open Model Profiles in GitHub.

Each project folder contains a .project file. Each model folder – profiles are also models – contains a .di, .notation and .uml file.

The following steps explain how to import a model (profile or model) into Papyrus.

The Profile should be imported first.
It is also possible to import the Model first,
but the Profile shall be available before the model is opened the first time.

Right click in the Project Explorer area opens the menu containing the Import...-button:

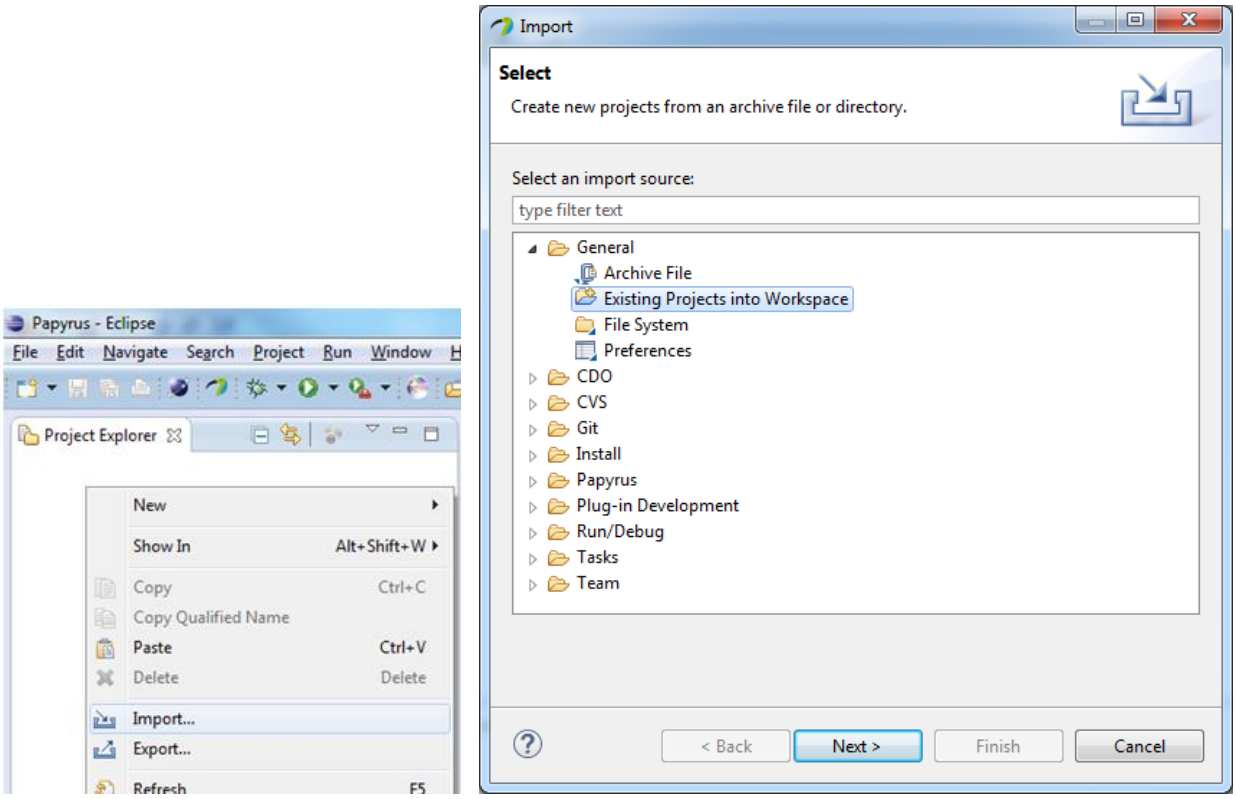


Figure 5-24: Importing a Model (1)

Select the Existing Projects into Workspace option when the profile - that has to be imported - contains a .project file. Otherwise create a new project and then import the profile using the File System option.

Click Next > and then point via Browse...

- to the folder containing the extracted files: Select root directory:
- to the zip-file containing the files: Select archive file:.

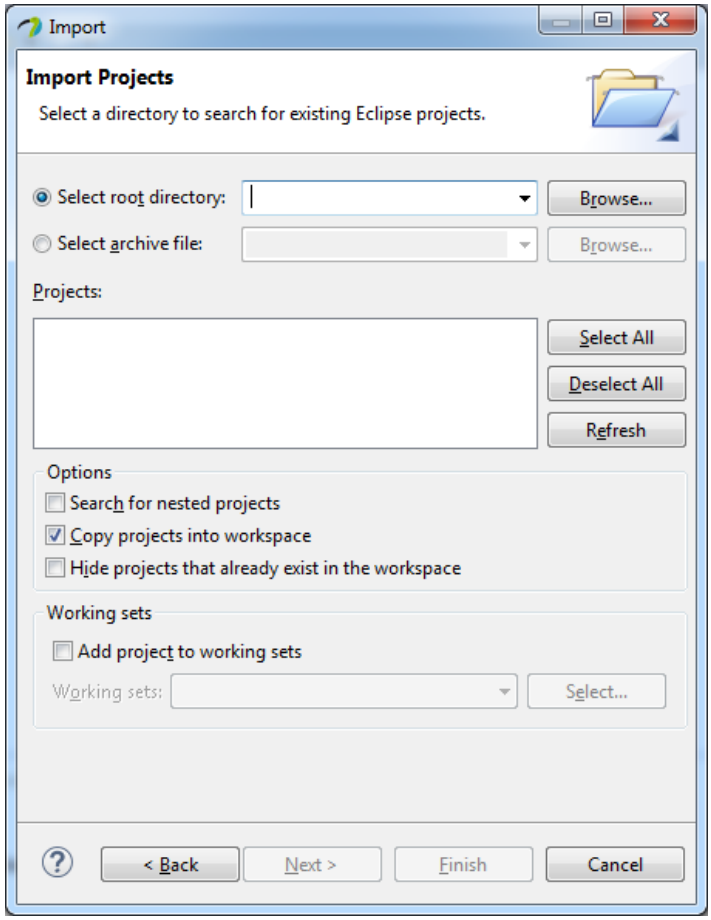
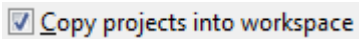
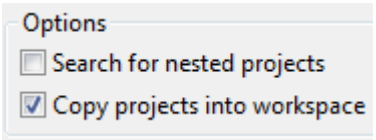
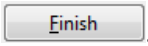


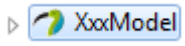
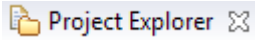
Figure 5-25: Importing a Model (2)

THEN select the option  if the model files shall be copied into the workspace, otherwise Papyrus only creates a pointer and works with the files contained in the extracted folder:



Click  .

Note:
The profile/model files can be located anywhere on the PC. It is not necessary to copy the files into the workspace-folder.

A double click e.g., on  in the  opens the XxxModel in the .

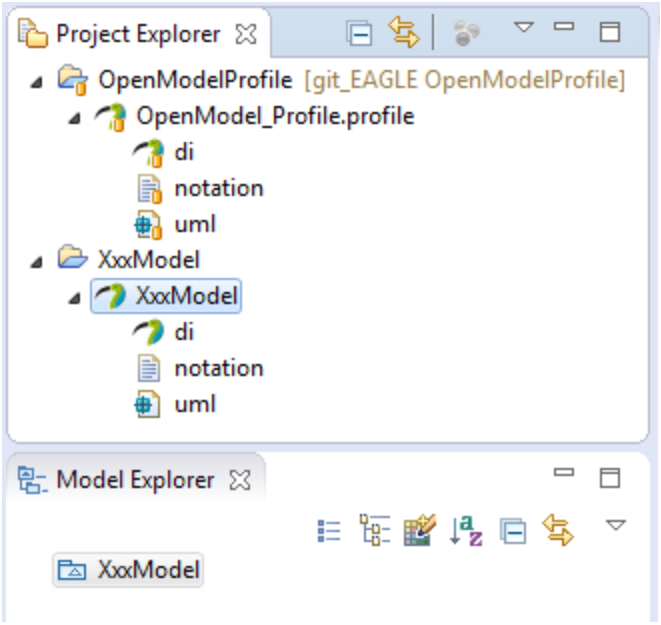


Figure 5-26: Open a Model

Now the model can be used.

5.6 Importing Common Models into a Model

It may be necessary to import the common UML Primitive Types (i.e., Boolean, Integer, String). This can be done by a right-click on the model pacakge XxxModel then going to Import / Import Registered Package and then select UMLPrimitiveTypes:

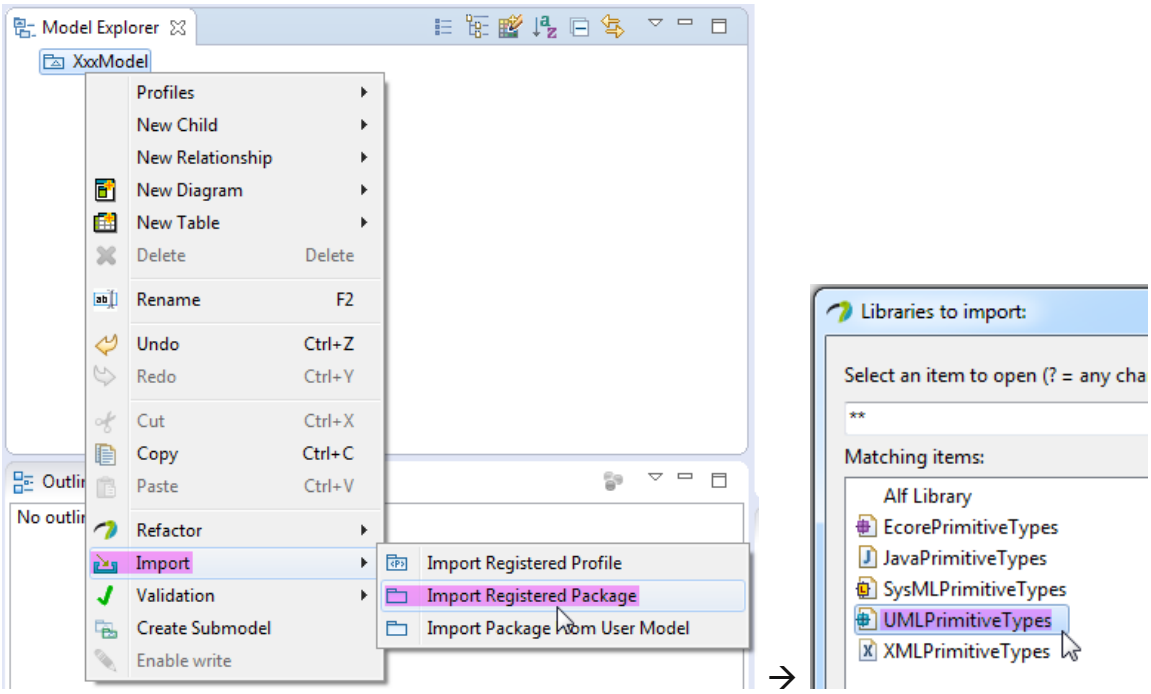


Figure 5-27: Importing UML Primitive Types

It may also be necessary to relate artifacts in the sub-module to artifacts defined in the core model. This can be done by a right-click on the model package XxxModel then via Import and Import Package From User Model select OnfModel [ONFInfoModel develop] / CoreModel.uml:

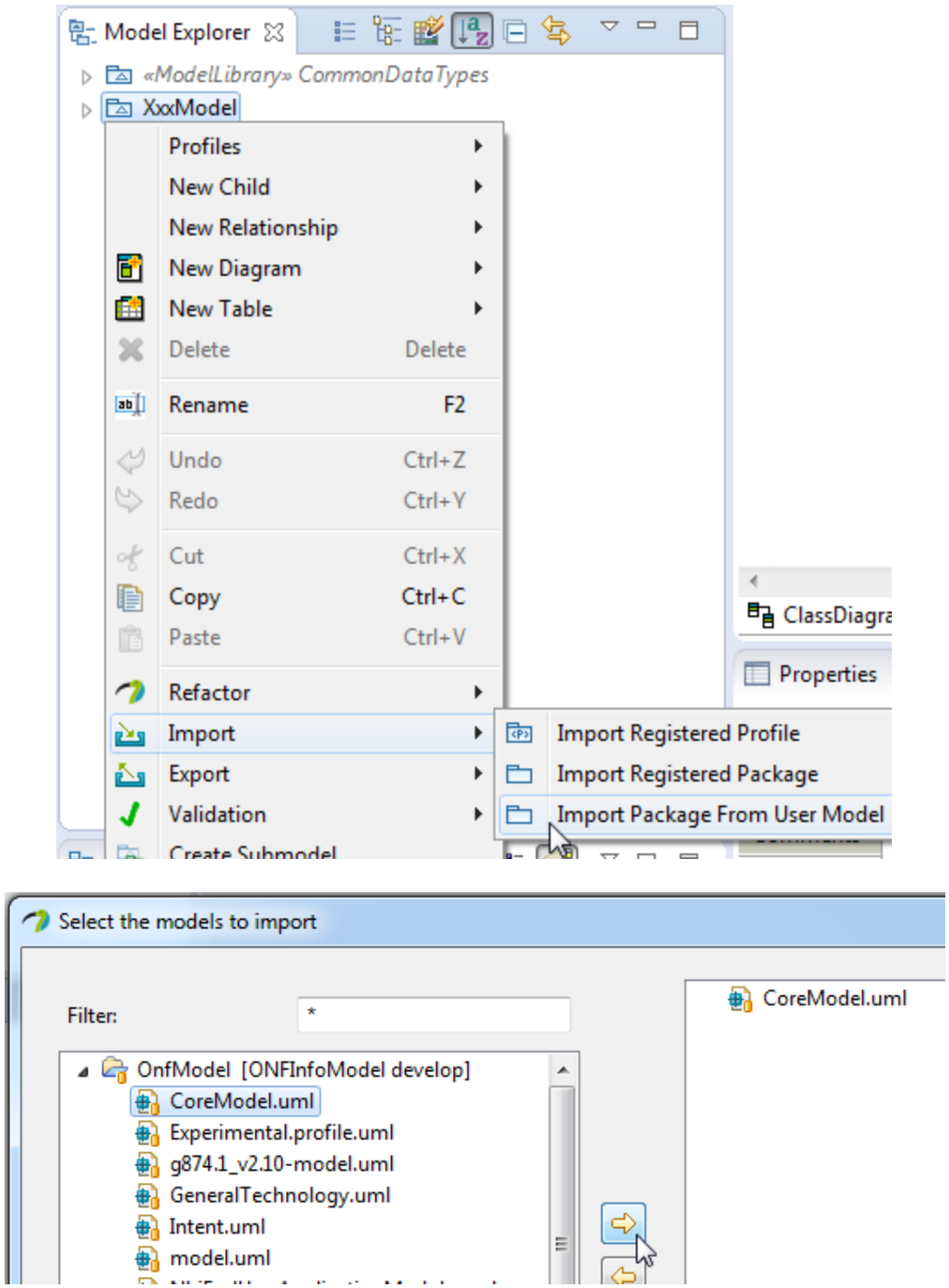


Figure 5-28: Importing Core Model Artifacts

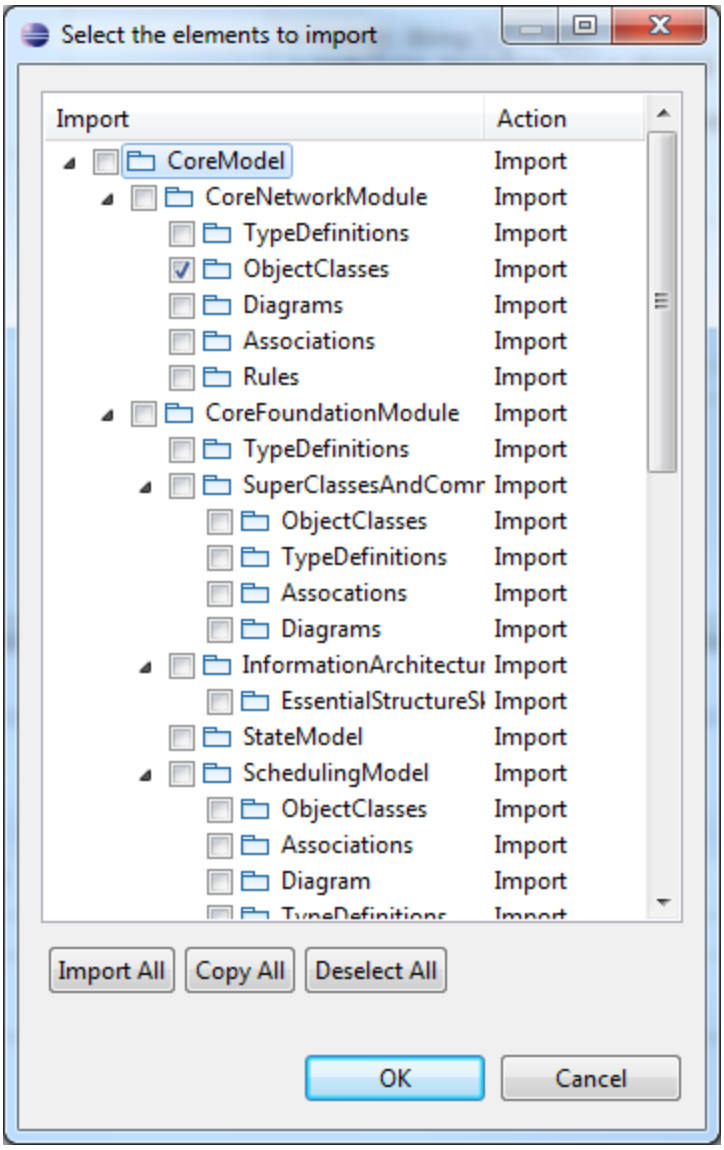


Figure 5-29: Selecting Core Model Artifacts

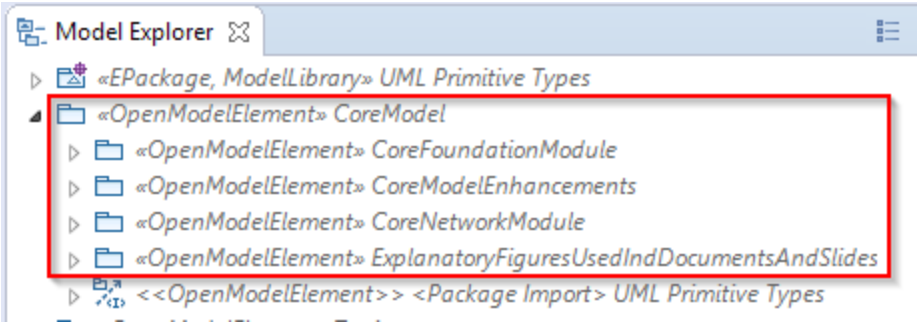


Figure 5-30: Imported Core Model

5.7 Deleting a Project

Projects can be deleted from the Project Explorer by a right click on the project (e.g., XxxModel) and selecting Delete.

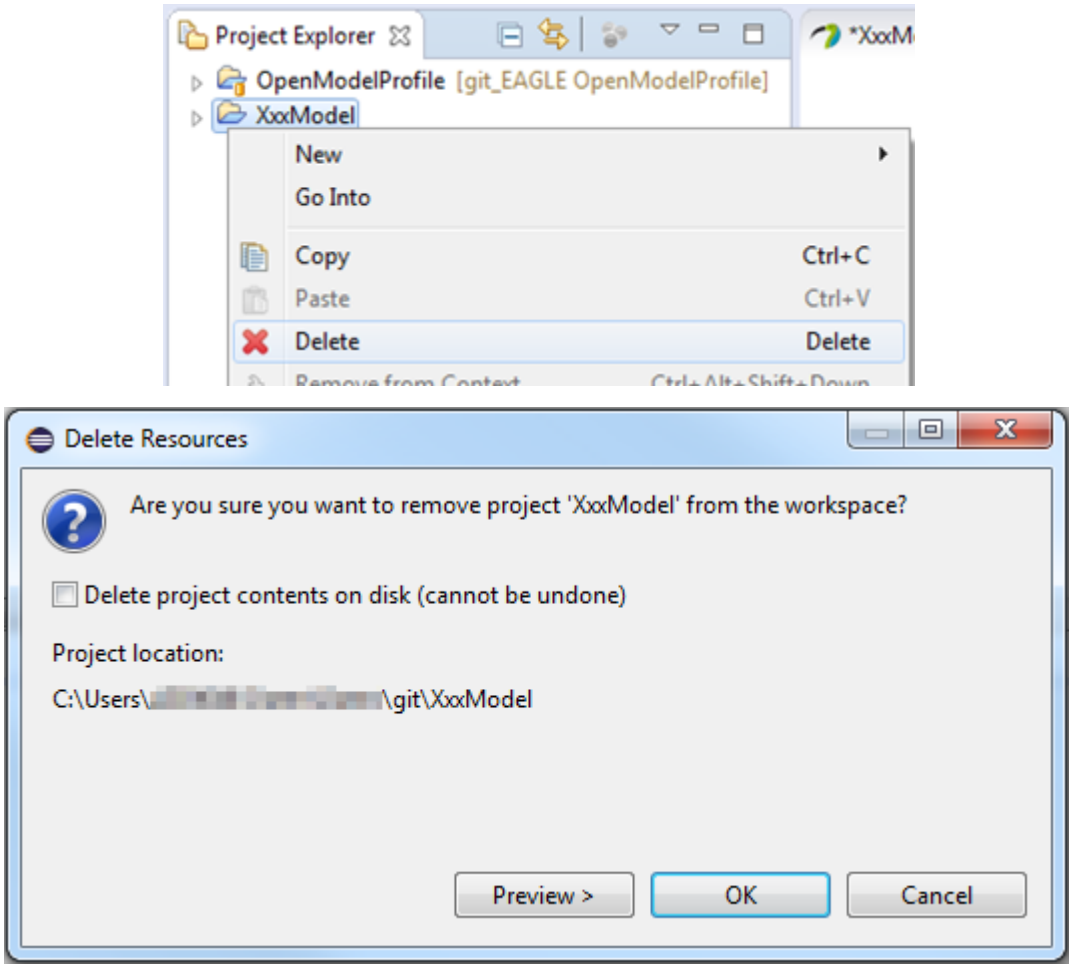


Figure 5-31: Delete a Project

Note: When Delete project contents on disk (cannot be undone) is not checked the model will only be removed from the Project Explorer.

6 GitHub Usage

6.1 Introduction to GitHub

Wikipedia: "*Git is a widely-used source code management system for software development. It is a distributed revision control system with an emphasis on speed, data integrity, and support for distributed, non-linear workflows.*"

Git is a protocol which allows managing data in repositories on hosts. Such hosts are provided by many companies and organizations.

GitHub

GitHub is a web-based git repository hosting service.

Note:
These Guidelines describe the usage of repositories stored in GitHub. The usage of other hosting services is similar.

The work flow recommended in these Guidelines is based on the usage of four repositories:

- a repository on GitHub containing the published files
- a copy of the published repository located in my (user/modeler) remote repository on GitHub
- a copy of my remote repository located on my local PC
- a copy of my local repository in my local working directory
  (all changes done by me will initially be stored in this working directory)

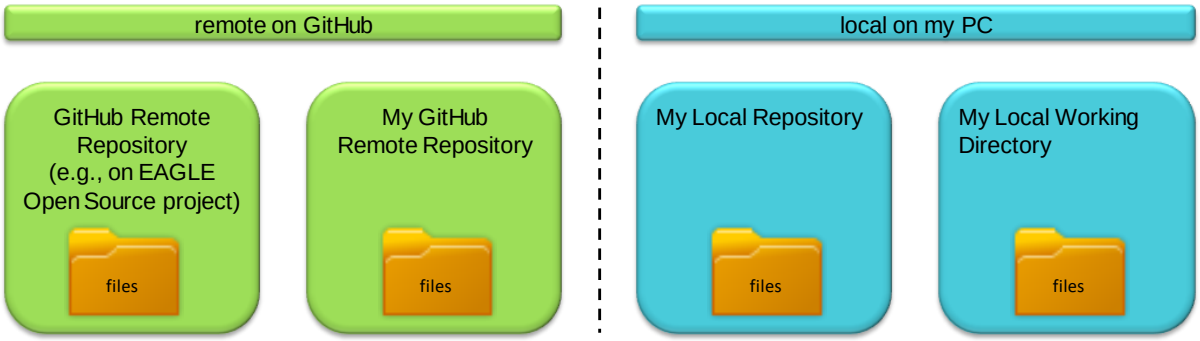


Figure 6-1: Recommended Repositories

6.2 Open Model Profiles and Tools on GitHub

IISOMI [5] is developing a set of SDO-neutral modeling infrastructure (common) artifacts such as:

- UML profiles
  Contain the definition of additional properties for UML artifacts. The additional properties are described in the UML Modeling Guidelines [4].
- UML class diagram style sheets
  Ensure a common look and feel of the UML class diagrams.
- common data types
- tools.

The following steps describe the recommended work flow for downloading these artifacts to the local PC for usage in Papyrus.

The following sub-sections correspond with the numbers ❶ in Figure 6-2.

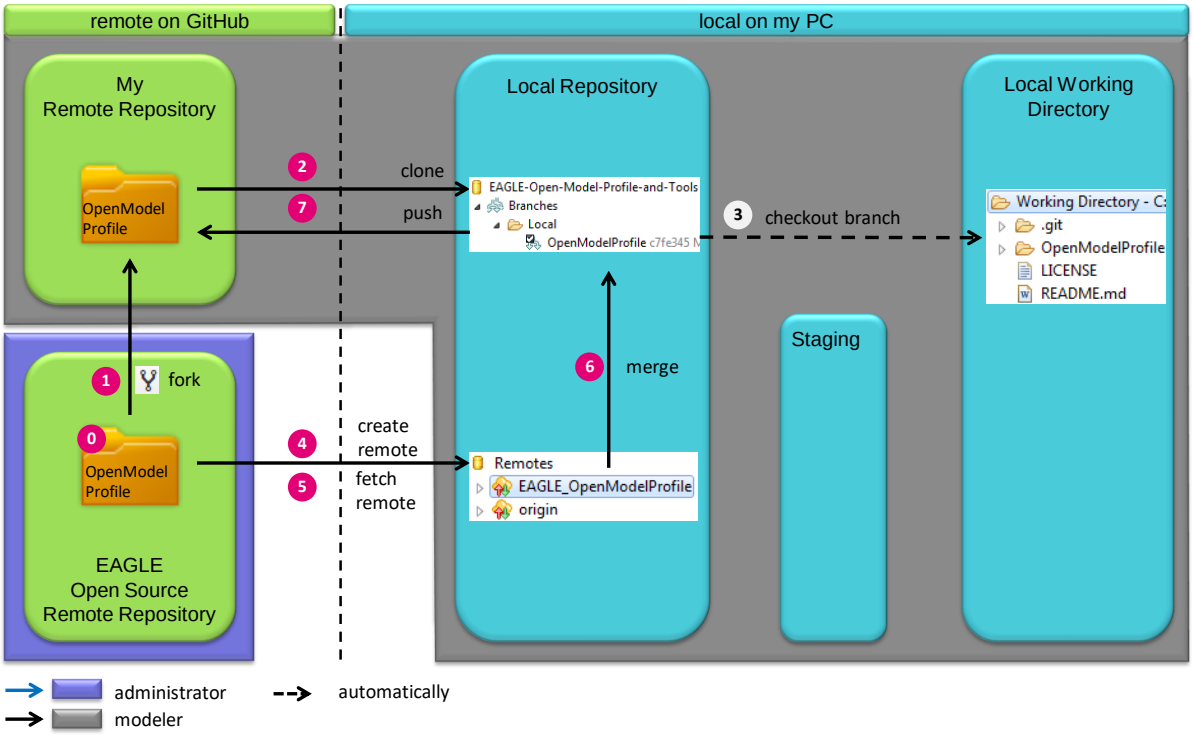


Figure 6-2: Recommended Work Flow for ToolChain Download

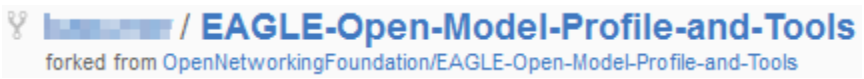
❶ Modeling Infrastructure Files on GitHub

The artifacts are published in the ToolChain branch of the ONF Open Source EAGLE-Open-Model-Profile-and-Tools repository (https://github.com/OpenNetworkingFoundation/EAGLE-Open-Model-Profile-and-Tools/tree/ToolChain).

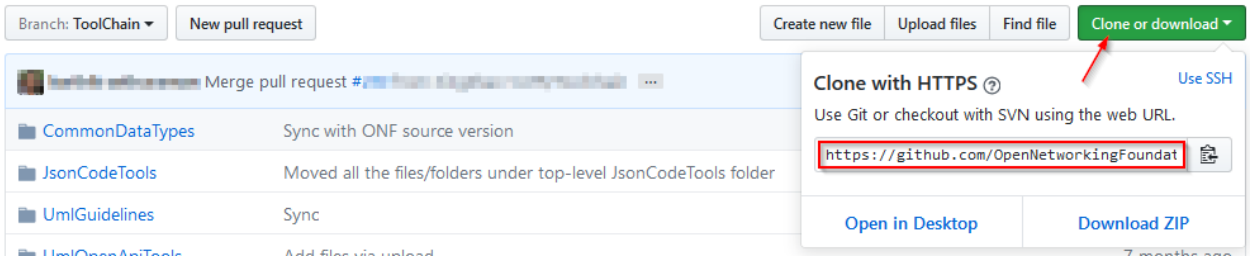
6.2.1 Forking

The modeler needs to copy the repository into its own GitHub space; by clicking Fork:

A copy of the complete repository is now contained in the modeler's GitHub space:



The URI pointing to this repository is provided here:



6.2.2 Cloning

The ToolChain branch needs to be cloned to the modeler's local repository. This is done using the git client that is contained in the Eclipse tool.

After Eclipse has been launched on the local PC, the git client can be started by clicking the *Open Perspective* button: and then choosing the Git perspective.

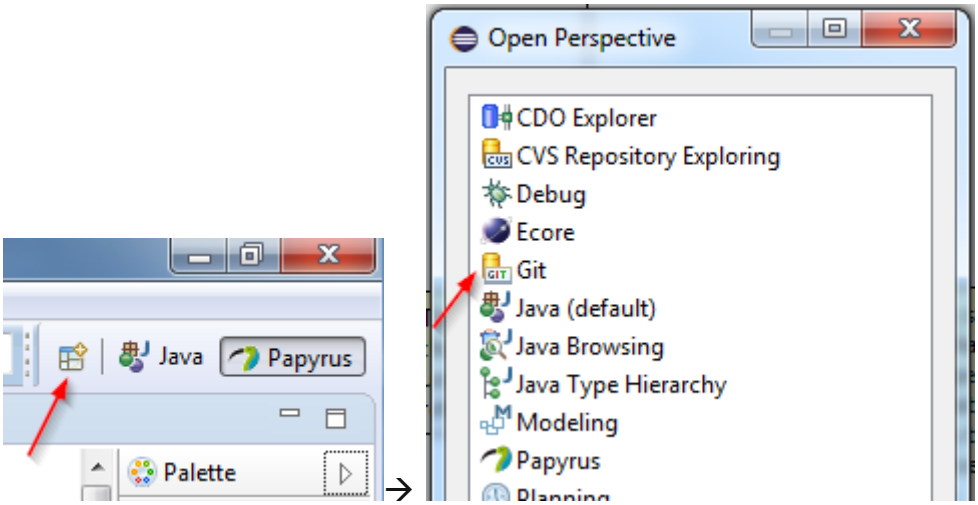


Figure 6-3: Open Git Perspective

In the Git Repositories window click on or Clone a Git repository :

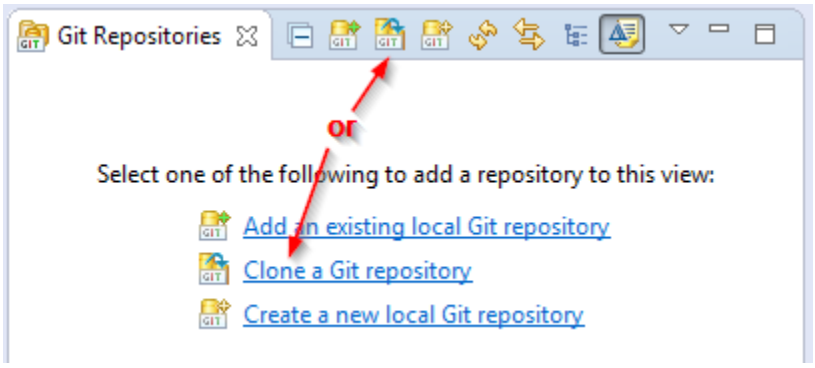


Figure 6-4: Add Repository Choices

The URI of the repository is provided on the modeler's GitHub space; see step 1.
Copy this address (https://github.com/<modeler's git user name>/EAGLE-Open-Model-Profile-and-Tools.git) into the URI field (the Host and Repository path fields are then automatically populated) and enter the modeler's GitHub username and password.

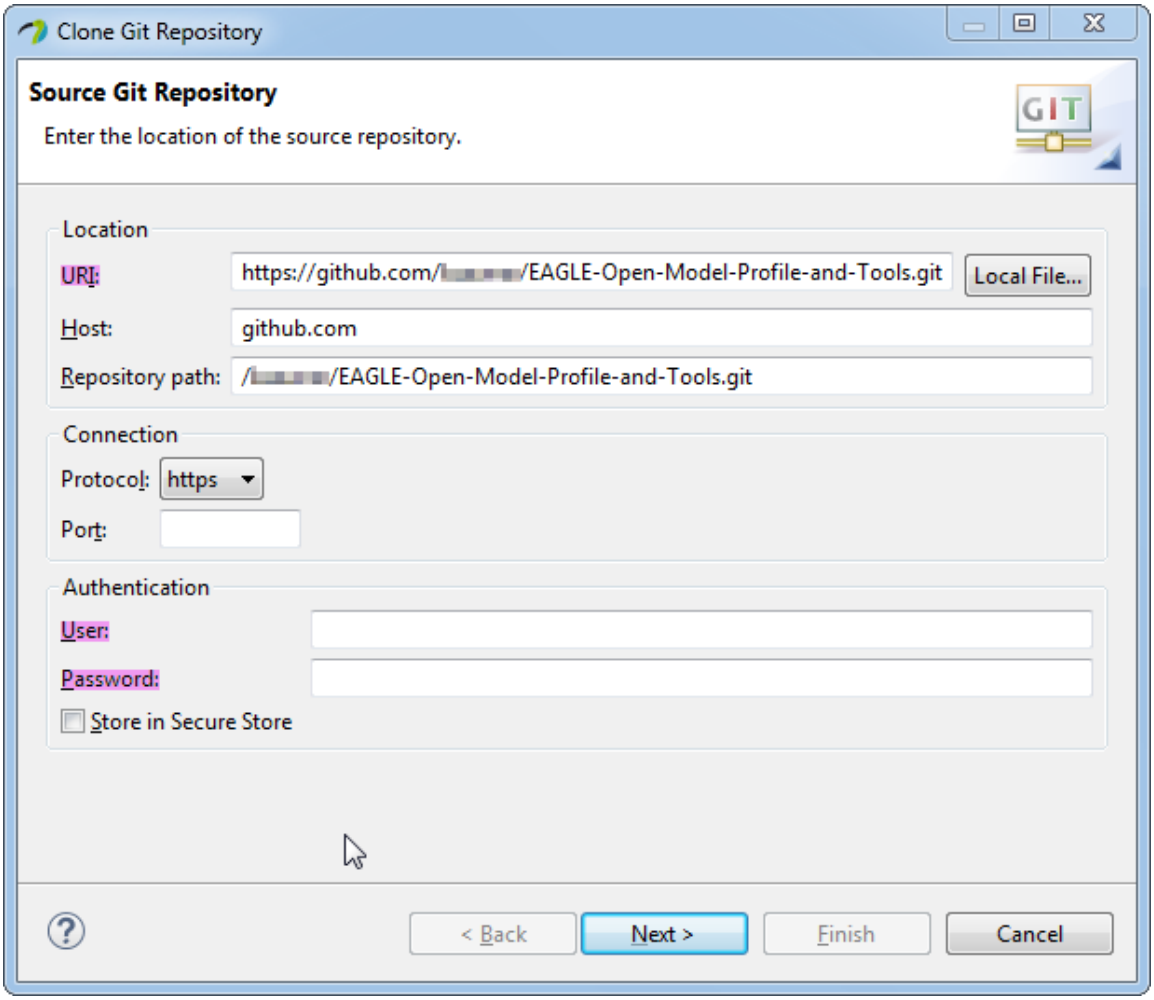
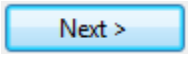
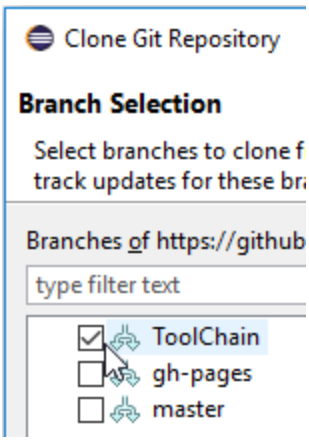


Figure 6-5: Source Git Repository Window

Click  and select **only** the ToolChain branch:



Click Next > and insert the destination directory on the local disk where the profile shall be stored.

Clone submodules and Import all existing Eclipse projects after clone finishes should be checked.

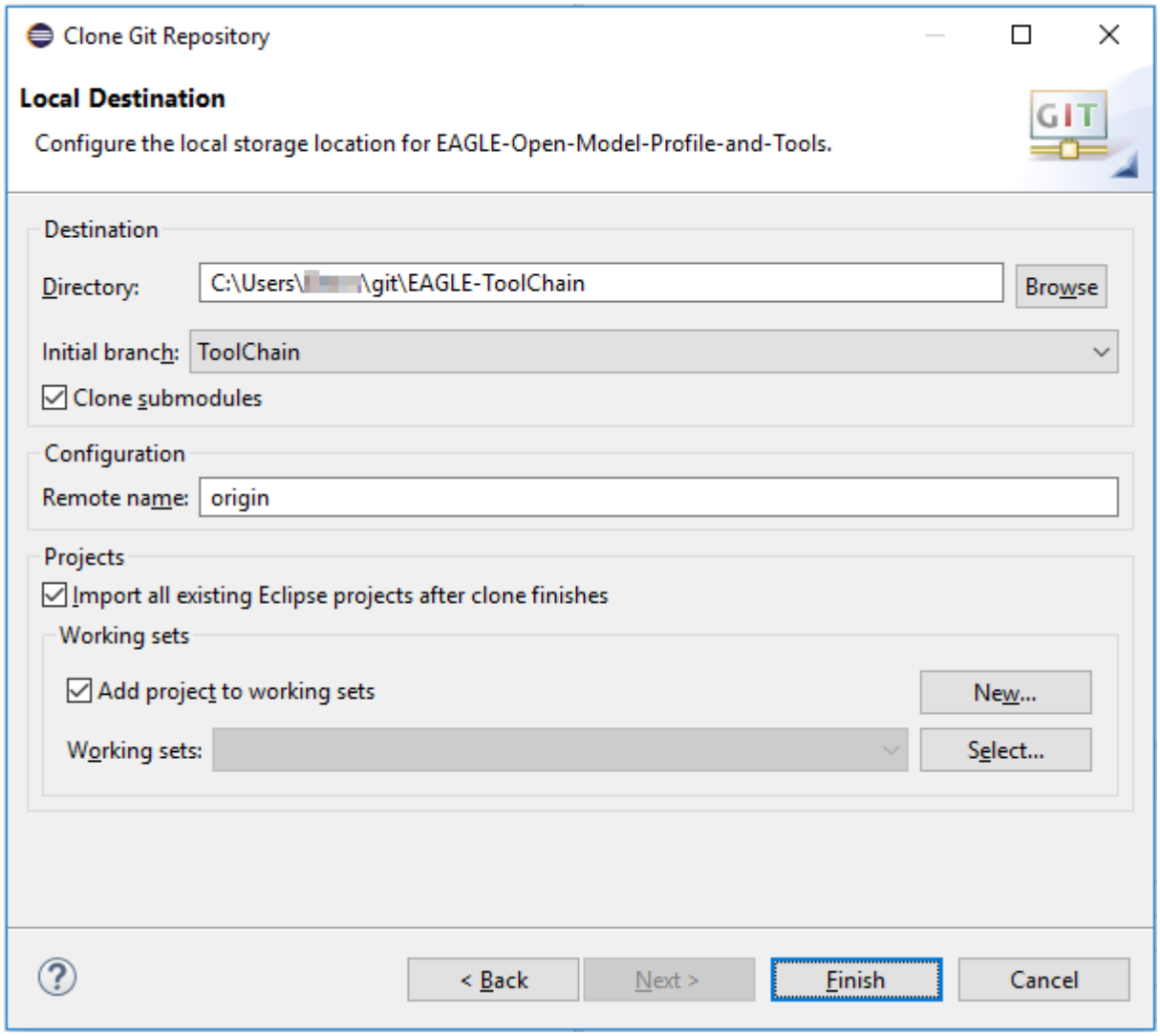


Figure 6-6: Local Destination Window

Click Finish. The ToolChain branch is now downloaded to the local PC.

6.2.3 Model in Papyrus

The Git Repositories window should then contain the following files:

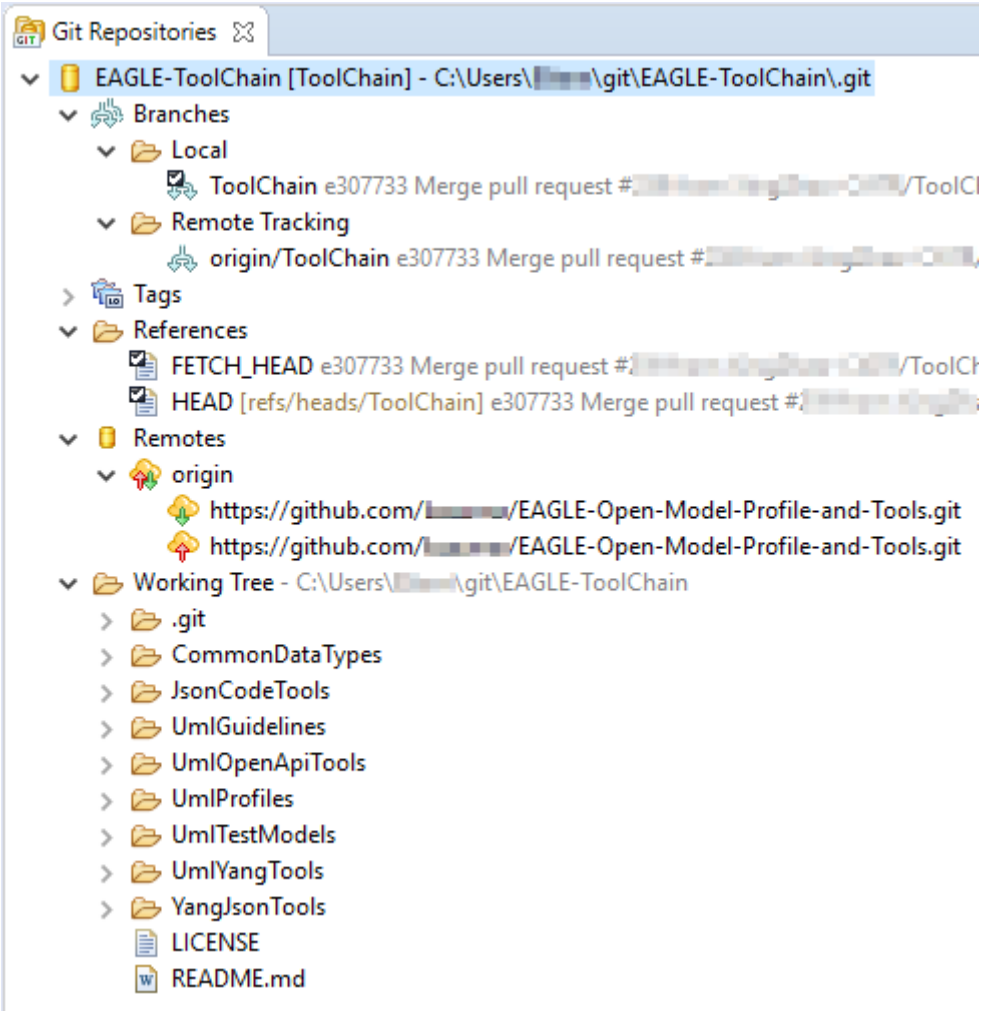
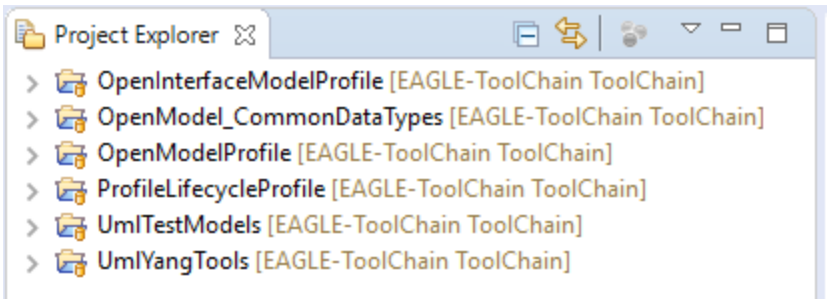


Figure 6-7: ToolChain Branch Cloned to Local PC

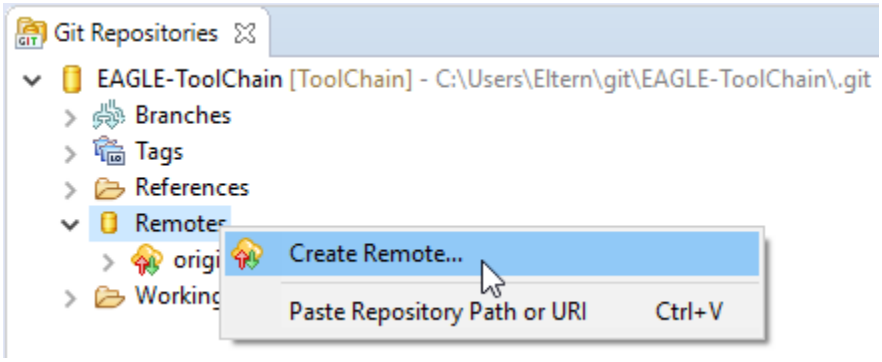
Since the Import all existing Eclipse projects after clone finishes checkbox was selected, all Papyrus projects contained in the ToolChain branch automatically appear in the Project Explorer window in the Papyrus perspective. This can be rechecked in the perspective:



6.2.4 Additional Remote Repository

The following steps describe how to update the local ToolChain files.
The latest version of the ToolChain files is stored in the EAGLE-Open-Model-Profile-and-Tools remote repository on GitHub. This repository needs to be added to the list of Remotes.

This can be configured in the Git Repositories window of the Git perspective. Right click on Remotes and select *Create Remote ...*:



Enter a name for the remote branch and select Configure fetch:

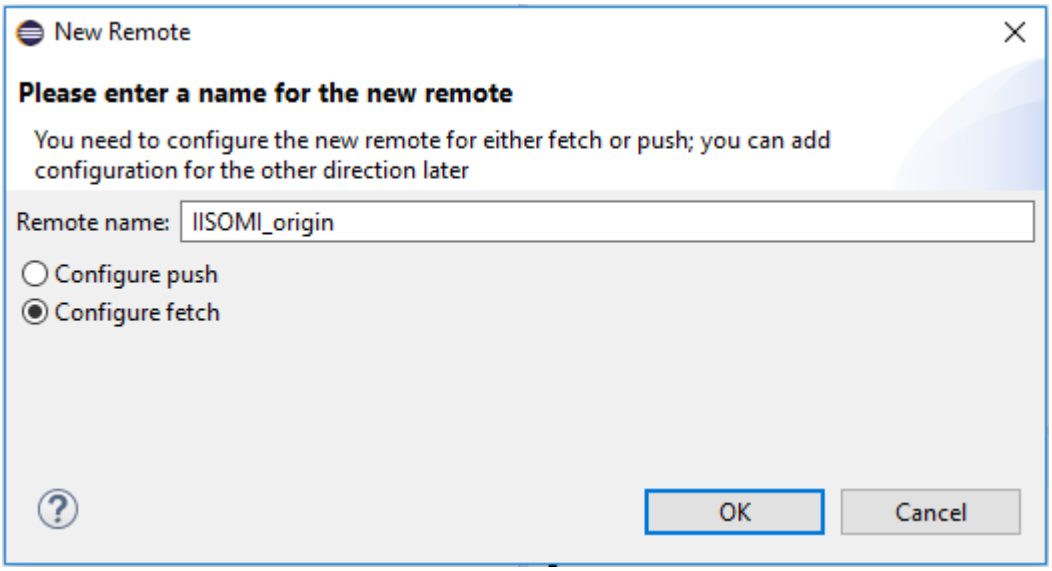
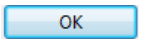


Figure 6-8: Create Additional Remote (1)

Click 

.

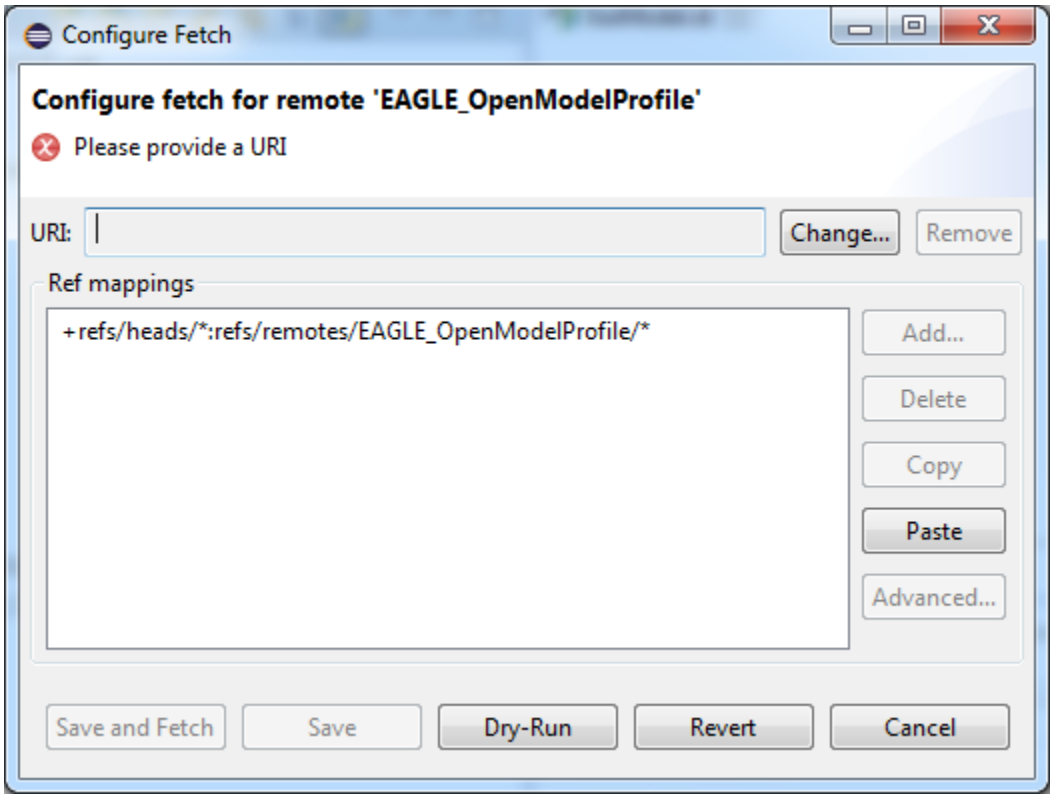
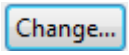


Figure 6-9: Create Additional Remote (2)

Click 

to enter the URI of the EAGLE repository (https://github.com/OpenNetworkingFoundation/EAGLE-Open-Model-Profile-and-Tools.git):

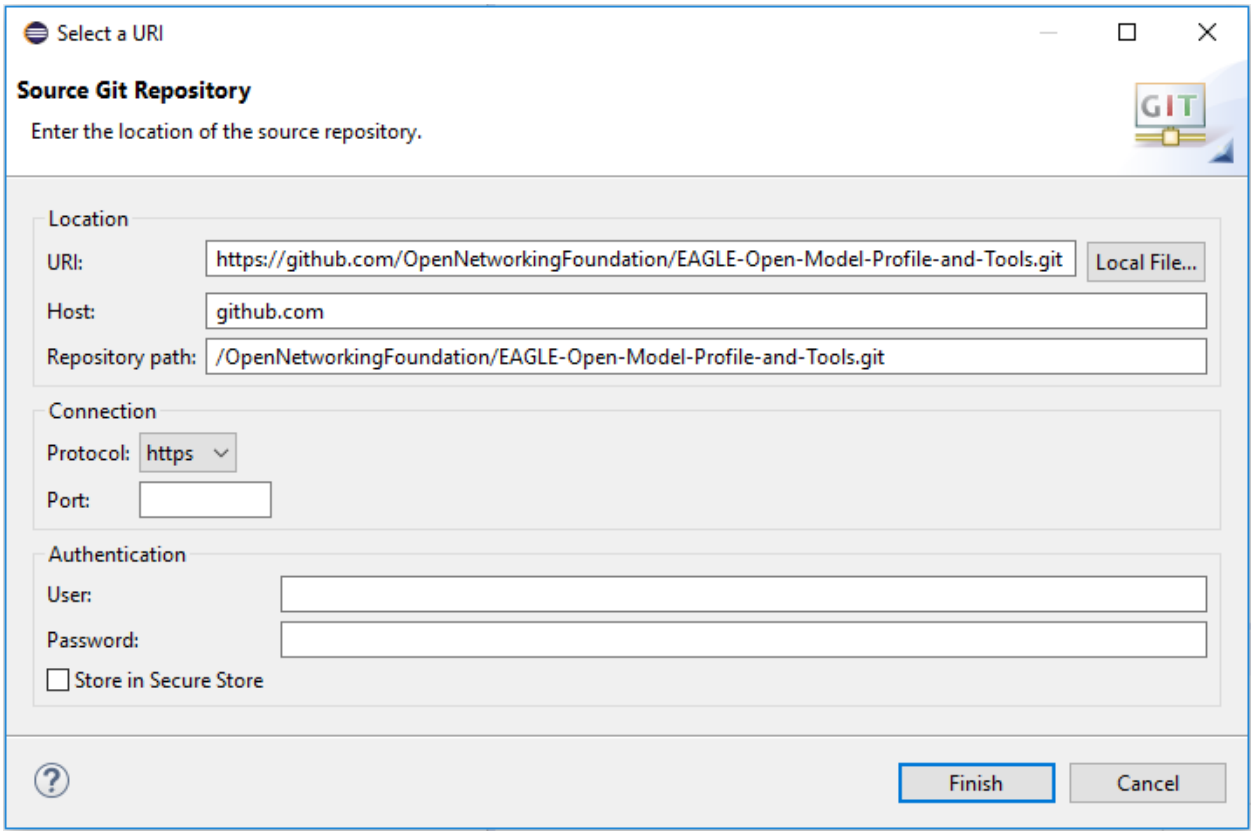
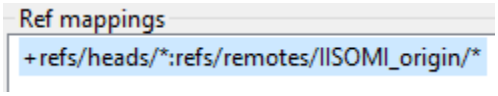


Figure 6-10: Create Additional Remote (3)

Enter user name and password and click Finish.
Delete the existing item in the *Configure Fetch* window:



Click then Add... in the *Configure Fetch* window. **Enter a space** in the *Source* field to get the list of the branches and select ToolChain [branch]:

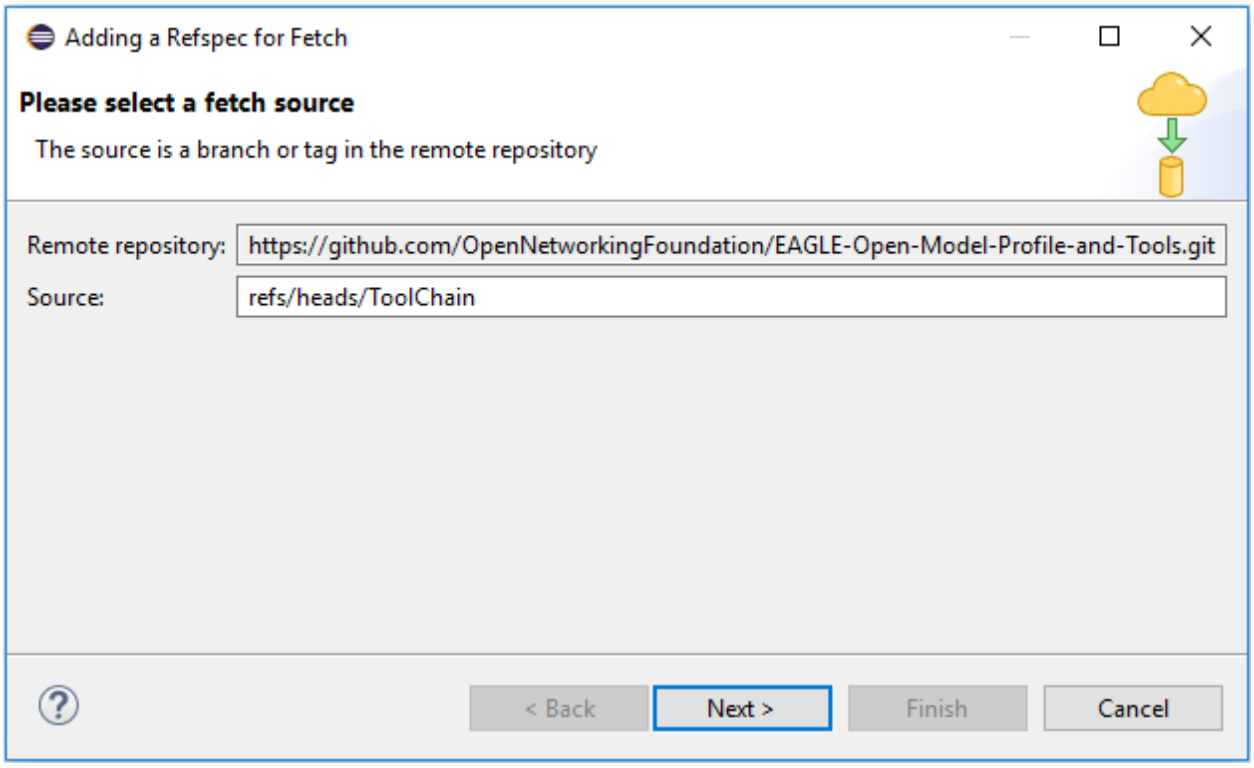


Figure 6-11: Create Additional Remote (4)

Click Next > and then Finish

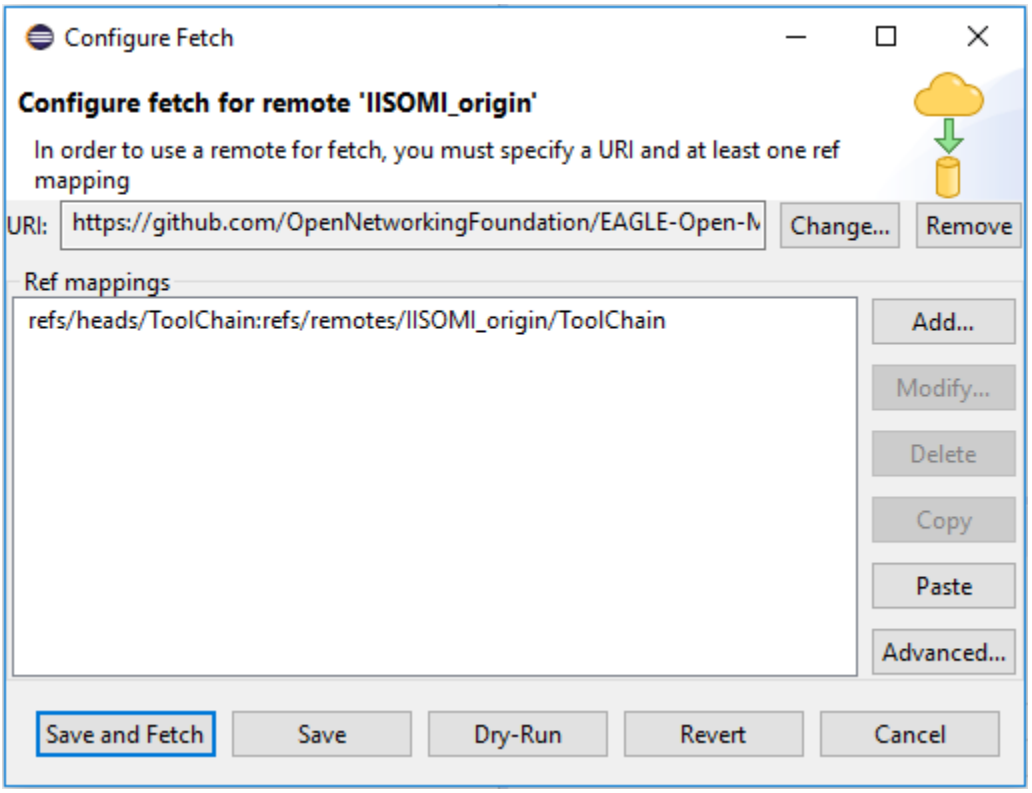
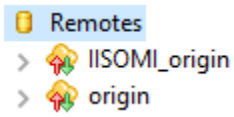


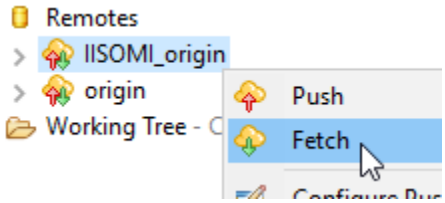
Figure 6-12: Create Additional Remote (5)

Click Save and Fetch.
The additional branch is added to the *Remotes* list:



6.2.5 Fetching Remote Repository

To align the ToolChain version in the local repository with an updated version on GitHub go to the Git Repositories window of the Git perspective. Right click on IISOMI_origin in the Remotes list and select Fetch:



After the request, a *Fetch Results* window lists

- either the commits that have been done on the version in the EAGLE-Open-Model-Profile-and-Tools repository since the local repository has been updated
- or everything is already up to date.

6.2.6 Updating Local Repository

Then the local branch, which has been checked out, needs to be merged with the new version that was fetched in Step 5.
Right click on the local OpenModelProfile branch and then select Merge... :

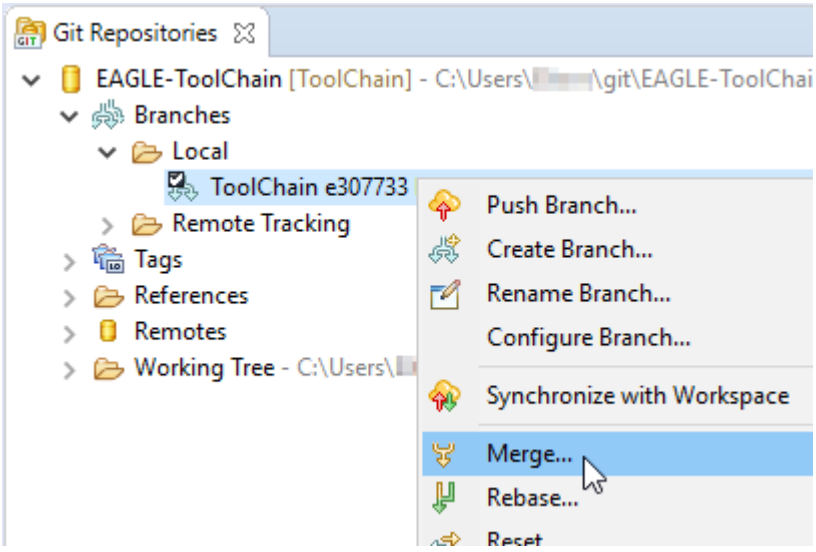


Figure 6-13: Merge Request (1)

Select the branch that needs to be merged into the ToolChain branch:

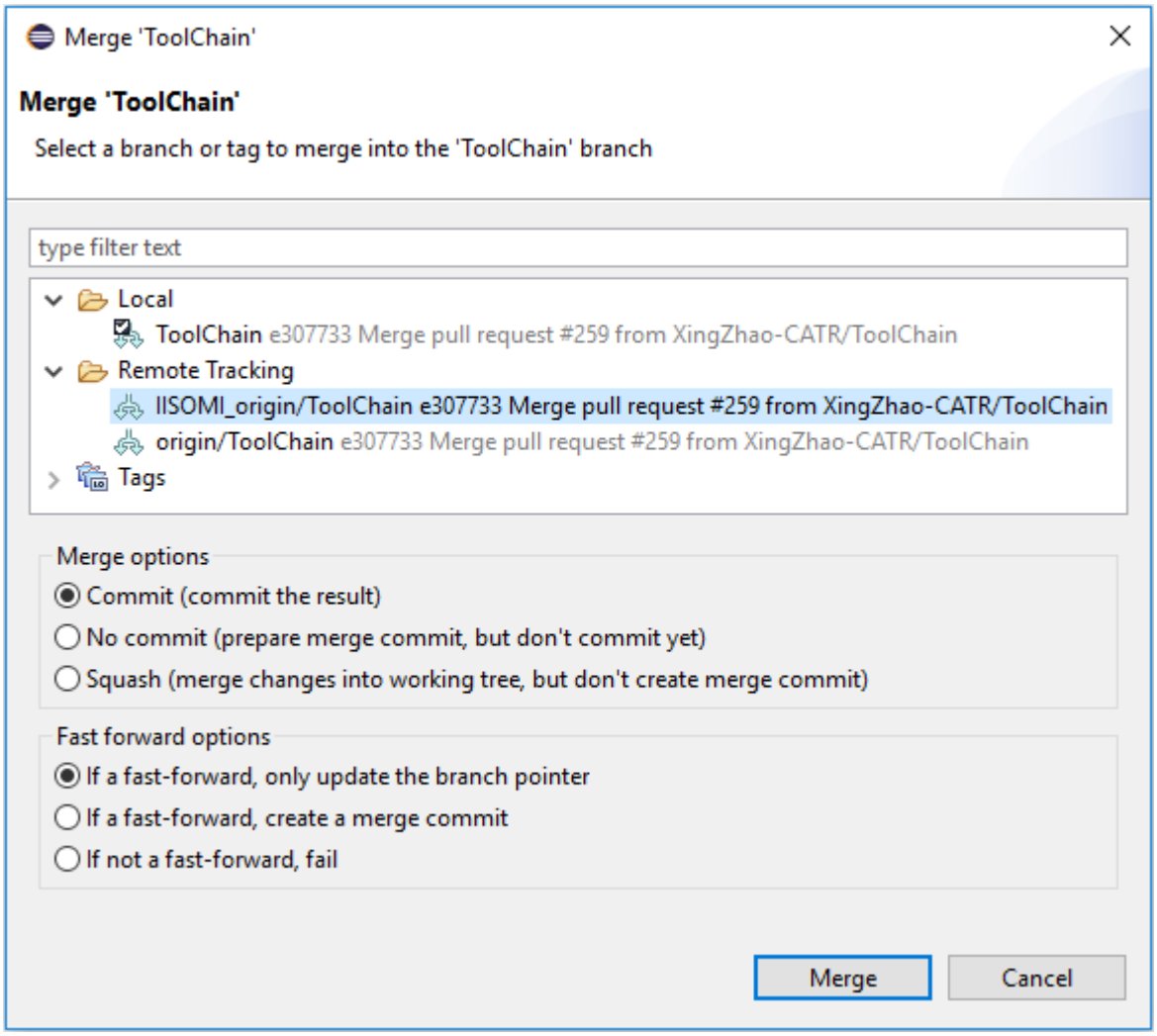
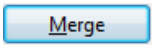


Figure 6-14: Merge Request (2)

Then click  and acknowledge the *Merge Results* window:

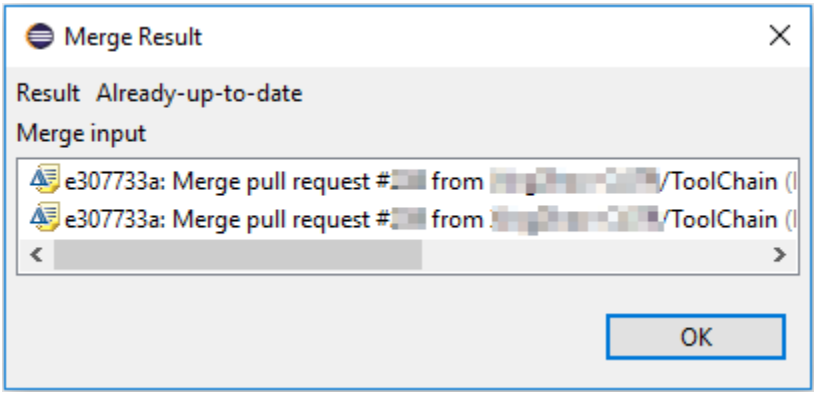
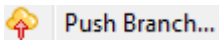


Figure 6-15: Merge Result

6.2.7 Updating Own Remote Repository

To update also the own remote repository right click on the local ToolChain branch and then select 

:

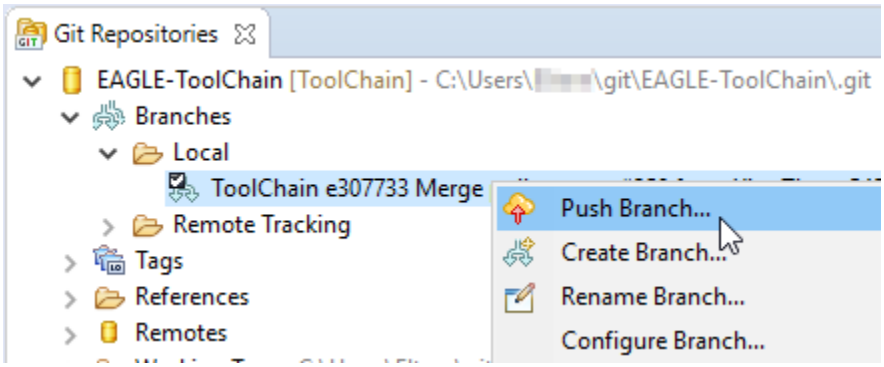


Figure 6-16: Push Request (1)

In the *Push Branch* window make sure that the modeler's remote Github repository is selected as the destination:

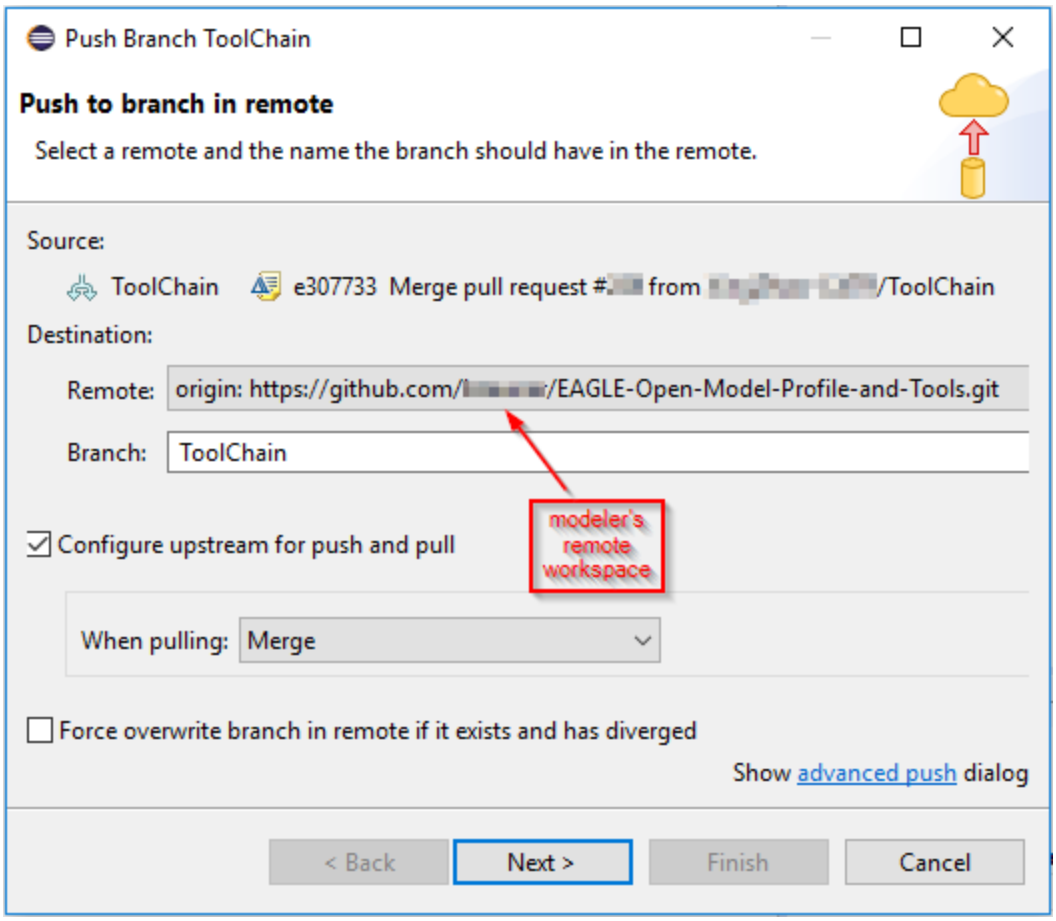


Figure 6-17: Push Request (2)

Click Next >.

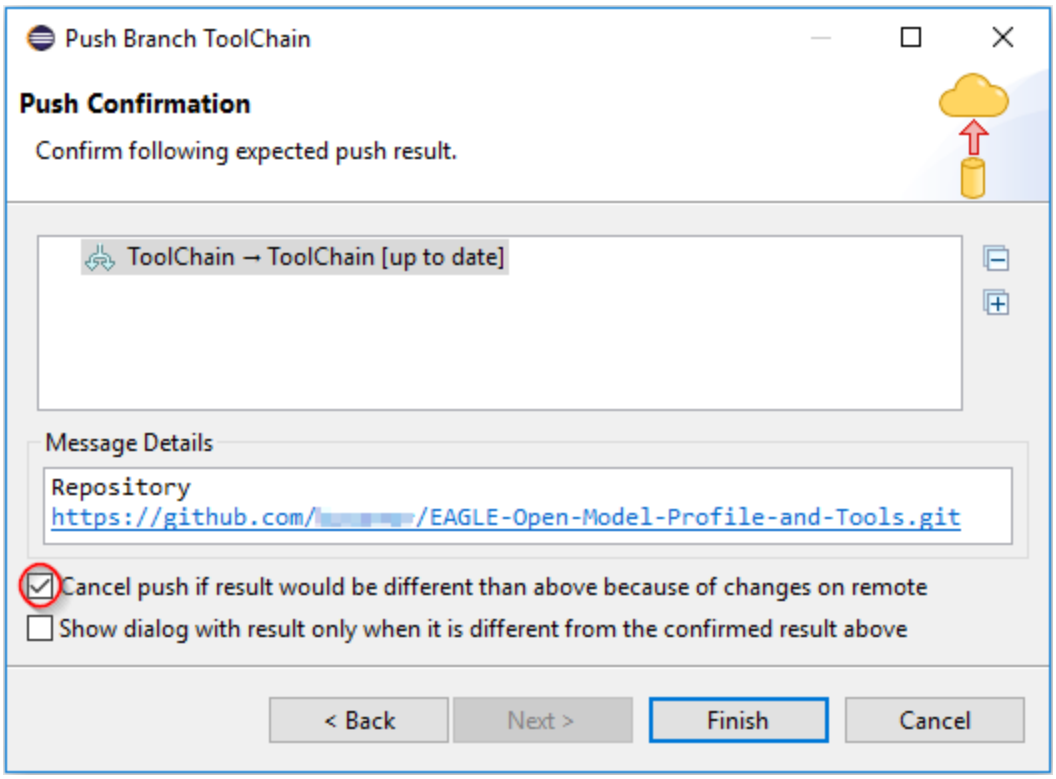
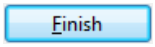


Figure 6-18: Push Request (3)

Click 

6.3 Information Model on GitHub

6.3.1 Placeholder XxxModel

Note:
The following GitHub usage description is using XxxModel as a placeholder for any Paprus UML model.

The XxxModel is stored in a repository on GitHub. The link to the repository is {XxxModel}.

The model development architecture identifies two groups of people that are working with the model: (a) Modelers who do the actual writing of the model pieces, and (b) Administrators who establish the working environment and control the “master copy” of the XxxModel.

6.3.2 XxxModel Structure on GitHub

The XxxModel is contained in a “master branch” on GitHub. A copy of this master branch is provided in the “develop branch”.

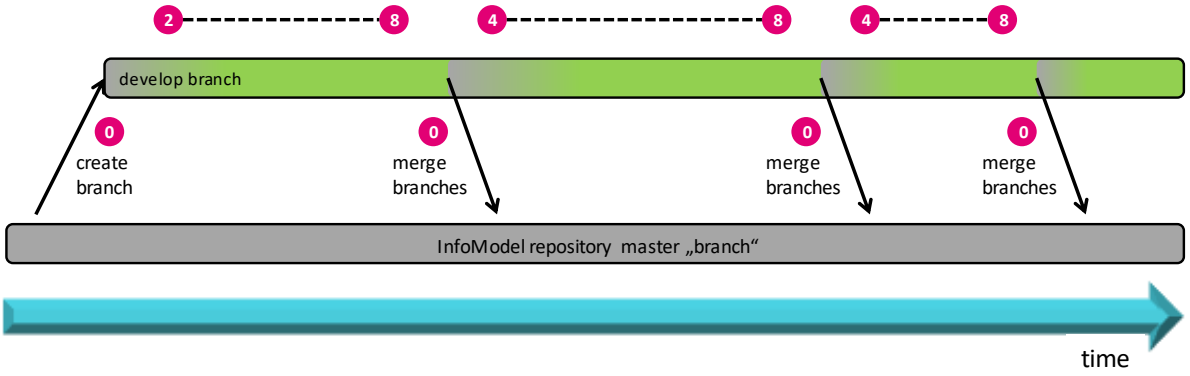


Figure 6-19: Initial Model Structure on GitHub

The modeling teams are developing their part of the model in their develop branch. All updates of the individual develop branches will be merged back to the master branch from time to time[1].

6.4 GitHub Work Flow

The following steps describe the work flow that a modeler has to follow when developing the XxxModel.

Clause 6.5 describes a more easy way of getting the XxxModel to the local PC. This way is restricted to “read only viewers” of the model since it does not allow to commit changes back to GitHub.

The sub-sections correspond with the numbers 1 in Figure 6-20.

[1] Driven by overall delivery schedule and coordinated by the administrators.

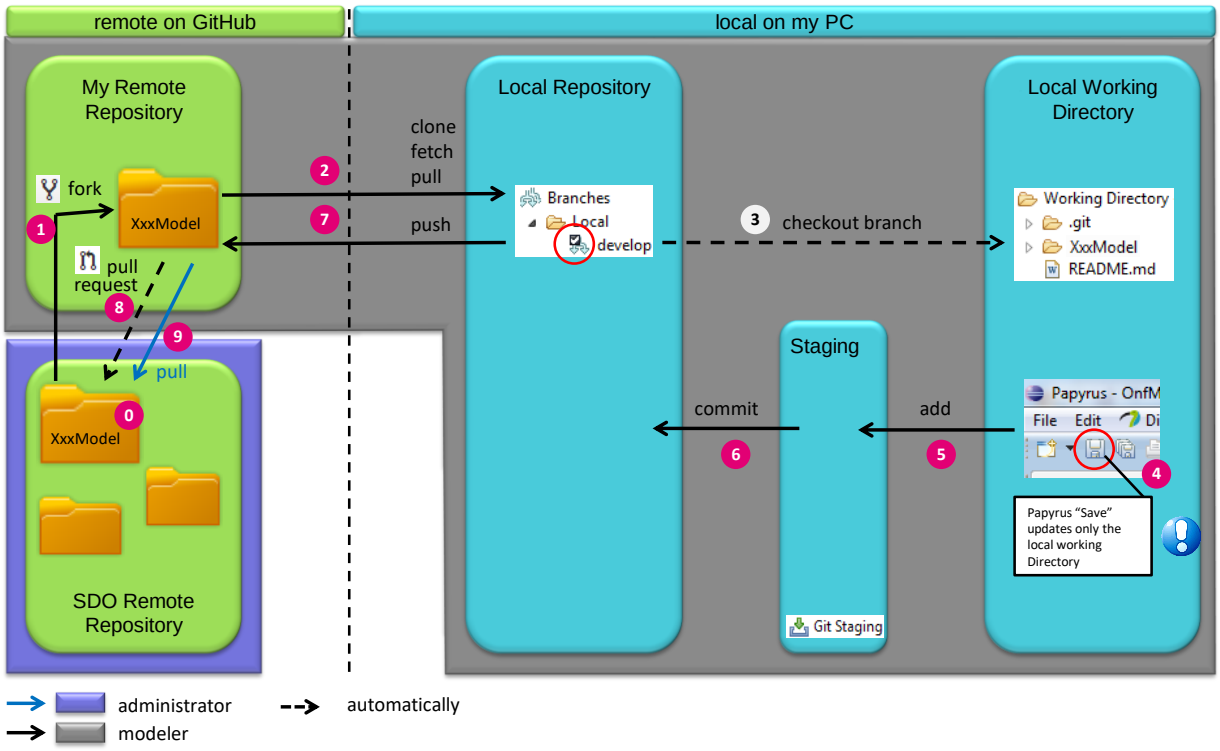
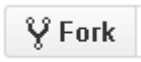


Figure 6-20: GitHub Work Flow

⓿ Model Structure

The administrator has established the XxxModel repository (containing the master and develop branches) in the SDO git space under the following URL: {XxxModel}.

6.4.1 Forking

The modeler needs to copy the repository from the SDO git space into its own git space; by clicking on 

:

6.4.2 Cloning

An individual modeler works only in the develop branch. This specific branch needs to be copied to the modeler's local PC into a local repository. This is done using the git client that is contained in the Eclipse tool on the local PC.

Make sure that the Eclipse Workspace that is selected during launch of Eclipse does not already contain the XxxModel project.

Note:
More than one version of the model can only be maintained on the local PC if they are in different Eclipse Workspaces.

After the Eclipse has been launched in the local PC, the git client can be started by clicking the "Open Perspective" button: and then choosing the Git perspective.

Note: Depending on the Eclipse version, the git client may have another name (including the term "Git")2.

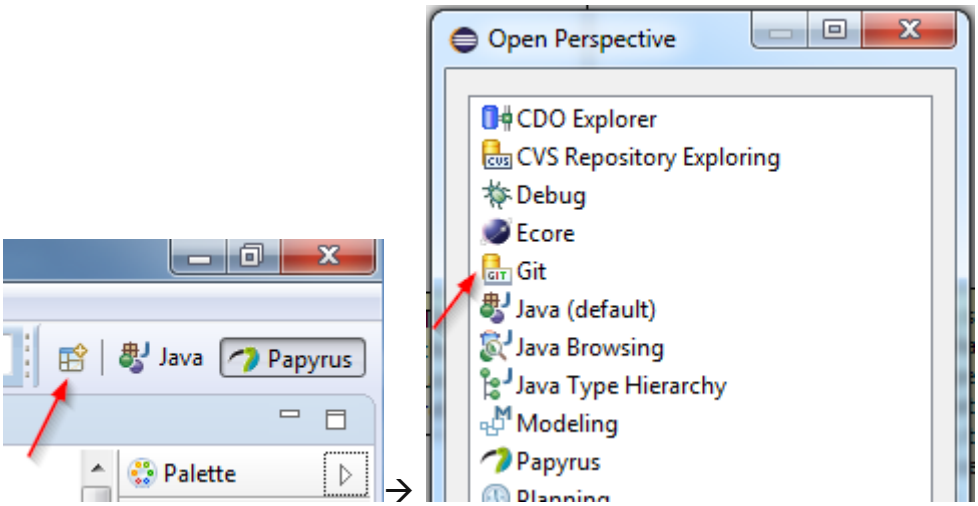


Figure 6-21: Open Git Perspective

In the Git Repositories window click on or Clone a Git repository :

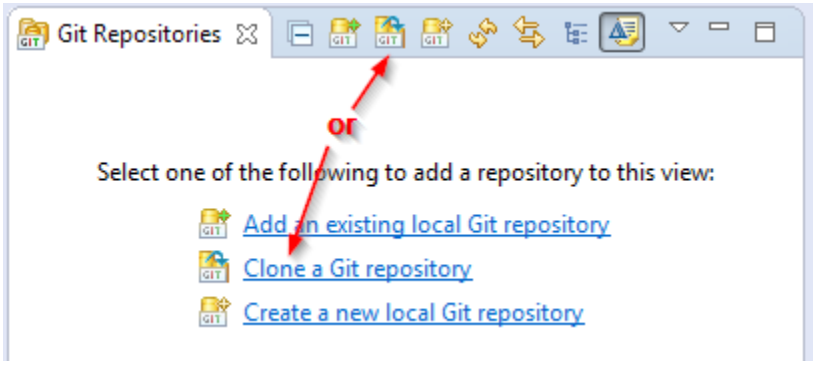


Figure 6-22: Add Repository Choices

Copy and paste the address that is provided on the web page of the XxxModel in the modeler's git space:

[2] It is vital that all modelers use the same version of Papyrus to prevent compatibility issues.

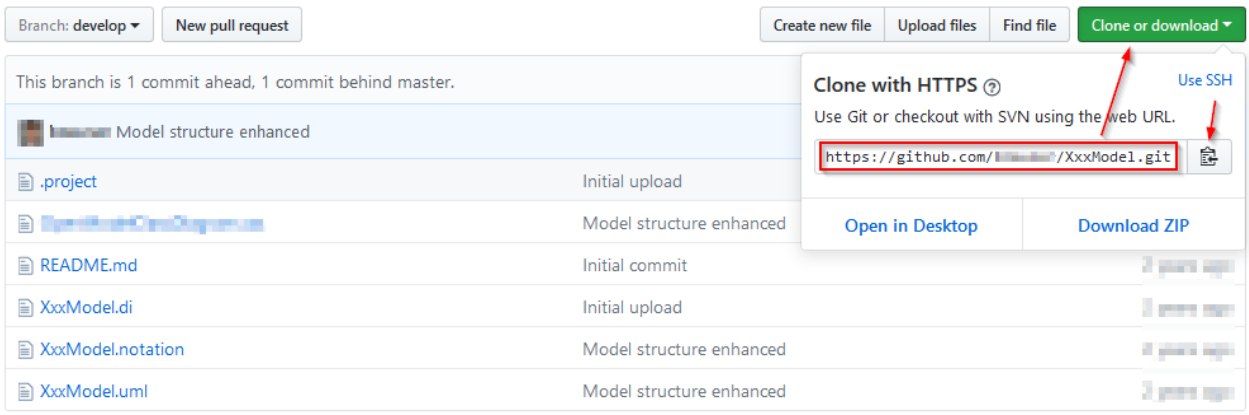


Figure 6-23: Location of the Repository Address

Copy this address (https://github.com/<modeler's git user name>/XxxModel.git) into the URI field (the Host and Repository path fields are then automatically populated) and enter the modeler's GitHub username and password.

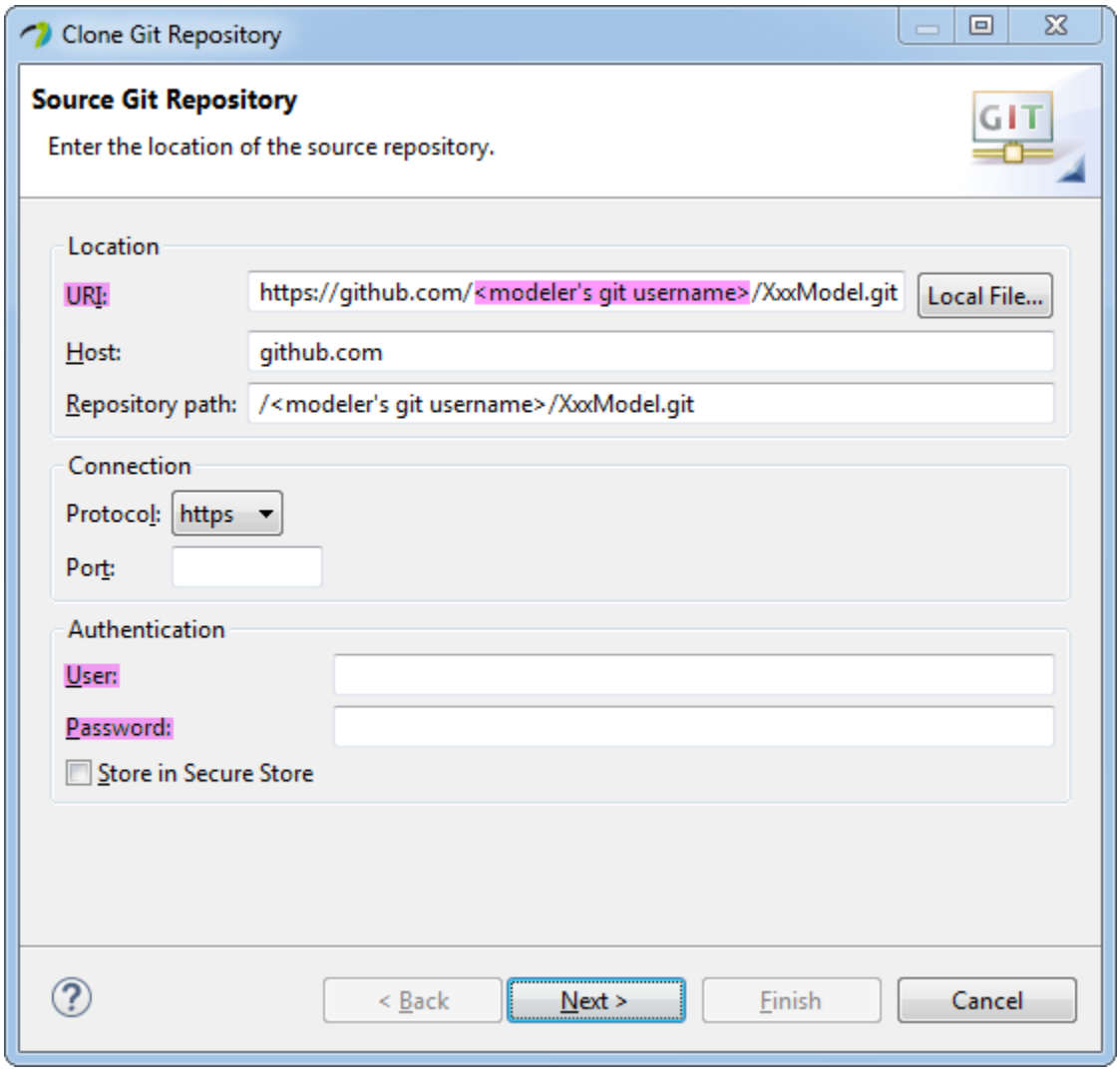


Figure 6-24: Source Git Repository Window

Click Next > and select **only** the develop branch.

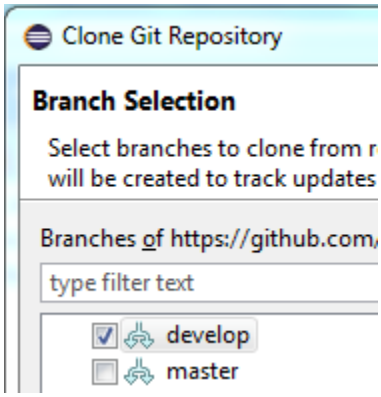


Figure 6-25: Branch Selection Window

Click Next > and insert the destination directory on the local disk where the model shall be stored.

☑ Clone submodules and ☑ Import all existing Eclipse projects after clone finishes should be checked.

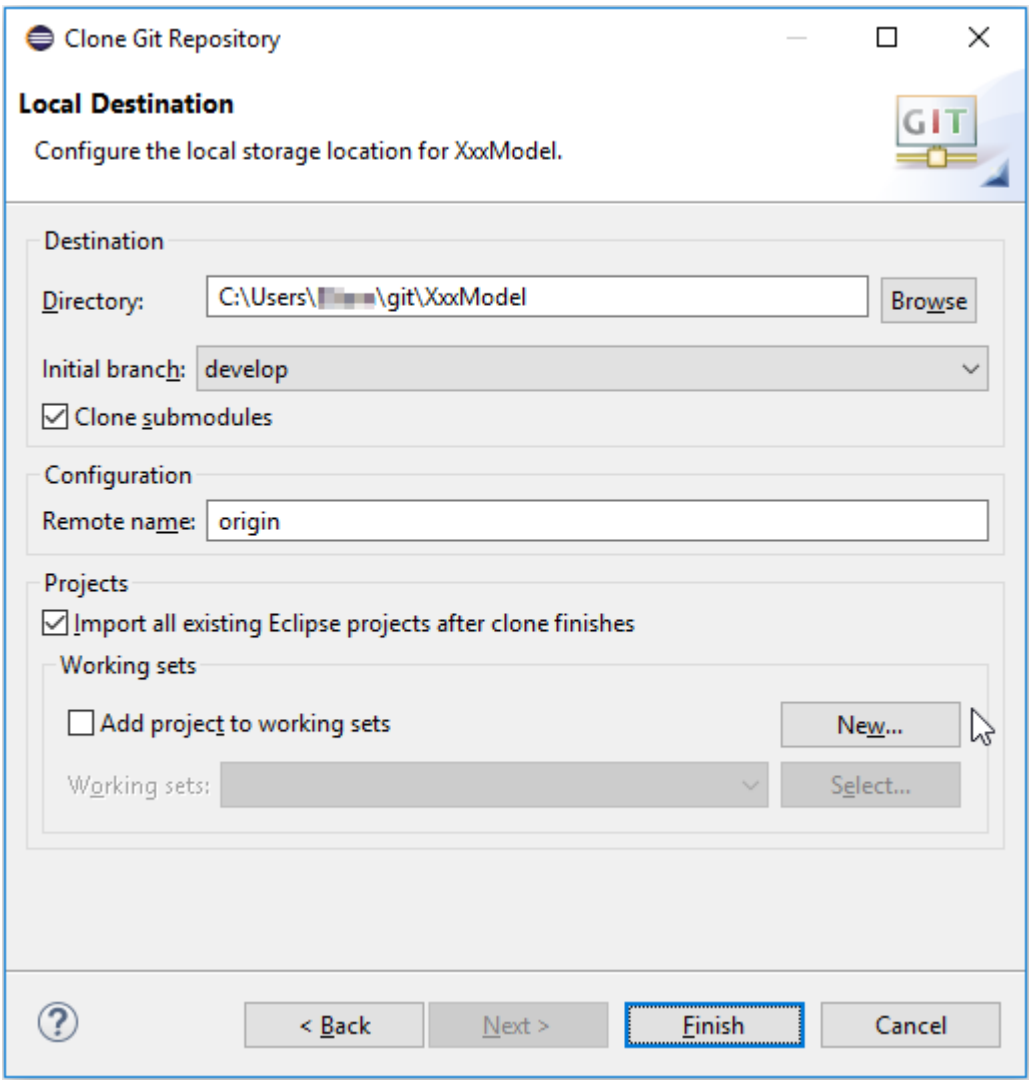
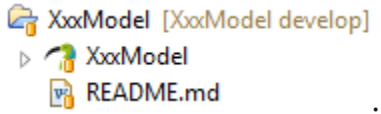


Figure 6-26: Local Destination Window

Click Finish. The selected branch of the XxxModel is now downloaded to the local PC.

6.4.3 Model in Papyrus

Since the Import all existing Eclipse projects after clone finishes checkbox was selected, the XxxModel project should automatically appear in the Project Explorer window in the Papyrus perspective. This can be rechecked in the perspective:



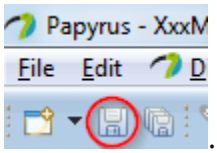
6.4.4 Developing the Model

A double-click on XxxModel starts the modeling in Papyrus and opens the XxxModel in the Model Explorer window.

Figure 6-27: XxxModel Shown in Papyrus Model Explorer

The designer of the XxxModel can now develop the model.

Note: Papyrus “Save” updates only the local working Directory



6.4.5 Identify Changes to be upload to Repository

After saving the changes in Papyrus the updated files are shown in Git Staging in the Git perspective.

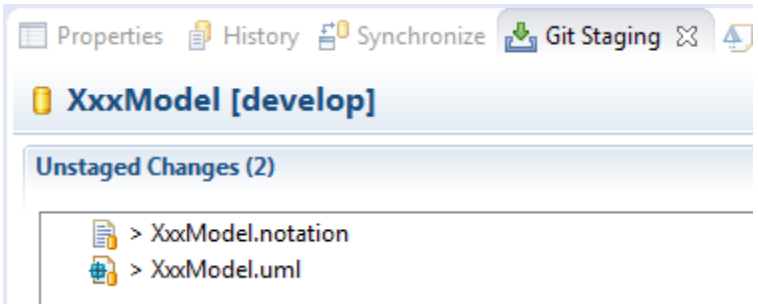


Figure 6-28: Unstaged Changes in Git Staging

Pick all XxxModel files, right-click and select Add to Index:

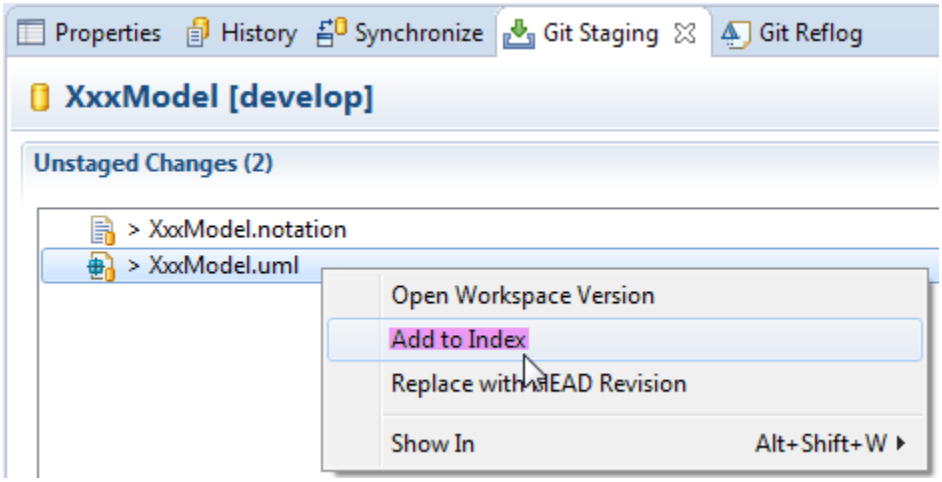


Figure 6-29: Add Files to Git Stage

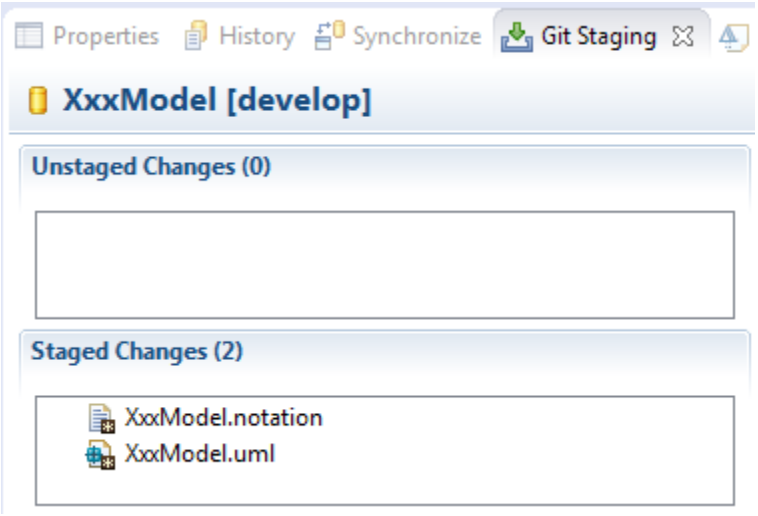


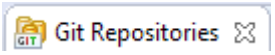
Figure 6-30: Staged Changes in Git Staging

6.4.6 Updating Local Repository

To commit the staged changes to the modeler's local repository, provide a commit message describing the changes that were done and then click on 

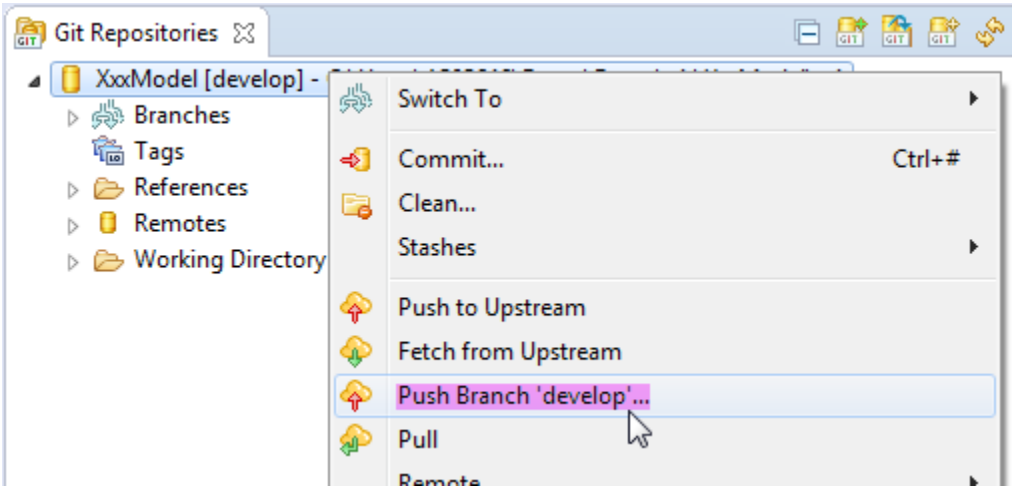
.

6.4.7 Updating Own Remote Repository

Steps 4 – 6 are all dealing only with changes on the local PC of the modeler. To save the changes to the modeler's remote repository, right-click on the repository in the 

tab and select 

.



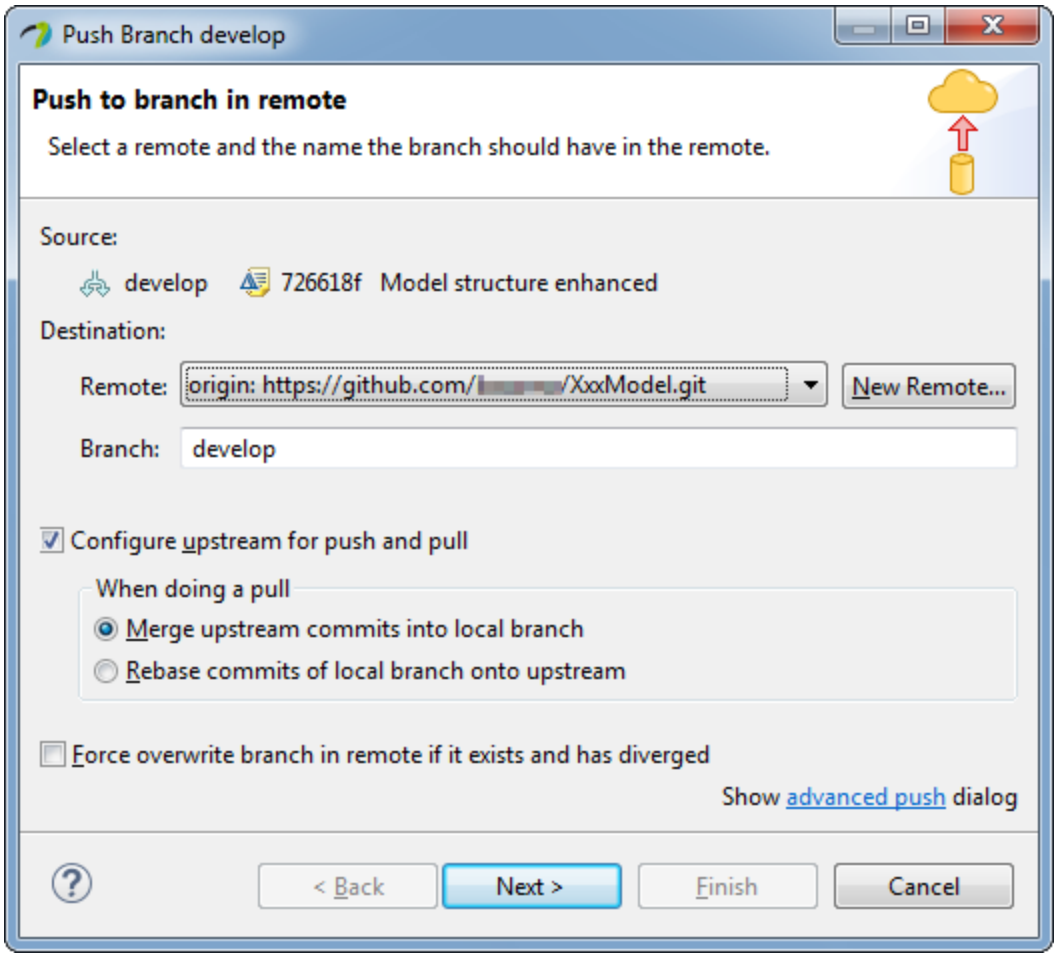


Figure 6-31: Push Updated Branch to Remote Repository

Make sure Configure upstream for push and pull and Merge upstream commits into local branch are checked.

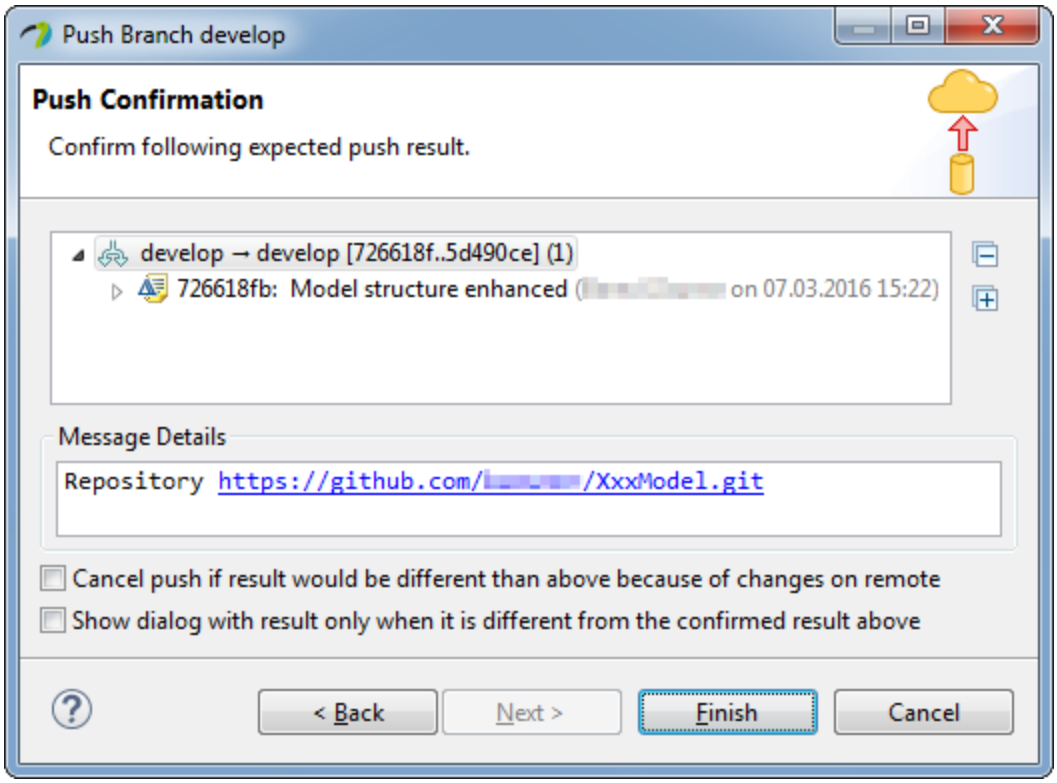
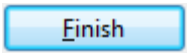


Figure 6-32: Push Confirmation Window

Finally click 

.

6.4.8 Updating Remote Repository

Steps 3 – 7 can be done as often as necessary.
Once all updates of the sub-model for the next release of the XxxModel are finished, the modeler needs to notify the administrators that a stable version is ready. This is done by a pull request from the modeler's remote repository using 

or 
.

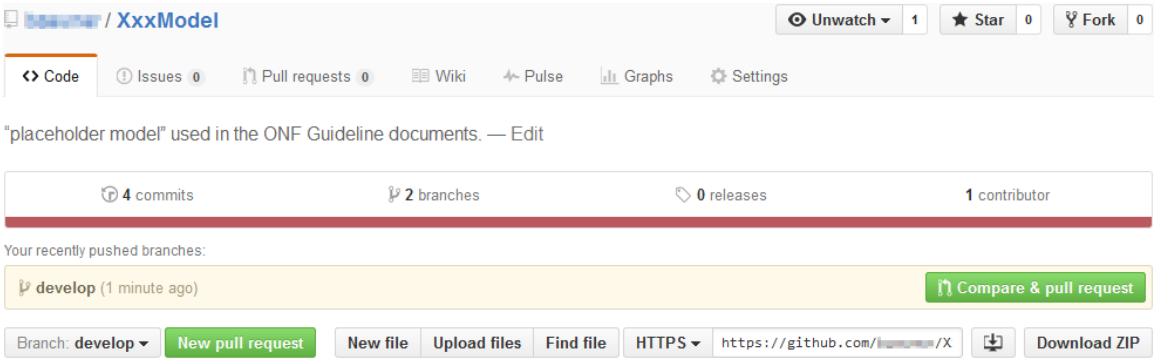


Figure 6-33: Compare in Modeler's Remote Repository

After click on 

or 
git provides a detailed comparison of all changes done in all files.

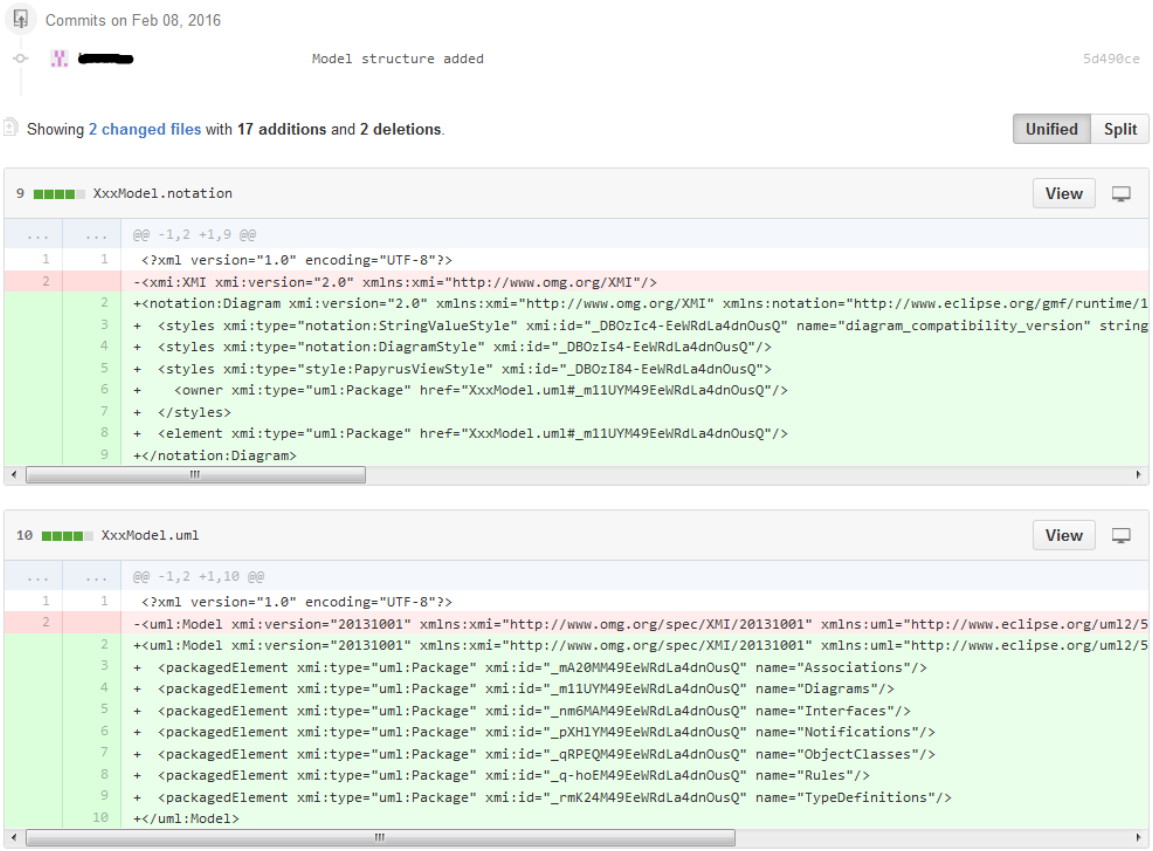


Figure 6-34: Detailed Comparison in Modeler's Remote Repository

The modeler can review the changes, add a comment to this updated version of the sub-model and then send the pull request by clicking on 

.

6.4.9 Merging with Remote Repository

The administrator of the SDO remote repository receives the pull request and can merge the updates into its repository.

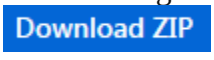
6.5 Downloading a Model from github for "Read Only Use"

This clause describes an easier way of getting the XxxModel to the local PC. This way is restricted to "read only viewers" of the model since it does not allow to commit changes back to GitHub.

The XxxModel repository is located in the SDO git space under the following URL: {XxxModel}.

Select the right branch 

, click the 

and 

buttons of the github web page to download the repository to the local PC.

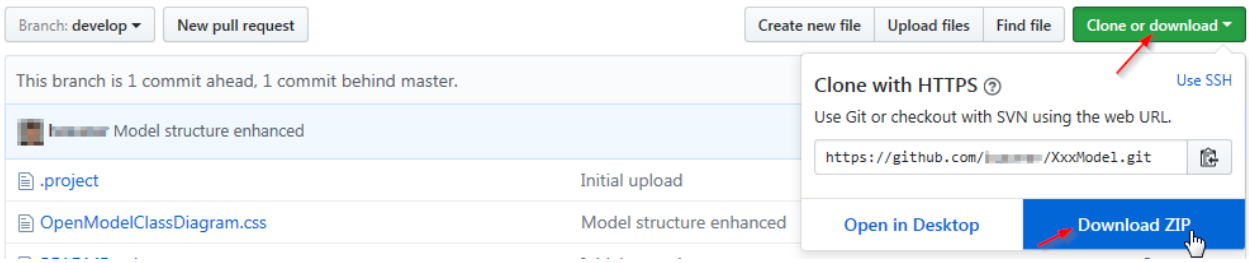


Figure 6-35: Download XxxModel Repository

Import the zip-file to the desired Eclipse Workspace.
Right click in the Project Explorer area opens the menu containing the Import...-button:

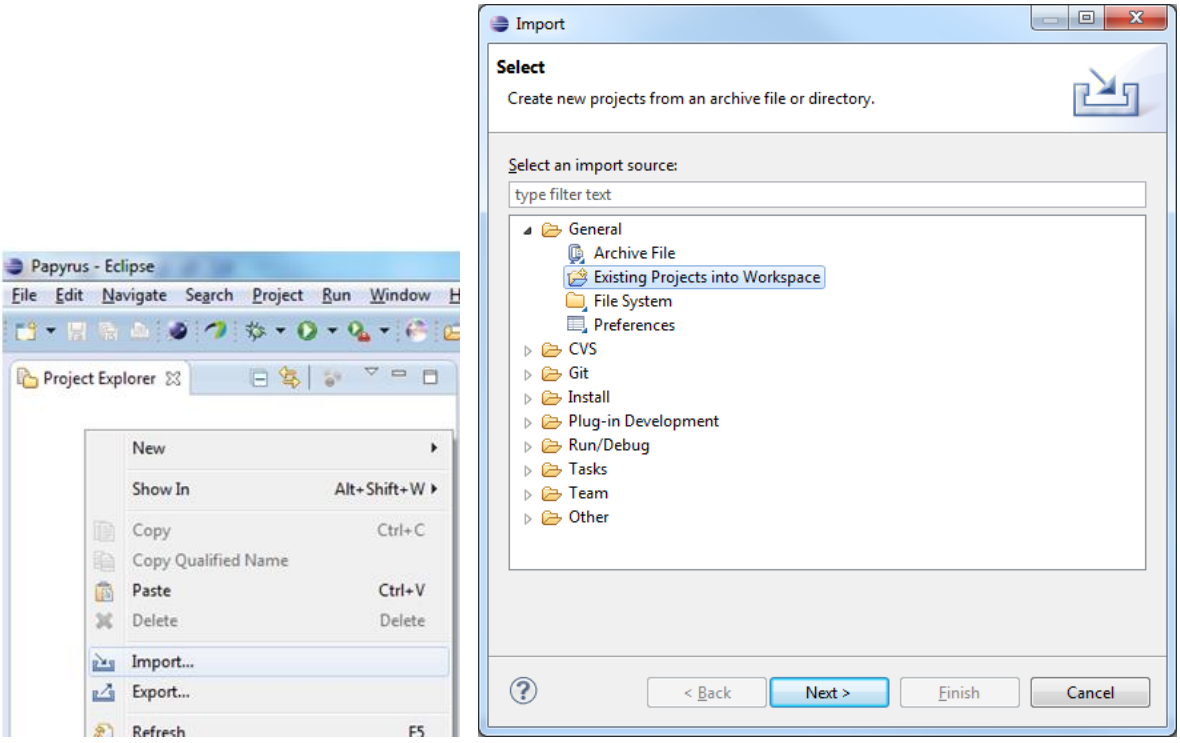


Figure 6-36: Importing the XxxModel into Papyrus (1)

Select the Existing Projects into Workspace option since the downloaded repository contains already the .project files for the model and profile.

Click Next > and then point via Browse... to the zip file downloaded in step 1.

The XxxModel is already selected in the Projects box.

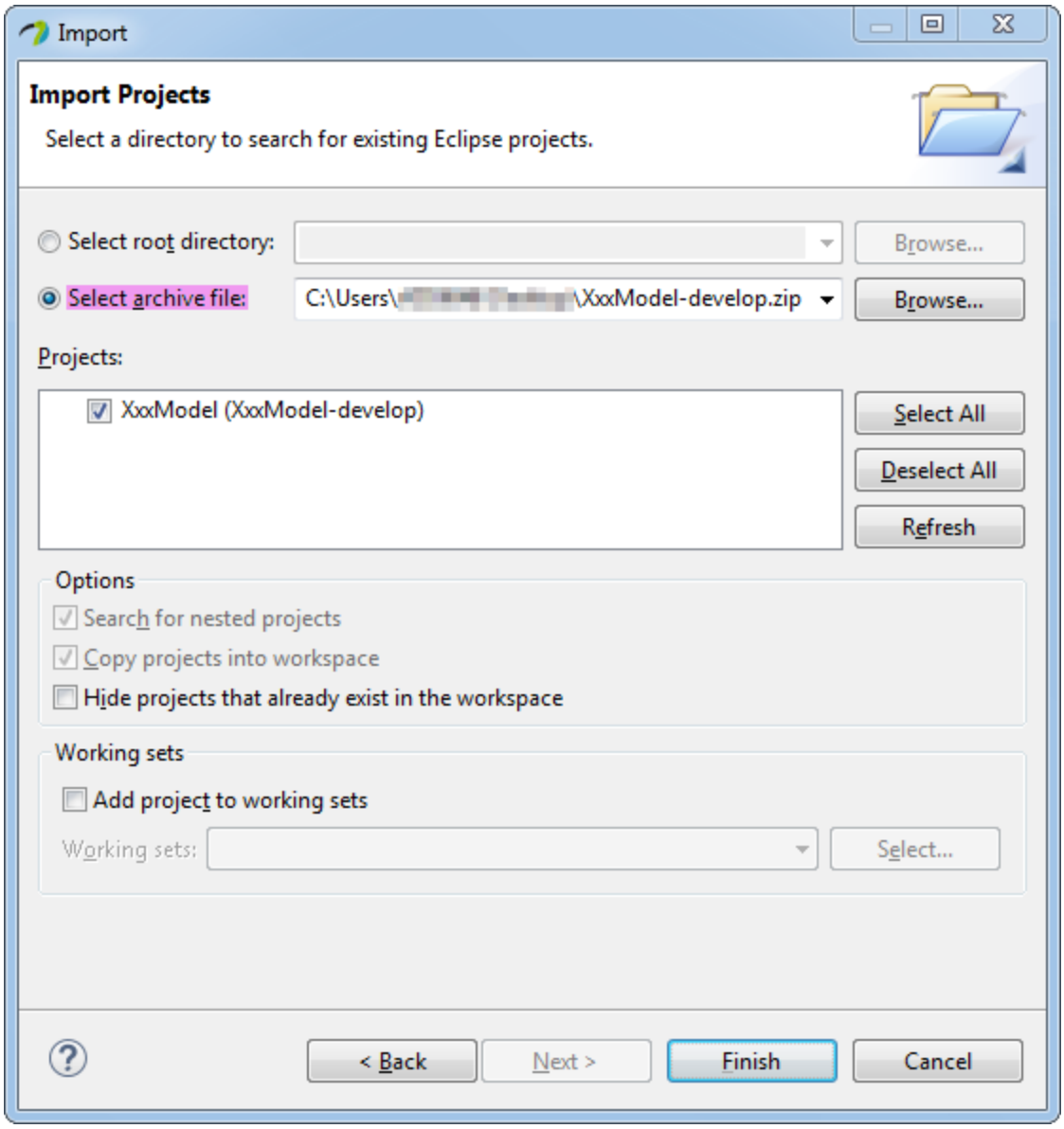


Figure 6-37: Importing the XxxModel into Papyrus (2)

Click Finish.

7 Using Papyrus

7.1 Illustrative Profile and Model

This guideline document uses an illustrative UML profile and an illustrative core-model and sub-model to explain the handling of Papyrus.

UML artifacts are defined by their properties (i.e., a kind of Meta Model). Standard properties are defined by the UML Specification [3] which are usually already supported by the UML tool (e.g., Papyrus). Additional specific properties are defined in a UML Profile (model).
The UML Guidelines document [4] describes the additional properties in detail.

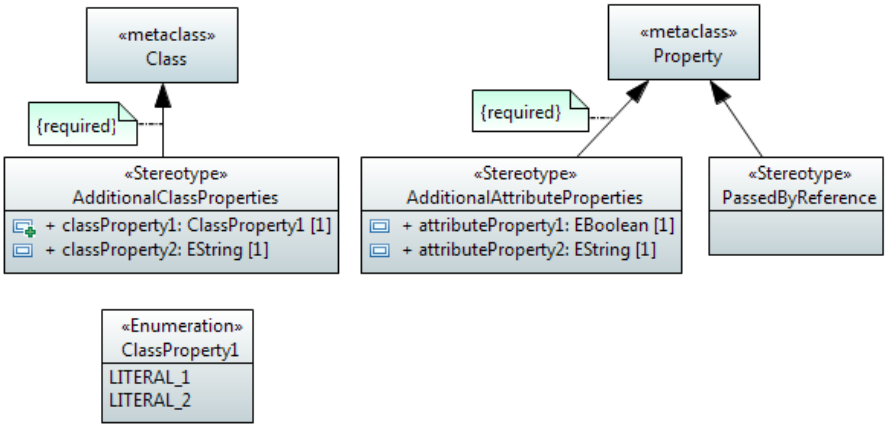


Figure 7-1: Illustrative UML Profile

The AdditionalClassProperties stereotype adds properties classProperty1 and classProperty2 to the object classes in the model. The extension relationship has been defined as “required” which adds the additional properties to all object classes; i.e., for every class created, the AdditionalClassProperties stereotype will be present by default.

The AdditionalAttributeProperties stereotype adds properties attributeProperty1 and attributeProperty2 to the attributes in the model. The extension relationship has been defined as “required”, which adds the additional properties to all attributes; i.e., for every attribute created, the AdditionalAttributeProperties stereotype will be present by default.

The PassedByReference stereotype identifies an attribute or an operation parameter being passed by value or passed by reference. The extension relationship has not been defined as “required”, which means that the stereotype has to be associated to the attribute on a case by case basis.
Note:
Only those attributes and operation parameters that refer to object classes may have the PassedByReference stereotype.

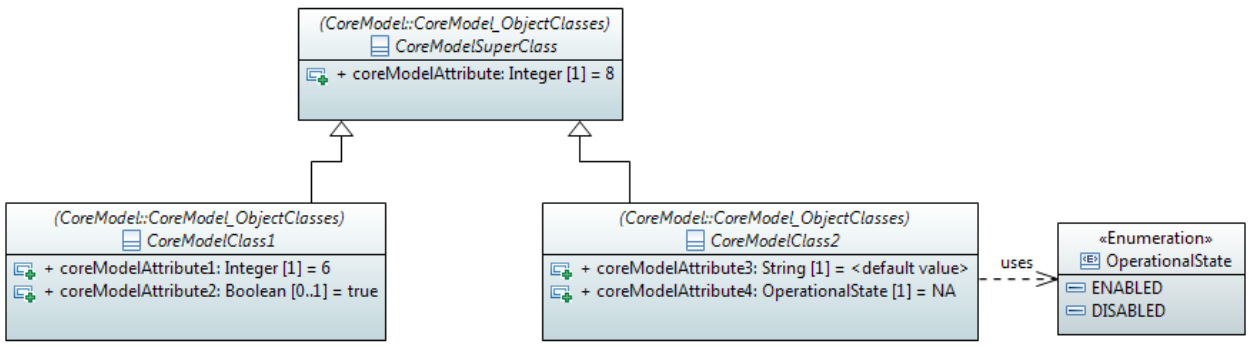


Figure 7-2: Illustrative Core Model

The initial core model contains a super-class and two sub-classes.

The profile from Figure 7-1 is associated to the model. This adds the additional properties to the artifacts in the model or allows their use in the model respectively.

It is possible to check if a profile is associated to the model (and which one) by clicking on XxxModel inside the Model Explorer and then click the **Profile** tab of the Properties view.

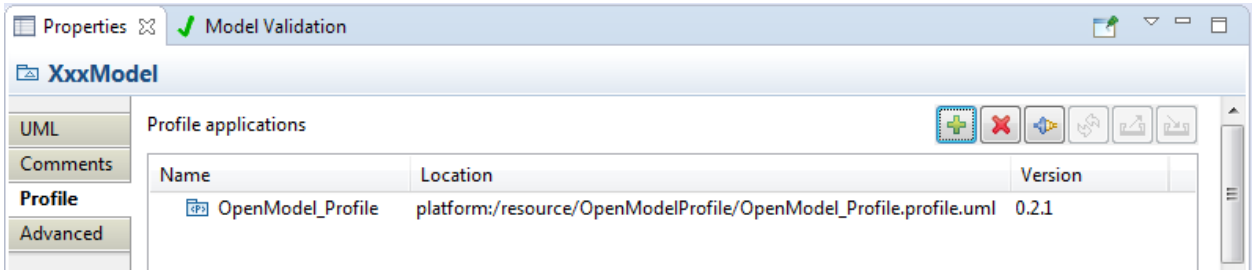
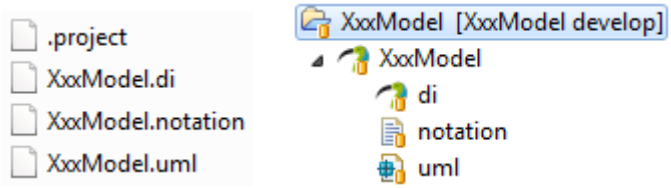


Figure 7-3: Profile Associated to the Model

7.2 Papyrus File Structure

A Papyrus model is stored in three different files (.di, .notation, .uml):



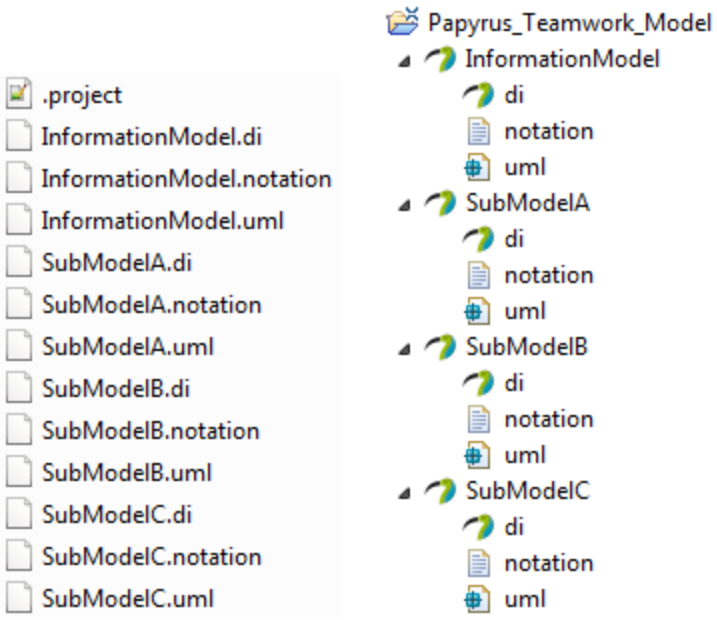
(Structure on the file system (left side); structure in the Papyrus Project Explorer (right side))

Figure 7-4: Papyrus File Structure

As already mentioned in clause 5.3, a model cannot exist on its own in Papyrus. It has to be contained by a “project”. A project can contain many models (i.e., multiple sets of .di, .notation, .uml files, as shown in Figure 7-5 below). The .project file contains the information about the project.

7.3 Model Splitting

Papyrus is able to split a UML model into different pieces (i.e., different files) allowing various teams to develop the model in a collaborative manner. The model pieces can be edited independently of the core model and then be re-merged with the core model.



(Structure on the file system (left side); structure in the Papyrus Project Explorer (right side))

Figure 7-5: Papyrus File Structure after Splitting

Each sub-model designer will be provided with all profile and model files to allow a comprehensive view (including cross-associations) on the Information Model at a given time of specification. Only the own sub-model files (.di, .notation, .uml) are writeable; all other files are write protected.

Write protected files shall not be changed.
Changes in the other parts of the model are only allowed by the respective model designer.

The sub-model designer shall be able to relate the sub-model object classes to the core object classes and to use the data types defined in the common data types library.
This is enabled by importing the core-model object classes and core-model type definitions into the sub-model. The common UML Primitive Types (i.e., Boolean, Integer, String) also need to be imported. Clause 6.4 (step 6.4.4) explains how to import additional artefacts.

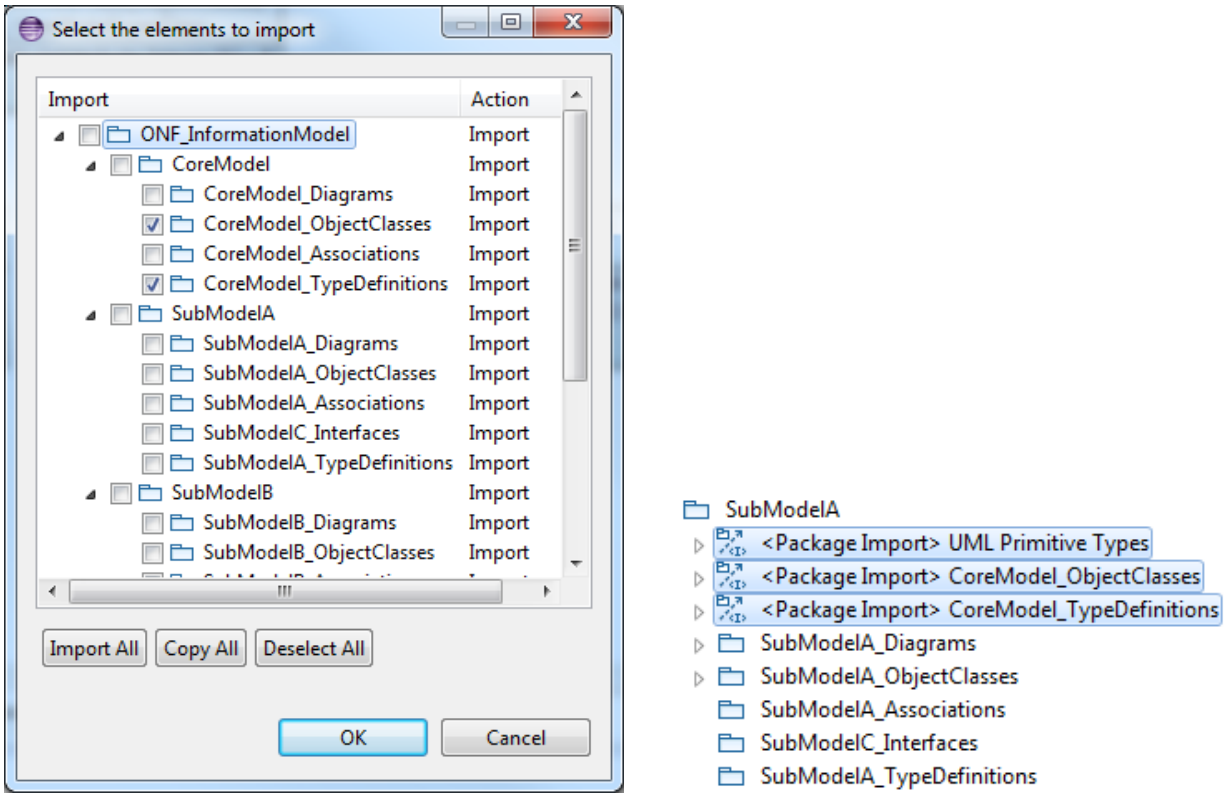


Figure 7-6: Imported UML Artifacts

Note:
In case one sub-model needs to refer to object classes or type definitions from another sub-model, these artifacts also need to be imported into the sub-model. In case such a definition is used in more than one other sub-model, the definition should be “elevated” to the core-model.

7.4 Constructing Modeling Environment for a new Model

This clause describes the prerequisites that have to be configured in order to define a new model. The following steps need to be executed:

1. Download model infrastructure files (CommonDataTypes, StyleSheets, Open Model Profiles) from Github
2. Create a new model in a new project
3. Copy downloaded model infrastructure files into new project
4. Apply the Open Model Profiles to the new model
5. Create Default Packages
6. Add the style sheet to the new model
7. Load CommonDataTypes libraries into the new model

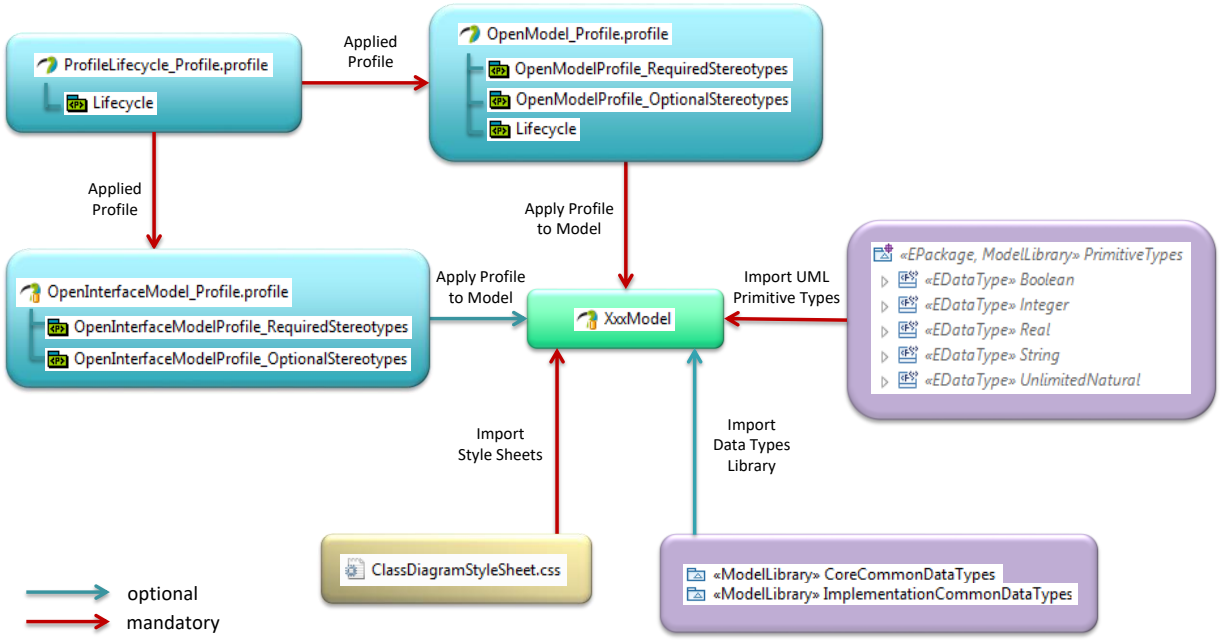


Figure 7-7: Building Modeling Infrastructure

7.4.1 Download model infrastructure files (CommonDataTypes, StyleSheets, Open Model Profiles) from Github

See clause 6.2: Steps 0 - 6.2.3.

After step 3 the Project Explorer should have this state:

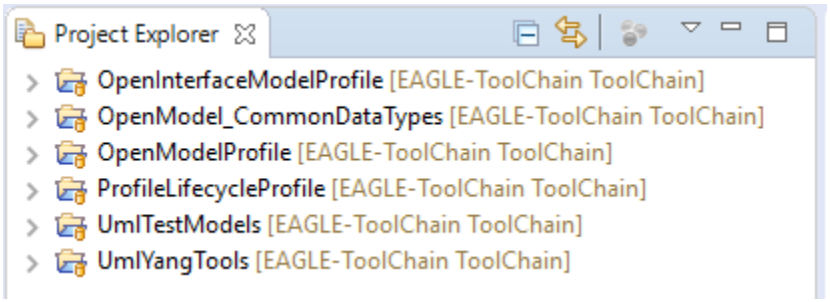


Figure 7-8: Downloaded Model Infrastructure Files

7.4.2 Create a new model in a new project

Right click in the Project Explorer area and select *New*, Papyrus Project :

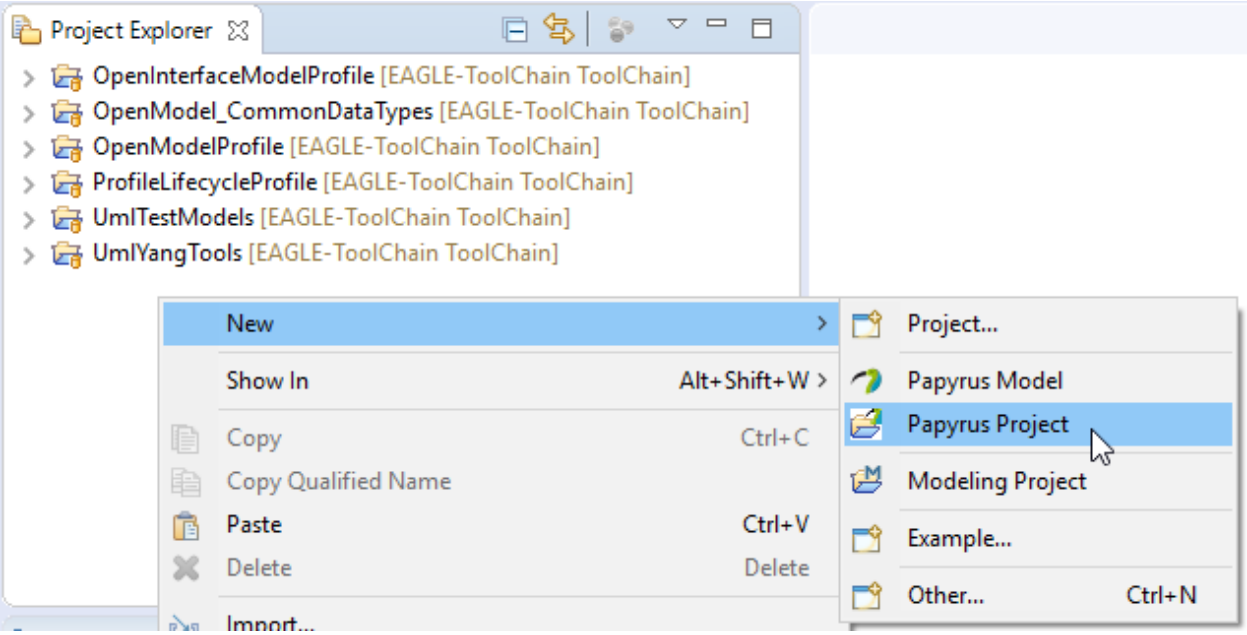


Figure 7-9: Creating a new Papyrus Project (1)

Select ☑ UML:

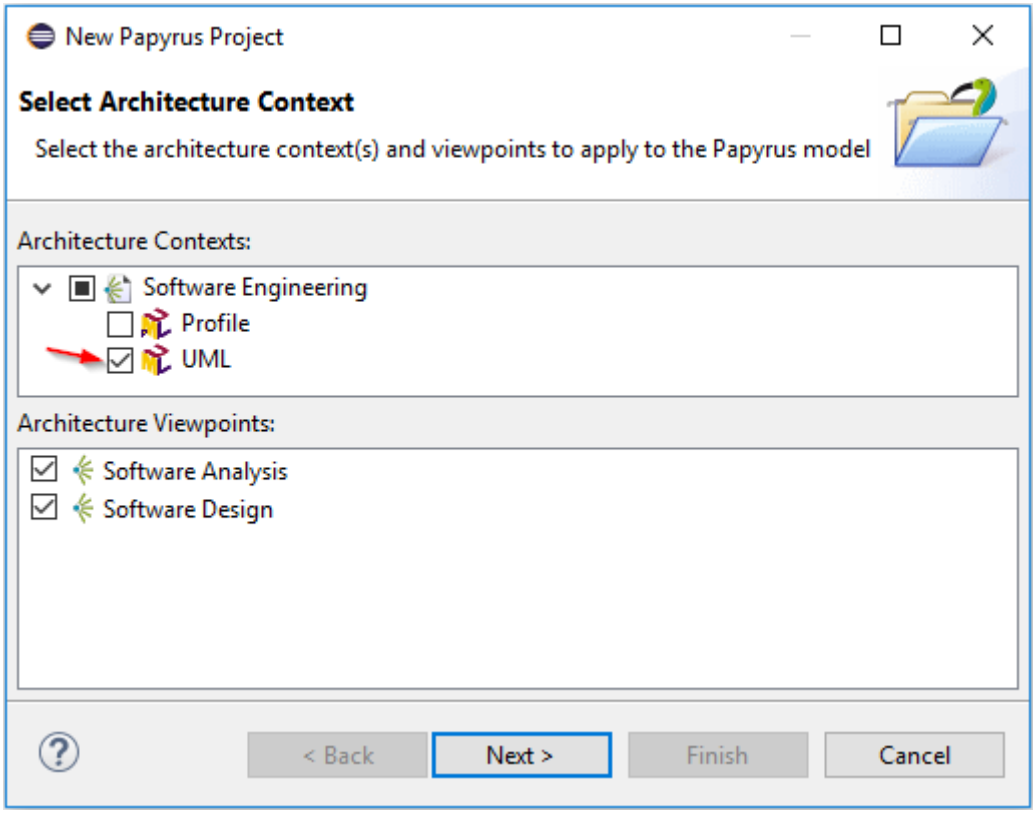
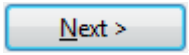


Figure 7-10: Creating a new Papyrus Project (2)

Click  , enter the project name and the model file name.

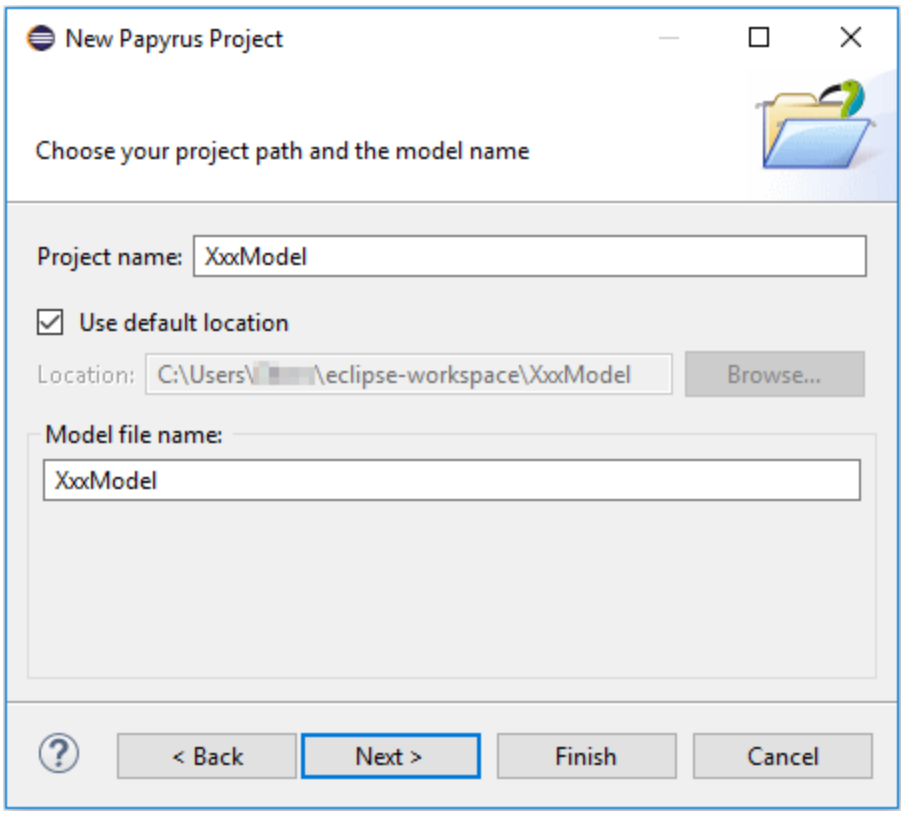


Figure 7-11: Creating a new Papyrus Project (3)

Click Next >, enter the root model name and check A UML model with basic primitive types which makes

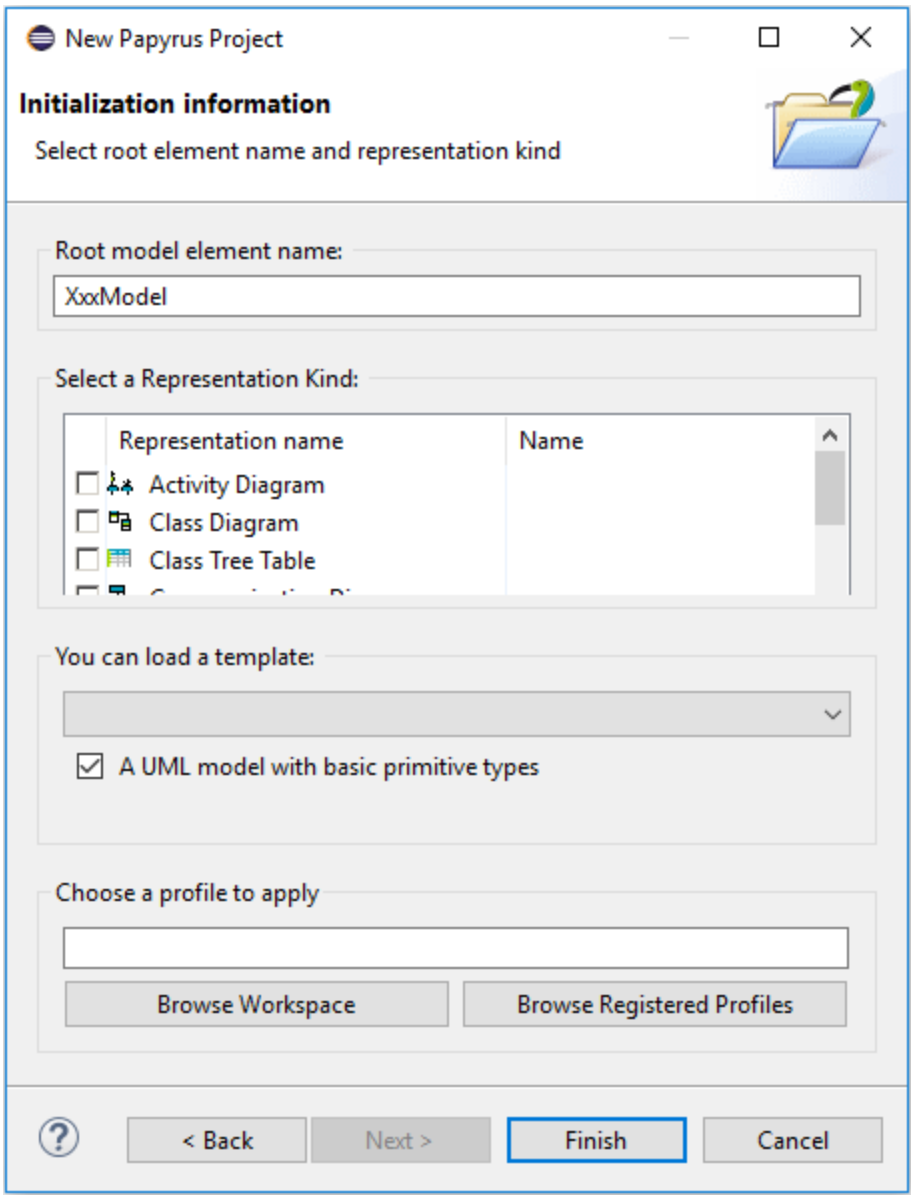
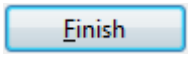


Figure 7-12: Creating a new Papyrus Project (4)

Click 

.

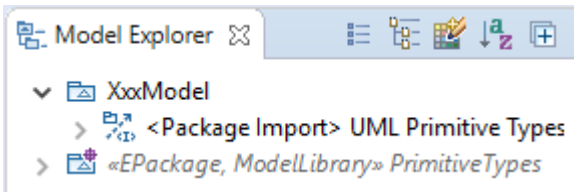


Figure 7-13: Creating a new Papyrus Project (5)

7.4.3 Copy downloaded model infrastructure files into new project

The model infrastructure files need to be copied into the new project,
to allow that different models in a workspace use different versions of the files.

The easiest way is to copy&paste the downloaded files directly using the Windows-Explorer (i.e., outside Papyrus).

Copy the CommonDataTypes and UmlProfiles folders from the downloaded EAGLE-ToolChain branch located in the git folder:

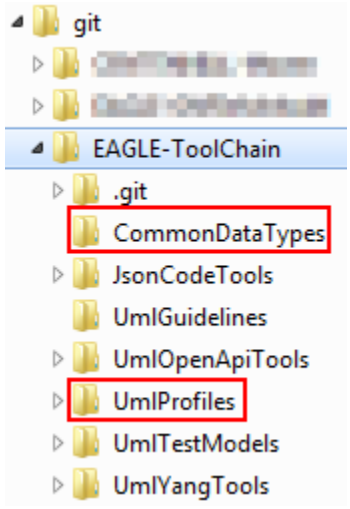


Figure 7-14: CommonDataTypes and UMLProfiles Folders on Local PC

Paste the CommonDataTypes and UmlProfiles folders into the XxxModel folder in the workspace:

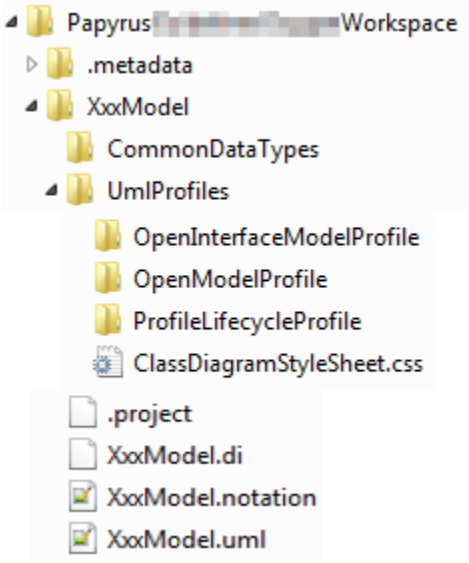


Figure 7-15: CommonDataTypes and UMLProfiles Folders pasted to Model Folder

Delete the .project files from the inserted folders:

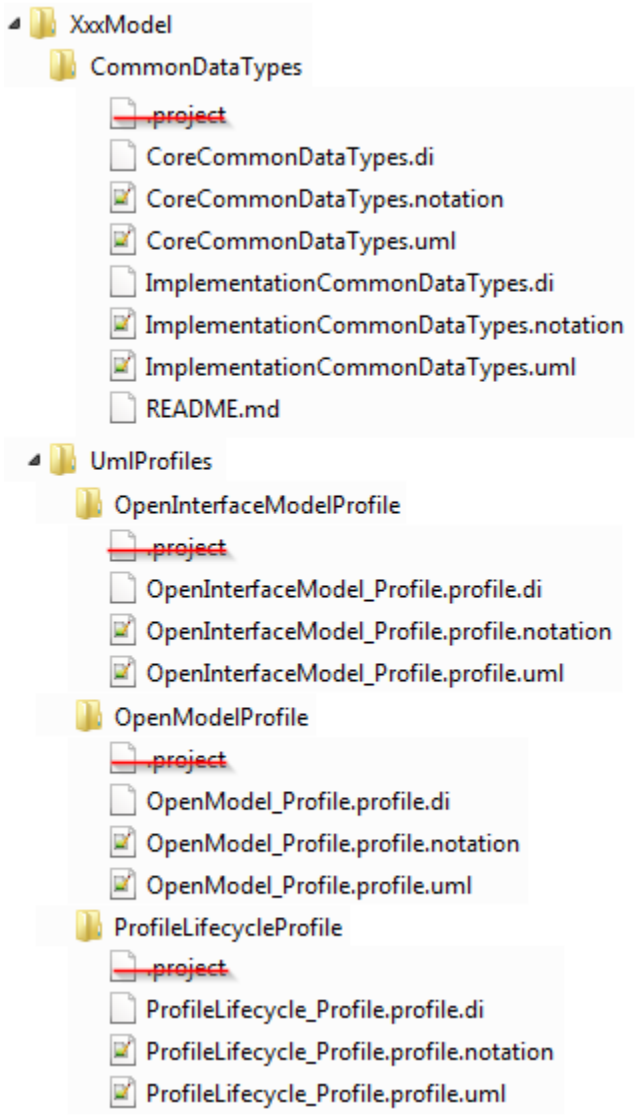


Figure 7-16: .project Files to be deleted from the Folders

Refresh the XxxModel in the Project Explorer of Papyrus to make the model infrastructure folders visible.

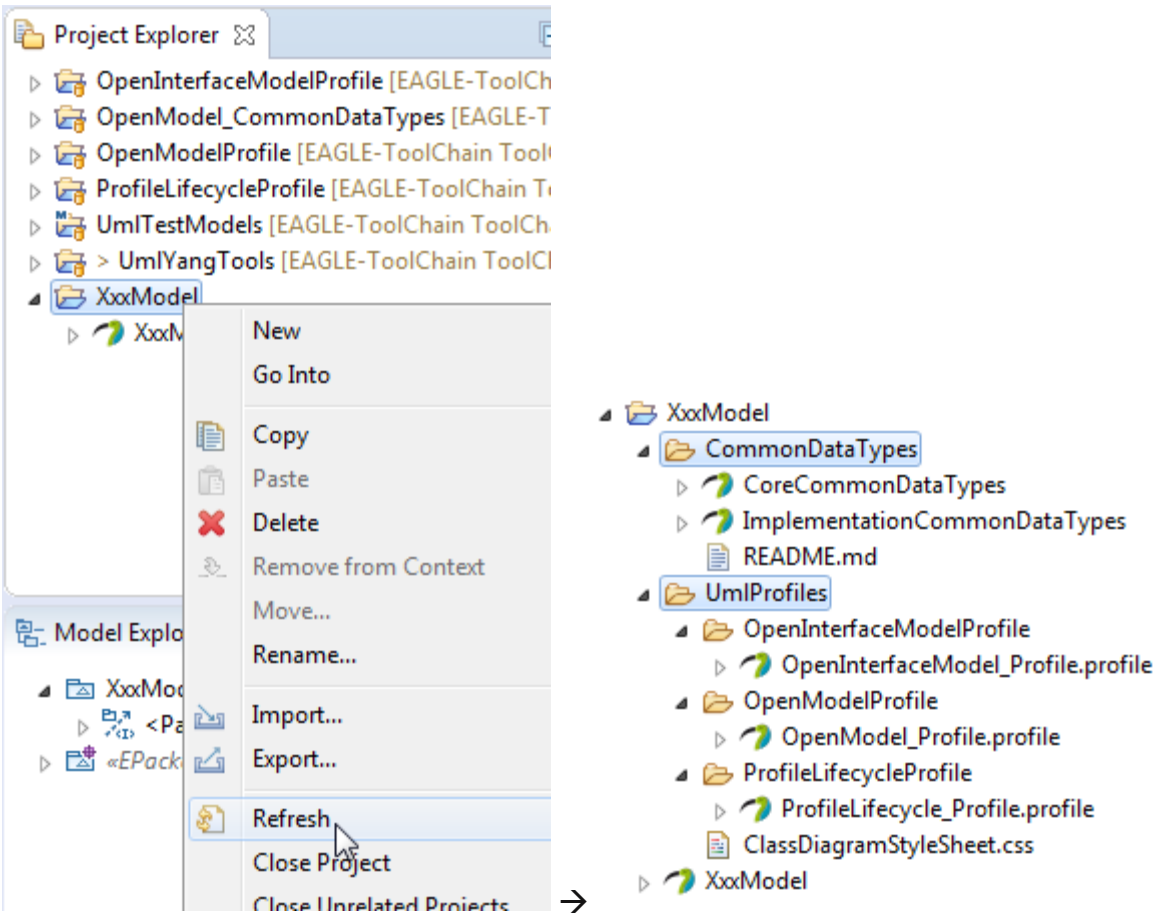


Figure 7-17: Model Infrastructure Files in XxxModel Project

7.4.4 Apply the Open Model Profiles to the new model

Select XxxModel in the Model Explorer, then select the *Profile* tab in Properties and click on the *Apply profile* button :

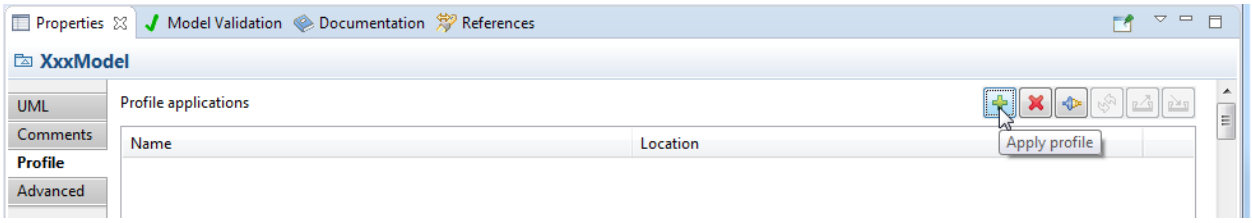


Figure 7-18: Applying Profiles (1)

Identify the three profiles and click OK:

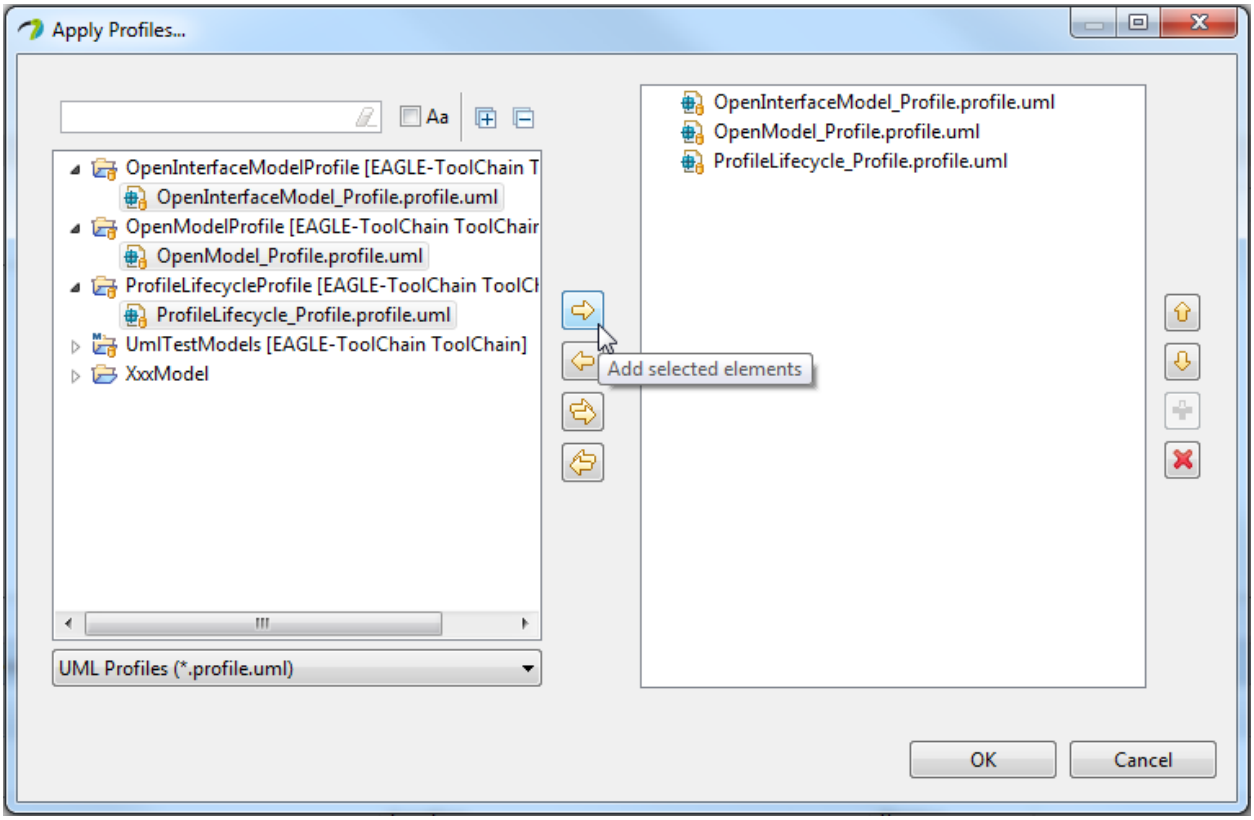
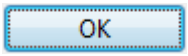


Figure 7-19: Applying Profiles (2)

Select all profiles and click 

:

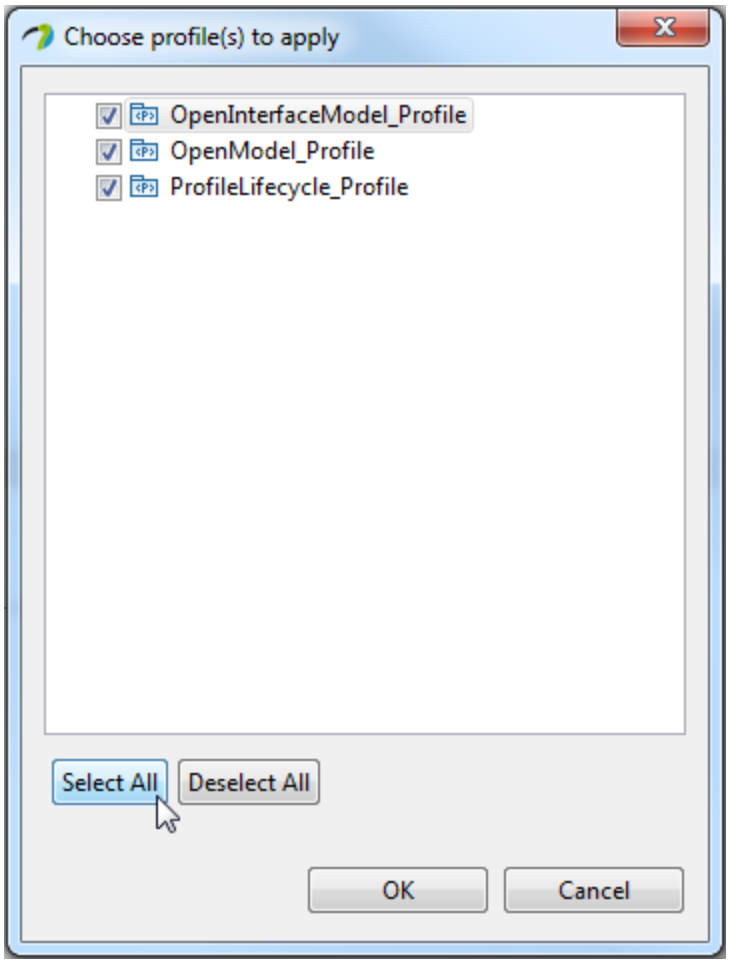


Figure 7-20: Applying Profiles (3)

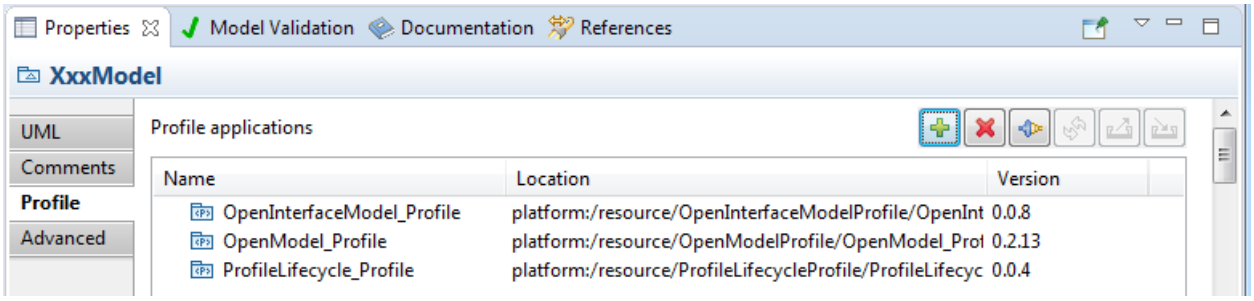


Figure 7-21: Profiles Applied

7.4.5 Create Default Packages

In order to reduce clutter, the UML artefacts are grouped in packages instead of having all kinds of the various artefacts mashed up at the same level. This provides a human friendly structure for the model. This structure accelerates the manual search for specific kinds of artefacts.

Right click on XxxModel and add all required default packages via *New Child* and Package:

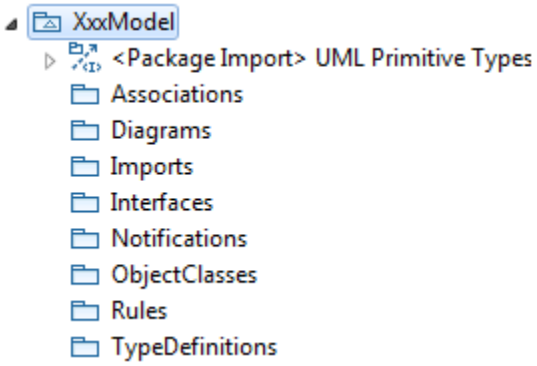


Figure 7-22: Default Package Structure

7.4.6 Add the Style Sheet to the new model

Create a class diagram in the Diagrams package via:

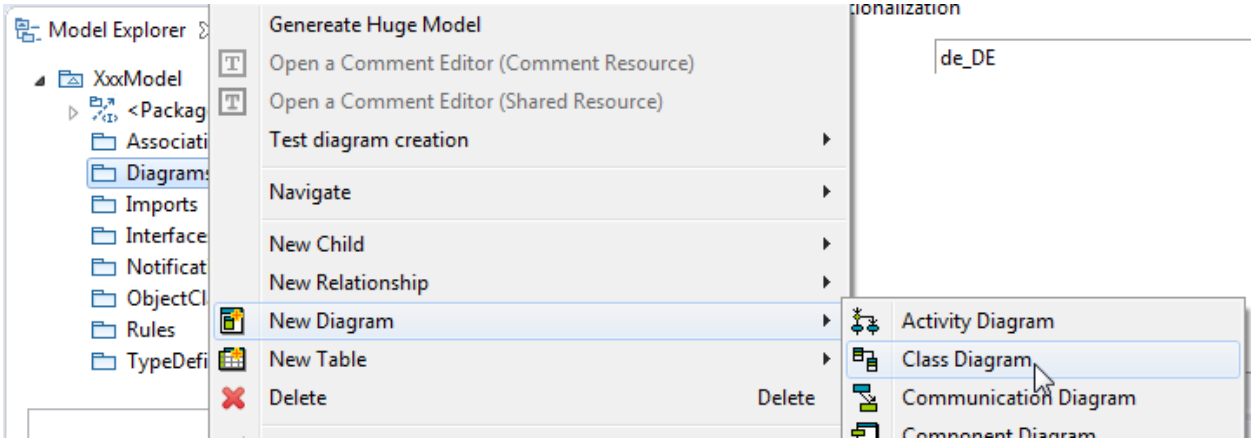


Figure 7-23: Creating a Class Diagram

Select the created NewClassDiagram in the Model Explorer, then select the *Style* tab in Properties and click on the *Add elements* button  of the **Model** *style sheets* box:

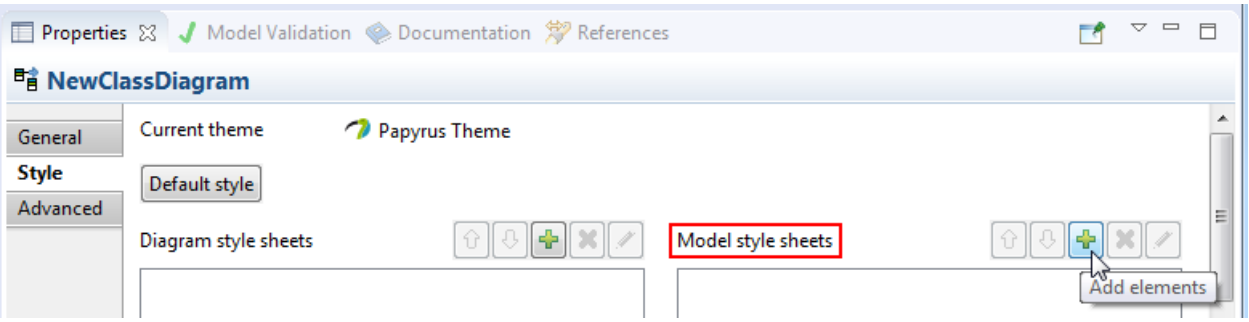
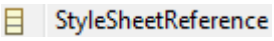


Figure 7-24: Adding Class Diagram Style Sheet (1)

Note: This guideline describes how to assign the style sheet to all diagrams in the model. It is also possible to assign a style sheet to an individual class diagram by clicking on the *Add elements* button  of the ***Diagram style sheets*** box.

Click the *Create a new element* button  , select  :

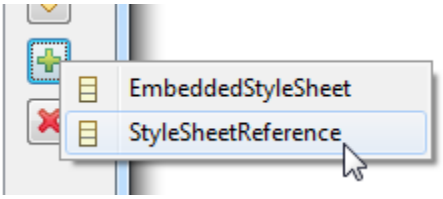
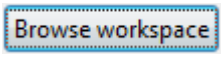
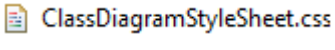


Figure 7-25: Adding Class Diagram Style Sheet (2)

Click  and select  file in the XxxModel:

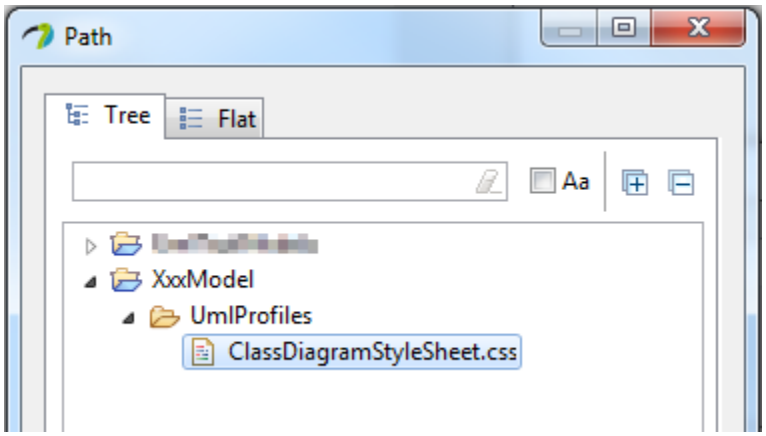


Figure 7-26: Adding Class Diagram Style Sheet (3)

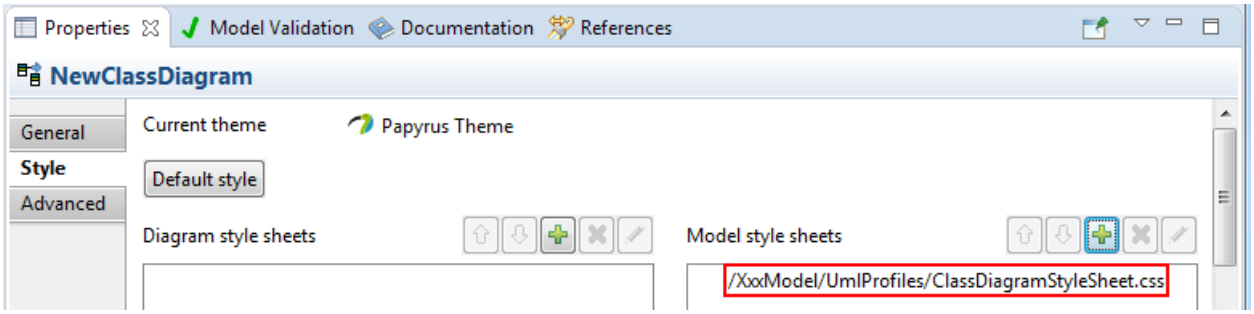


Figure 7-27: Adding Class Diagram Style Sheet (4)

7.4.7 Load CommonDataTypes libraries into the new model

Right-click on the model package XxxModel, select Import and then Import Package From User Model:

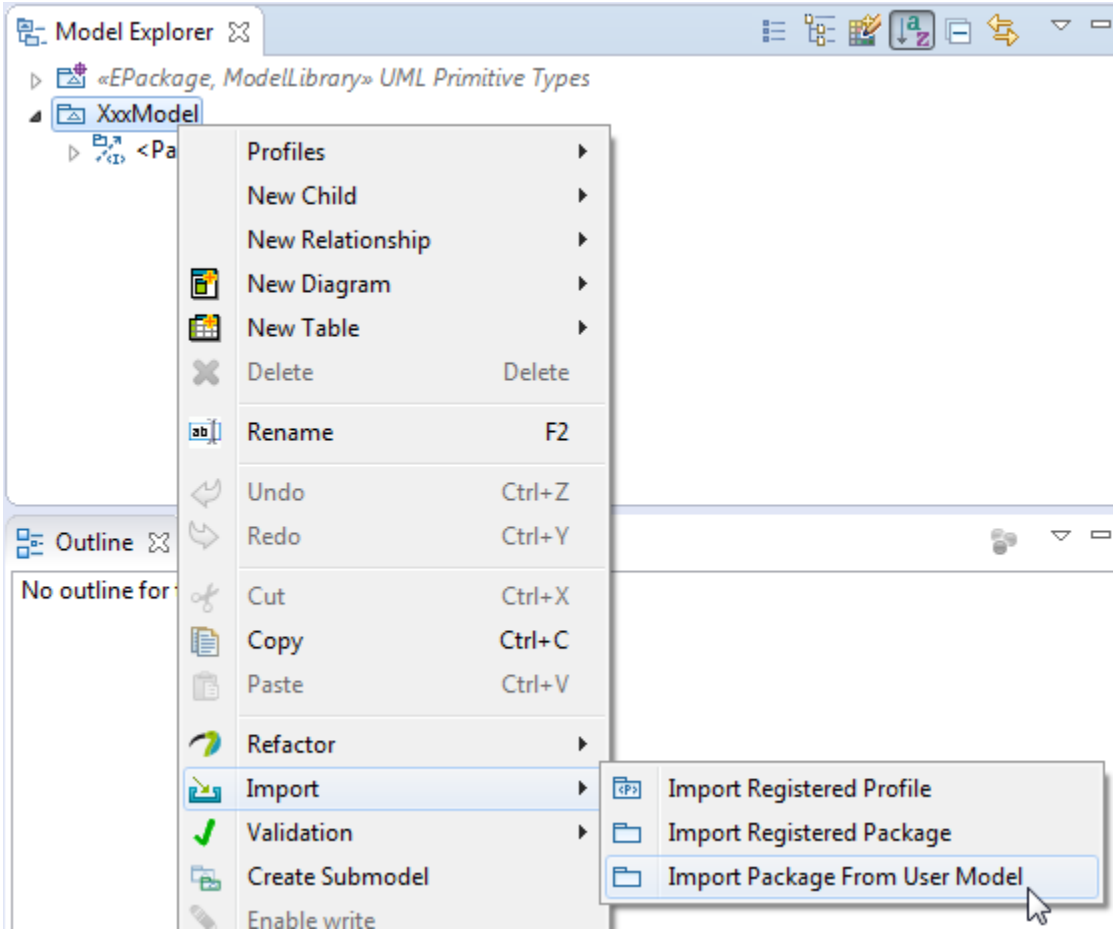


Figure 7-28: Loading CommonDataTypes Libraries (1)

Expand XxxModel and CommonDataTypes, select CoreCommonDataTypes.uml and ImplementationCommonDataTypes.uml and add them:

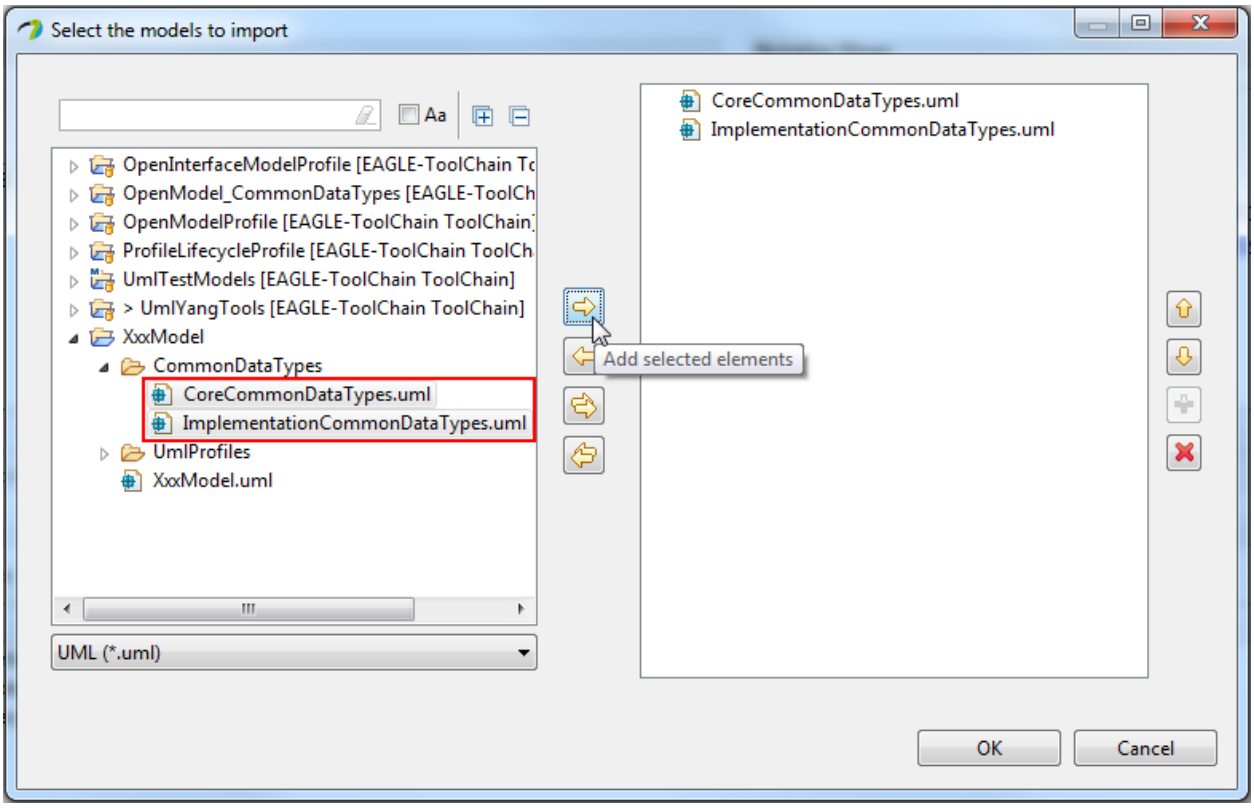


Figure 7-29: Loading CommonDataTypes Libraries (2)

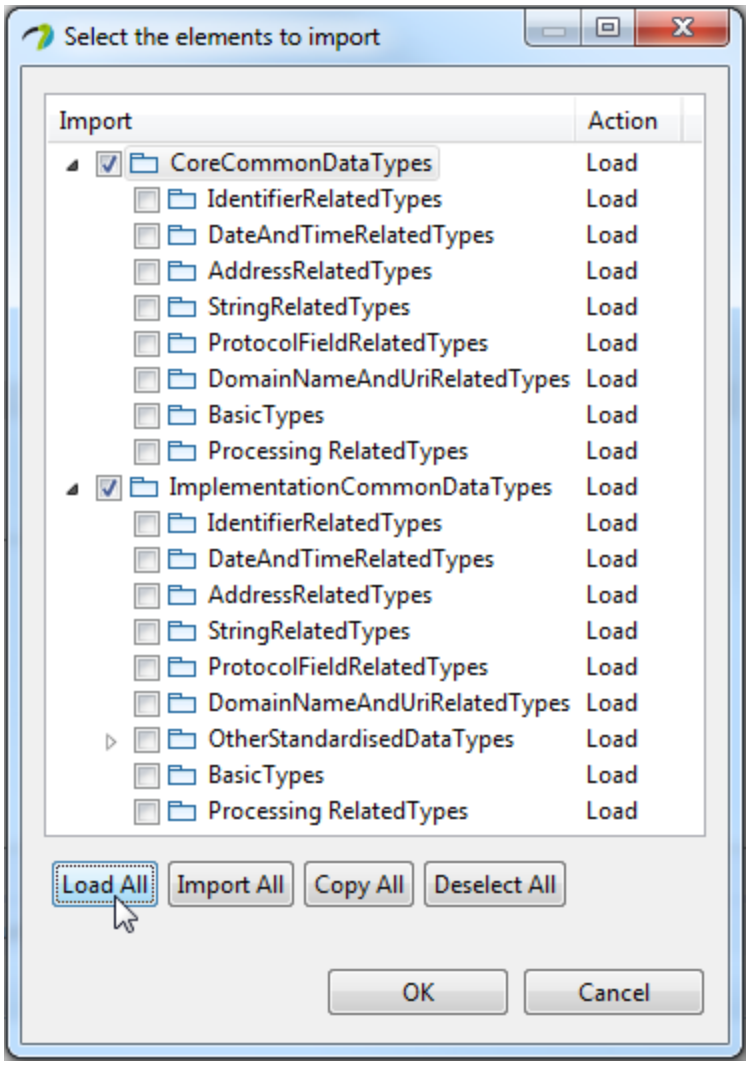
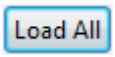


Figure 7-30: Loading CommonDataTypes Libraries (3)

It is possible to  or just a subset of the Data Types. This is based on the requirements of the new model.

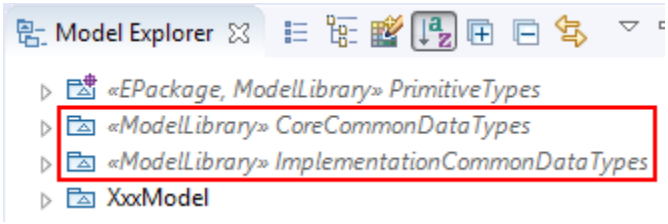


Figure 7-31: Loading CommonDataTypes Libraries (4)

The Common Data Types are now available for use in the XxxModel.

8 Generating Model Documentation

8.1 Introduction

This clause describes how to extract diagrams, comments and details for a Data Dictionary from a Papyrus model using the Gendoc plugin.

Editor's notes:
This clause is still a draft and will likely be changed in future versions.
Please check the known issues in clause 8.17 for any limitation which exists at the time.

A basic document generation tutorial is available at https://www.eclipse.org/gendoc/documentation/Gendoc_v0.6_tutorial.pdf. This provides detail in some areas but does not cover all aspects of usage. The following subclauses provide further guidance along with template fragments to assist understanding. A template that generates a normal form of model documentation is included for a dummy model.

Gendoc works with Microsoft Word and the template is a Word file. The template can be a mix of Gendoc script, normal text, Word figures, tables etc. Thus, if a word document with other non-model related information shall be created, but then have a clause specifically for the model, insert the "gendoc" related information as part of that Word document. The following clause builds up a template from the basic framing script through to a full template. The target document resulting from the gendoc template:

- Is produced in a form that can be published having all the necessary cover material, table of contents, headers, footers etc
- Contains specific figures extracted from the model in a specific with additional interleaved description and word figures
- Contains a data dictionary clause

The template described does not:

- Take advantage of the package structure of the model
- Interleave class description text with figures
    - Instead the figures refer to the data dictionary clause for formal description and structure

It should be noted that at the time of writing this clause there are a number of know issues with Gendoc (highlighted at the end of this clause).

8.2 Template usage

The template is stored in a system folder accessible via an Eclipse project so that the template can be seen in the Papyrus Project Explorer. The folder could be a sub-folder of the system folder containing the project that includes the model to be documented. Alternatively, the template can be stored in a specific project that just includes the template. The template "points" to the model to be extracted via the <context model> statement. The template is highlighted with the right-click menu and the "Generate documentation using Gendoc" item is selected. This will run the template, i.e. Gendoc is initiated from the template NOT from the model.

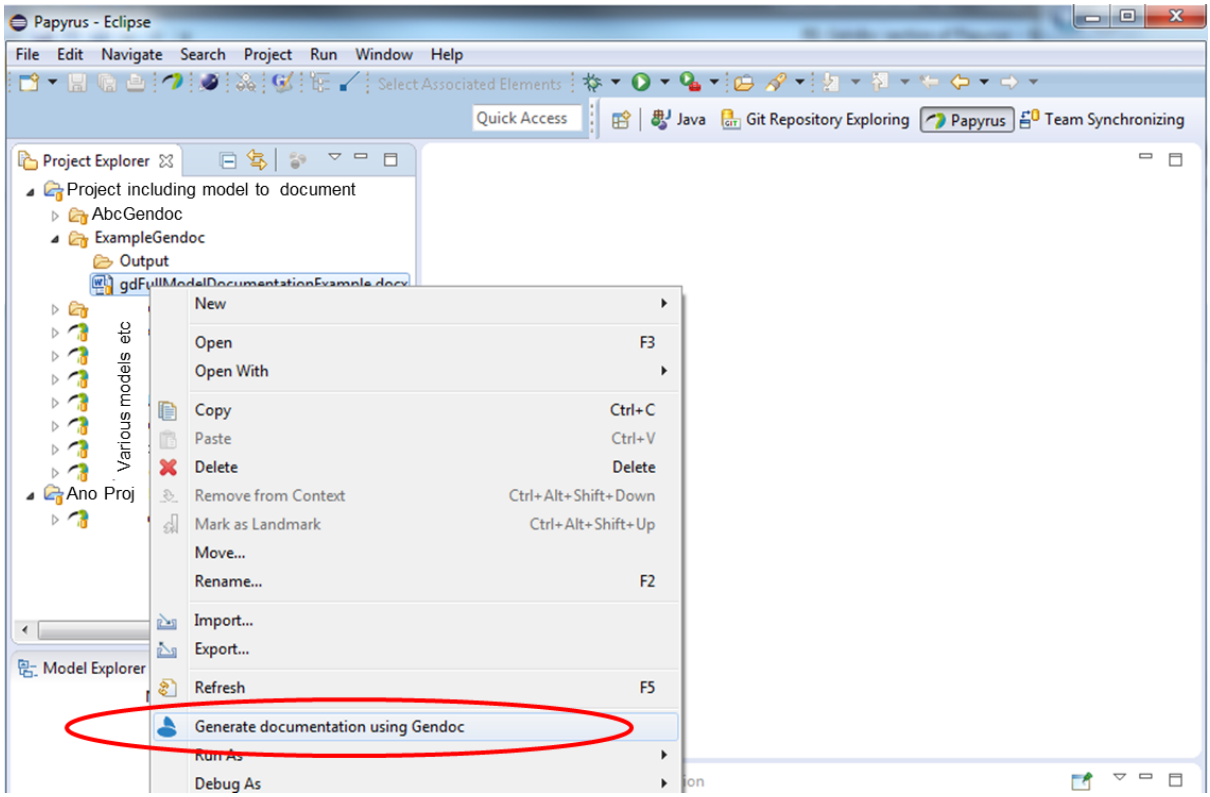


Figure 8-1: Initiating Gendoc for a particular template

The following clause explain how the template is targeted at the desired model.

8.3 Basic template

The template includes the name of the model to be documented and the name of the target model (substitute path and model name in the structure below). Both the .uml and .notation files are required.[3]

```
<config>
<output path='C:\Users\---appropriate path name--\ModelOutput.docx' />
</config>
<context model='C:\Users\---appropriate path name--\ModelName.notation' element='{0}'
importedBundles='gmf;papyrus' />
<gendoc><drop/>
Provides access to diagrams
</gendoc><drop/>
<context model='C:\Users\---appropriate path name--\ModelName.uml' element='{0}'
importedBundles='gmf;papyrus' />
<gendoc><drop/>
Provides access to class details
</gendoc><drop/>
```

[3] It is expected that the .notation file will eventually not be necessary and intertwining of class content and model diagrams will be more straight forward.

This particular template produces a document with the two black text items only. In the above example, the entire model “ModelName.uml” is taken as input. In order to select only one package in the model, one would set “element=’ModelName/{package name}’”.

8.4 Cover, contents, closing text etc

Any text and figures[4] inserted between the <gendoc> to </gendoc> space will be produced in the output.

```
<gendoc><drop/>
Any text and diagrams etc…
</gendoc><drop/>
```

8.5 Figures from the model with interleaved text

The figures can be extracted in alphabetical order from the model. The following script (in bold) will print the figure titles from the model in alphabetical order.

```
<config>
<output path=’C:\Users\---appropriate path name--\ModelOutput.docx' />
</config>
<context model=’C:\Users\---appropriate path name--\ModelName.notation' element=’{0}’
importedBundles='gmf;papyrus' />
<gendoc><drop/>
[for (d : notation::Diagram |notation::Diagram.allInstances()->sortedBy(name))]<drop/>
[d.name/]
[/for]<drop/>
</gendoc><drop/>
```

Clearly this by itself is not particularly useful but substituting the “[d.name/]” with the following bold script and replacing “specificDiagramName“ with the name (or unique substring of a name) of a diagram in the model will extract a specific named diagram (printing the diagram and its name)[5].

```
<gendoc><drop/>
Text and figures leading to diagram …
[for (d : notation::Diagram |notation::Diagram.allInstances()->sortedBy(name))]<drop/>
[if d.name.contains('specificDiagramName')] [d.name/]
<drop/>



<image object='[d.getDiagram()/]' maxW='true' keepH='false' keepW=’false’>
</image>
```

Figure 1 [d.name/]

[4] Note that certain special characters such as “[“ should be avoided. Any issues will be covered in the “known issues” document.
[5] There are current

```
[else]<drop/>
[/if]<drop/>
[/for]<drop/>
More text and figures after diagram ….
</gendoc><drop/>
```

The yellow area highlights a frame (which is otherwise not obvious). This frame is where Gendoc will place the figure. The frame will need to be sized to the right width in an actual usage (shrunk here to reduce space used in this document). The script will allow Gendoc to adjust the height but forces it to not exceed maximum width.

The following template extract expands for two figures (and could clearly be expanded to cover many more.

```
<gendoc><drop/>
Text and figures leading to diagram …
[for (d : notation::Diagram |notation::Diagram.allInstances()->sortedBy(name))]<drop/>
[if d.name.contains('specificDiagramName')] [d.name/]
<drop/>

<image object='[d.getDiagram()/]' maxW='true' keepH='false' keepW=’false’>
</image>
```

Figure 1 [d.name/]

```
[else]<drop/>
[/if]<drop/>
[/for]<drop/>
More text and figures after diagram and leading to the second diagram
[for (d : notation::Diagram |notation::Diagram.allInstances()->sortedBy(name))]<drop/>
[if d.name.contains('anotherSpecificDiagramName')] [d.name/]
<drop/>

<image object='[d.getDiagram()/]' maxW='true' keepH='false' keepW=’false’>
</image>
```

Figure 2 [d.name/]

```
[else]<drop/>
[/if]<drop/>
[/for]<drop/>
More text and figures after diagram ….
</gendoc><drop/>
```

The model may have many figures, the remainder will be ignored. Clearly care needs to be taken to ensure that one figure does not have a name string that is a sub-string of another name.

8.6 Figure in alphabetical order with no interleaved specific text

Alternatively the simple loop can be used to list all figures in alphabetical order. These cannot have associated document text inserted automatically.

```
<config>
<output path='C:\Users\---appropriate path name--\ModelOutput.docx' />
</config>
<context model='C:\Users\---appropriate path name--\ModelName.notation' element='{0}'
importedBundles='gmf;papyrus' />
<gendoc><drop/>
[for (d : notation::Diagram |notation::Diagram.allInstances()->sortedBy(name))]<drop/>
[d.name/]



<image object='[d.getDiagram()/]' maxW='true' keepH='false' keepW='false'>
</image>
```

Figure 1 [d.name/]

```
[/for]<drop/>
</gendoc><drop/>
```

8.7 Further explanation of the script

A majority of the script used in the previous clauses should be relatively obvious (e.g. [for…]… [/for] loop and [if…]… [else]… [/if] nested structures. The specific contents of the for and if statements is covered adequately in the tutorial material referenced earlier. One key thing to highlight here is the use of <drop/>. This instruction causes Gendoc to not throw a line break for the line that has <drop/>. Many blank lines may be found in the output. This will normally be because one or more <drop/> instructions have been forgotten.

There is a peculiar behavior with diagrams that requires a <drop/> on the blank line prior to the <image…> command. Without this part (but not all) of the <image… > instruction is printed.

8.8 Test template for printing diagrams and associated text

The following embedded file contains script which when modified to select the right model (several places in the file) and output locations will print three diagrams from the model.

gdDiagList.docx

8.9 Data Dictionary template overview

In the following subclauses the template is extended to add data dictionary content. This particular example data dictionary includes Classes and Data Types along with their attributes and for each provides (as appropriate):

- Comments
- Properties
- Stereotypes

Note that the stereotypes examples are from the OpenModelProfile [7].

8.10 Adding the class and its stereotypes

Considering the basic template described above and replacing the text “Provides access to class details” with the clause in bold below will provide the class name, class comments comments and the class stereotypes

```
<config>
<output path=’C:\Users\---appropriate path name--\ModelOutput.docx' />
</config>
<context model=’C:\Users\---appropriate path name--\ModelName.notation' element=’{0}’
importedBundles='gmf;papyrus' />
<gendoc><drop/>
Provides access to diagrams
</gendoc><drop/>
<context model=’C:\Users\---appropriate path name--\ModelName.uml' element=’{0}’
importedBundles='gmf;papyrus' />
<gendoc><drop/>
[for (cl:Class | Class.allInstances()->sortedBy(name))]<drop/>
[cl.name/]
[for (co:Comment | cl.ownedComment)]<drop/>
<dropEmpty>[co._body.clean()/]</dropEmpty>
[/for]<drop/>
Applied stereotypes:
[for (st:Stereotype | cl.getAppliedStereotypes())]<drop/>

    • [st.name/]
[for (oa:Property|st.ownedAttribute)]<drop/>
        • [if (not oa.name.contains('base'))][oa.name/]: [if (not cl.getValue(st,
          oa.name).oclIsUndefined())][if oa.name.contains('condition')][cl.getValue(st,
          oa.name).oclAsType(String)/] [else][cl.getValue(st,
          oa.name).oclAsType(EnumerationLiteral).name/][/if][else]<drop/>[/if]
[/if] <drop/>
[/for]<drop/>
[/for]<drop/>
[/for]<drop/>
</gendoc><drop/>
```

Note that:

- The classes are in alphabetical order
- The comment body is not printed if empty using the <dropEmpty>..</dropEmpty> instruction
- Any properties of stereotypes that have “base” in their name are not printed
- “conditions” are only printed if there is a specified condition
- The [cl.name/] would be expected to be a heading in a normal document

8.11 Adding properties and stereotypes in tabular form

The script from the previous clause has been extended with the additional script highlighted below in bold (other than the table contents). The table produced by the script here includes an explicit structuring of the stereotypes.

<config>
<output path='C:\Users\---appropriate path name--\ModelOutput.docx' />
</config>
<context model='C:\Users\---appropriate path name--\ModelName.notation' element='{0}' importedBundles='gmf;papyrus' />
<gendoc><drop/>
Provides access to diagrams
</gendoc><drop/>
<context model='C:\Users\---appropriate path name--\ModelName.uml' element='{0}' importedBundles='gmf;papyrus' />
<gendoc><drop/>
[for (cl:Class | Class.allInstances()->sortedBy(name))]<drop/>
[cl.name/]
[for (co:Comment | cl.ownedComment)]<drop/>
<dropEmpty>[co._body.clean()/]</dropEmpty>
[/for]<drop/>
Applied stereotypes:
[for (st:Stereotype | cl.getAppliedStereotypes())]<drop/>

- [st.name/]

[for (oa:Property|st.ownedAttribute)]<drop/>

- [if (not oa.name.contains('base'))][oa.name/]: [if (not cl.getValue(st, oa.name).oclIsUndefined())][if oa.name.contains('condition')][cl.getValue(st, oa.name).oclAsType(String)/] [else][cl.getValue(st, oa.name).oclAsType(EnumerationLiteral).name/][/if][else]<drop/>[/if]

[/if] <drop/>
[/for]<drop/>
[/for]<drop/>
[if cl.ownedAttribute->notEmpty()]<drop/>

Table 1: Attributes for [cl.name/]

<drop/>
<table><drop/>

Attribute Name	Type	Multiplicity	Access	Stereotypes	Description
[for (p:Property|cl.ownedAttribute)]<drop/>					
[p.name/]	[p.type.name/]	[if(p.lower=p.upper)]1[else][p.lower/]..[if(p.upper=-1)]*[else][p.upper/][/if][/if]	[if(not(p.isReadOnly))]RW[else]R[/if]	[for (st:Stereotype | p.getAppliedStereotypes())]<drop/> [st.name/] [for(oa:Property|st.ownedAttribute)]<drop/> • [if oa.name.contains('attribute')]AVC: [p.getValue(st, oa.name).oclAsType(EnumerationLiteral).name/] [else]<drop/> • [if oa.name.contains('invariant')]isInvariant: [p.getValue(st, oa.name).oclAsType(Boolean)/] [else]<drop/> • [if oa.name.contains('value')]valueRange: [if (not p.getValue(st, oa.name).oclIsUndefined())][p.getValue(st, oa.name).oclAsType(String)/][else] no range constraint [/if] [else]<drop/> • [if oa.name.contains('support')]support: [p.getValue(st, oa.name).oclAsType(EnumerationLiteral).name/] [else]<drop/> • [if oa.name.contains('condition')][if (not p.getValue(st, oa.name).oclIsUndefined())]condition:[p.getValue(st, oa.name).oclAsType(String)/][else]<drop/> [/if] [else]<drop/> [/if]<drop/> [/if]<drop/> [/if]<drop/> [/if]<drop/> [/if]<drop/> [/for]<drop/> [/for]<drop/>	[for (c:Comment | p.ownedComment)]<drop/> [c._body.clean()/] [/for]

[/for]<drop/>
</table><drop/>
[else][/if]<drop/>
[/for]<drop/>

</gendoc><drop/>

Note that this and following clauses have been reoriented to landscape as is advisable in a document when producing attribute data dictionary content.

8.12 Adding complex data types

A complex data types have a very similar structure to a class and hence the Gendoc commands are similar. The following snippet highlights in bold the differences between the class script above and the data type script replacing from "[for (cl:Class | Class.allInstances()->sortedBy(name))]<drop/>"

[for **(dt:DataType | DataType.**allInstances()->sortedBy(name))]<drop/>
[if **dt.**oclIsTypeOf**(DataType)**]<drop/>
[**dt**.name/]
[for (co:Comment | **dt.**ownedComment)]<drop/>
<dropEmpty>[co._body.clean()/]</dropEmpty>
[/for]<drop/>
Applied Stereotypes:
[for (st:Stereotype | dt.getAppliedStereotypes())]<drop/>

- **[st.name/]**

[/for]<drop/>
[if **dt**.ownedAttribute->notEmpty()]<drop/>
Table 2: Attributes for [dt.name/]
<drop/>
<table><drop/>

Attribute Name	Type	Multiplicity	Access	Stereotypes	Description

[for (p:Property|dt.ownedAttribute)]<drop/>

…… then the table is the same as for the class attributes…

8.13 Adding other data types

8.13.1 Enumeration Types

The following script will extract all enumerations with their comments and list the literals with their comments for each. Clearly stereotypes can be extracted using fragments of script from earlier clauses.

[for (dt:DataType | DataType.allInstances()->sortedBy(name))]<drop/>
[if dt.oclIsTypeOf(Enumeration)]<drop/>
[dt.name/]
[for (co:Comment | dt.ownedComment)]<drop/>
<dropEmpty>[co._body.clean()/]</dropEmpty>
[/for]<drop/>
Contains Enumeration Literals:
[for (e:EnumerationLiteral|dt.oclAsType(Enumeration).ownedLiteral)]<drop/>

- [e.name/]:
  - [for (co:Comment | e.ownedComment)]<drop/>
  - <dropEmpty>[co._body.clean()/]
  - </dropEmpty>[/for]<drop/>

[/for]<drop/>
[else] [/if]<drop/>
[/for]<drop/>

8.13.2 Primitive Types

Primitive types can be listed with comments using the followings script.

[for (dt:DataType | DataType.allInstances()->sortedBy(name))]<drop/>
[if dt.oclIsTypeOf(PrimitiveType)]<drop/>
[dt.name/]
[for (co:Comment | dt.ownedComment)]<drop/>
<dropEmpty>[co._body.clean()/]</dropEmpty>
[/for]<drop/>

[else] [/if]<drop/>
[/for]<drop/>

8.14 Using Fragments

8.14.1 Fragment Definitions

8.14.1.1 Introduction

Gendoc provides the capability of reusing Gendoc scripts inside the same document by defining subroutines called “fragments”. It is possible to instruct the subroutines using transfer parameters.

The following clauses contain pre-defined fragments.

8.14.1.2 Insert Class Fragment

```
<fragment name=’insertClass’ importedBundles=’commons;gmf;papyrus’><drop/>
<arg name=’cl’ type=’uml::Class’/><drop/>
<arg name=’className’ type=’String’/><drop/>
<arg name=’packageName’ type=’String’/><drop/>
[if (not cl.qualifiedName.contains(packageName))]<drop/>
[else] <drop/>
[if(cl.name.contains(className))]<drop/>

Qualified Name: [cl.qualifiedName/]

[for (co:Comment | cl.ownedComment)]<drop/>

<dropEmpty>[co._body.clean()/]</dropEmpty>

[/for]<drop/>
[for (st:Stereotype | cl.getAppliedStereotypes())]<drop/>
[if(not st.name.contains(‘OpenModelClass’))]<drop/>

This class is [st.name/].

[else] <drop/>
[/if]<drop/>
[/for]<drop/>
[else] <drop/>
[/if]
[/if]
</fragment><drop/>
```

8.14.1.3 Insert Standard Diagram Fragment


```
<fragment name='insertStandardDiagram' importedBundles='commons;gmf;papyrus'><drop/>
<arg name='p' type='uml::Package'/><drop/>
<arg name='diagramName' type='String'/><drop/>
<arg name='diagramTitle' type='String'/><drop/>

[for (d:Diagram|p.getPapyrusDiagrams())]<drop/>
[if d.name.contains(diagramName)]
<drop/>
<image object='[d.getDiagram()/]' maxW='true' keepH='false' keepW = 'false'>
```

```
</image>
```

Figure 8-2 [diagramTitle/]

CoreModel diagram: [d.name/]

```
[else]<drop/>
[/if]<drop/>
[/for]<drop/>
</fragment><drop/>
```

8.14.1.4 Insert Small Diagram Fragment

```
<fragment name='insertSmallDiagram' importedBundles='commons;gmf;papyrus'><drop/>
<arg name='p' type='uml::Package'/><drop/>
<arg name='diagramName' type='String'/><drop/>
<arg name='diagramTitle' type='String'/><drop/>

[for (d:Diagram|p.getPapyrusDiagrams())]<drop/>
[if d.name.contains(diagramName)]
<drop/>
```


<image object='[d.getDiagram()/]' maxW='true' keepH='false' keepW = ‘false’>

</image>

Figure 8-3 [diagramTitle/]

CoreModel diagram: [d.name/]

[else]<drop/>
[/if]<drop/>
[/for]<drop/>
</fragment><drop/>

8.14.1.5 Insert Attribute Row Brief Fragment

<fragment name=’insertAttributeRowBrief’ importedBundles=’commons;gmf;papyrus’><drop/>
<arg name=’p’ type=’uml::Property’/><drop/>

[p.name/]	[for (st:Stereotype | p.getAppliedStereotypes())]<drop/> [if(not st.name.contains(‘OpenModelAttribute’))] [st.name/] [/if]<drop/> [/for]<drop/>	[if p.ownedComment->notEmpty()]<drop/> [for (c:Comment | p.ownedComment)] <drop/> [c._body.clean()/] [/for] [else] [if (p.name.contains (‘_’))] See referenced class [else] NO DESCRIPTION [/if]<drop/> [/if]<drop/>

</fragment><drop/>

8.14.1.6 Start Attribute Table Brief Fragment

<fragment name=’insertAttributeTableHeader’
importedBundles=’commons;gmf;papyrus’><drop/>
<arg name=’cl’ type=’uml::Class’/><drop/>

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description

</fragment><drop/>

8.14.1.7 Insert Attribute Table Brief Fragment

```
<fragment name=’insertAttributeTableBrief’ importedBundles=’commons;gmf;papyrus’
importedFragments='insertAttributeTableHeader;insertAttributeRowBrief’><drop/>
<arg name=’cl’ type=’uml::Class’/><drop/>
[if  cl.ownedAttribute->notEmpty()]<drop/>
```

Table 3: Attributes for [cl.name/]

```
<table><drop/>
[cl.insertAttributeTableHeader ()/]
[for (p:Property|cl.ownedAttribute)]<drop/>
[if (not p.name.contains(‘_’))]<drop/>
[p.insertAttributeRowBrief ()/]
[/if]<drop/>
[/for]<drop/>
[for (p:Property|cl.ownedAttribute)]<drop/>
[if (p.name.contains(‘_’))]<drop/>
[p.insertAttributeRowBrief ()/]
[/if]<drop/>
[/for]<drop/>
</table><drop/>
[/if]<drop/>
</fragment><drop/>
```

8.14.1.8 Insert Ten Specified Attribute Table Brief Fragment

```
<fragment name=’insertTenSpecifiedAttributeTableBrief’
importedBundles=’commons;gmf;papyrus’
importedFragments='insertAttributeTableHeader;insertAttributeRowBrief’><drop/>
<arg name=’cl’ type=’uml::Class’/><drop/>
<arg name=’p1’ type=‘String’/><drop/>
<arg name=’p2’ type=‘String’/><drop/>
<arg name=’p3’ type=‘String’/><drop/>
<arg name=’p4’ type=‘String’/><drop/>
<arg name=’p5’ type=‘String’/><drop/>
<arg name=’p6’ type=‘String’/><drop/>
<arg name=’p7’ type=‘String’/><drop/>
<arg name=’p8’ type=‘String’/><drop/>
<arg name=’p9’ type=‘String’/><drop/>
<arg name=’p10’ type=‘String’/><drop/>
[if  cl.ownedAttribute->notEmpty()]<drop/>
```

Table 4: Attributes for [cl.name/]

```
<table><drop/>
[cl.insertAttributeTableHeader ()/]
[for (p:Property|cl.ownedAttribute)]<drop/>
[if  (p.name.contains(p1) or p.name.contains(p2) or p.name.contains(p3) or p.name.contains(p4)
or p.name.contains(p5) or p.name.contains(p6) or p.name.contains(p7) or p.name.contains(p8) or
p.name.contains(p9) or p.name.contains(p10))]<drop/>
[if (not p.name.contains('_'))]<drop/>
[p.insertAttributeRowBrief ()/]
[/if]<drop/>
[/if]<drop/>
[if  (p.name.contains(p1) or p.name.contains(p2) or p.name.contains(p3) or p.name.contains(p4)
or p.name.contains(p5) or p.name.contains(p6) or p.name.contains(p7) or p.name.contains(p8) or
p.name.contains(p9) or p.name.contains(p10))]<drop/>
[if (p.name.contains('_'))]<drop/>
[p.insertAttributeRowBrief ()/]
[/if]<drop/>
[/if]<drop/>
[/for]<drop/>
</table><drop/>
[/if]<drop/>
</fragment><drop/>
```

8.14.1.9 Insert Data Type Fragment

```
<fragment name='insertDataType' importedBundles='commons;gmf;papyrus'><drop/>
<arg name='dt' type='uml::DataType'/><drop/>
<arg name='dataTypeName' type='String'/><drop/>
<arg name='packageName' type='String'/><drop/>
[if (dt.qualifiedName.contains(packageName))]<drop/>
[if(dt.name.contains(dataTypeName))]<drop/>
```

Qualified Name: [dt.qualifiedName/]

```
[for (co:Comment | dt.ownedComment)]<drop/>

<dropEmpty>[co._body.clean()/]</dropEmpty>

[/for]<drop/>
[for (st:Stereotype | dt.getAppliedStereotypes())]<drop/>
```

This class is [st.name/].

```
[/for]<drop/>
[else] <drop/>
[/if]
[/if]
</fragment><drop/>
```

8.14.1.10 Start Data Type Attribute Table Brief Fragment

```
<fragment name=’insertDataTypeAttributeTableHeader’
importedBundles=’commons;gmf;papyrus’><drop/>
<arg name=’dt’ type=’uml::DataType’/><drop/>
```

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description

```
</fragment><drop/>
```

8.14.1.11 Insert Data Type Attribute Table Brief Fragment

```
<fragment name=’insertDataTypeAttributeTableBrief’
importedBundles=’commons;gmf;papyrus’
importedFragments='insertDataTypeAttributeTableHeader;insertAttributeRowBrief’><drop/>
<arg name=’dt’ type=’uml::DataType’/><drop/>
[if  dt.ownedAttribute->notEmpty()]<drop/>
```

Table 5: Attributes for [dt.name/]

```
<table><drop/>
[dt.insertDataTypeAttributeTableHeader ()/]
[for (p:Property|dt.ownedAttribute)]<drop/>
[p.insertAttributeRowBrief ()/]
[/for]<drop/>
</table><drop/>
[/if]<drop/>
</fragment><drop/>
```

8.14.1.12 Insert Classes (DD only) Fragment

<fragment name=’insertClasses’ importedBundles=’commons;gmf;papyrus’><drop/>
<arg name=’cl’ type=’uml::Class’/><drop/>

<Word section heading> [cl.name/]

Qualified Name: [cl.qualifiedName/]

[for (co:Comment | cl.ownedComment)]<drop/>
<dropEmpty>[co._body.clean()/]</dropEmpty>
[/for]<drop/>

Applied stereotypes:
[for (st:Stereotype | cl.getAppliedStereotypes())]<drop/>

- [st.name/]

[for (oa:Property|st.ownedAttribute)]<drop/>

- [if (not oa.name.contains('base'))][oa.name/]: [if (not cl.getValue(st, oa.name).oclIsUndefined())][if oa.name.contains('condition')][cl.getValue(st, oa.name).oclAsType(String)/] [else][cl.getValue(st, oa.name).oclAsType(EnumerationLiteral).name/][/if][else]<drop/>[/if]

[/if] <drop/>
[/for]<drop/>
[/for]<drop/>
[if cl.ownedAttribute->notEmpty()]<drop/>
<drop/>

Table 6: Attributes for [cl.name/]

<table><drop/>

Attribute Name	Type	Multiplicity	Access	Stereotypes	Description
[for (p:Property|cl.allAttributes())]<drop/>					
[p.name/]	[p.type.name/]	[if(p.lower=p.upper)]1[else][p.lower/]..[if	[if(not(p.isReadOnly)	[for (st:Stereotype | p.getAppliedStereotypes())]<drop/> [st.name/]	[if p.ownedComment->notEmpty()]<drop/> [for (c:Comment | p.ownedComment)] <drop/>

[if(not p.qualifiedName.contains(cl.name))]Inherited[/if]		(p.upper=-1)]*[else][p.upper/][/if][/if]	)]RW[else]R[/if]	[for(oa:Property|st.ownedAttribute)]<drop/> • [if oa.name.contains('key')]partOfObjectKey: [p.getValue(st, oa.name).oclAsType(Integer)/] [else]<drop/> • [if oa.name.contains('attribute')]AVC: [p.getValue(st, oa.name).oclAsType(EnumerationLiteral).name/] [else]<drop/> • [if oa.name.contains('invariant')]isInvariant: [p.getValue(st, oa.name).oclAsType(Boolean)/] [else]<drop/> • [if oa.name.contains('value')]valueRange: [if (not p.getValue(st, oa.name).oclIsUndefined())][p.getValue(st, oa.name).oclAsType(String)/][else] no range constraint [/if] [else]<drop/> • [if oa.name.contains('length')]bitLength: [p.getValue(st, oa.name).oclAsType(EnumerationLiteral).name/] [else]<drop/> • [if oa.name.contains('unsigned')]unsigned: [p.getValue(st, oa.name).oclAsType(Boolean)/] [else]<drop/> • [if oa.name.contains('encoding')]encoding: [p.getValue(st, oa.name).oclAsType(EnumerationLiteral).name/] [else]<drop/> • [if oa.name.contains('counter')]counter: [p.getValue(st, oa.name).oclAsType(EnumerationLiteral).name/] [else]<drop/> • [if oa.name.contains('unit')]unit: [if (not p.getValue(st, oa.name).oclIsUndefined())][p.getValue(st, oa.name).oclAsType(String)/][else] no unit defined [/if] [else]<drop/> • [if oa.name.contains('support')]support: [p.getValue(st, oa.name).oclAsType(EnumerationLiteral).name/] [else]<drop/>	[c._body.clean()/] [/for] [else] [if (p.name.contains ('_'))] See referenced class [else] NO DESCRIPTION [/if]<drop/> [/if]<drop/>

				• [if oa.name.contains('condition')][if (not p.getValue(st, oa.name).oclIsUndefined())]condition:[p.getValue(st, oa.name).oclAsType(String)/][else] <drop/> [/if] [else]<drop/> [/if]<drop/> [/if]<drop/> [/if]<drop/> [/if]<drop/> [/if]<drop/> [/if]<drop/> [/if]<drop/> [/if]<drop/> [/if]<drop/> [/if]<drop/> [/for]<drop/> [/for]<drop/>	

[/for]<drop/>
</table><drop/>
[else][/if]<drop/>
</fragment><drop/>

8.14.1.13 Insert Data Types Fragment (DD only)

<fragment name='insertDataTypes' importedBundles='commons;gmf;papyrus'><drop/>
<arg name='dt' type='uml::DataType'/><drop/>

<Word section heading> [dt.name/]

Qualified Name: [dt.qualifiedName/]

[for (co:Comment | dt.ownedComment)]<drop/>
<dropEmpty>[co._body.clean()/]</dropEmpty>
[/for]<drop/>
Applied stereotypes:
[for (st:Stereotype | dt.getAppliedStereotypes())]<drop/>

- [st.name/]

[/for]<drop/>
[if dt.ownedAttribute->notEmpty()]<drop/>
<drop/>

Table 7: Attributes for [dt.name/]

<table><drop/>

Attribute Name	Type	Multiplicity	Access	Stereotypes	Description
[for (p:Property|dt.ownedAttribute)]<drop/>					
[p.name/]	[p.type.name/]	[if(p.lower=p.upper)]1[else][p.lower/]..[if(p.upper=-1)]*[else][p.upper/][/if][/if]	[if(not(p.isReadOnly))]RW[else]R[/if]	[for (st:Stereotype | p.getAppliedStereotypes())]<drop/> [st.name/] [for(oa:Property|st.ownedAttribute)]<drop/> • [if oa.name.contains('key')]partOfObjectKey: [p.getValue(st, oa.name).oclAsType(Integer)/] [else]<drop/> • [if oa.name.contains('attribute')]AVC: [p.getValue(st, oa.name).oclAsType(EnumerationLiteral).name/] [else]<drop/> • [if oa.name.contains('invariant')]isInvariant: [p.getValue(st, oa.name).oclAsType(Boolean)/] [else]<drop/> • [if oa.name.contains('value')]valueRange: [if (not p.getValue(st, oa.name).oclIsUndefined())][p.getValue(st, oa.name).oclAsType(String)/][else] no range constraint [/if] [else]<drop/> • [if oa.name.contains('length')]bitLength: [p.getValue(st, oa.name).oclAsType(EnumerationLiteral).name/] [else]<drop/> • [if oa.name.contains('unsigned')]unsigned: [p.getValue(st, oa.name).oclAsType(Boolean)/] [else]<drop/> • [if oa.name.contains('encoding')]encoding: [p.getValue(st, oa.name).oclAsType(EnumerationLiteral).name/] [else]<drop/>	[for (c:Comment | p.ownedComment)] <drop/> [c._body.clean()/] [/for]

				• [if oa.name.contains('counter')]counter: [p.getValue(st, oa.name).oclAsType(EnumerationLiteral).name/] [else]<drop/> • [if oa.name.contains('unit')]unit: [if (not p.getValue(st, oa.name).oclIsUndefined())][p.getValue(st, oa.name).oclAsType(String)/][else] no unit defined [/if] [else]<drop/> • [if oa.name.contains('support')]support: [p.getValue(st, oa.name).oclAsType(EnumerationLiteral).name/] [else]<drop/> • [if oa.name.contains('condition')][if (not p.getValue(st, oa.name).oclIsUndefined())]condition:[p.getValue(st, oa.name).oclAsType(String)/][else]<drop/> [/if] [else]<drop/> [/if]<drop/> [/if]<drop/> [/if]<drop/> [/if]<drop/> [/if]<drop/> [/if]<drop/> [/if]<drop/> [/if]<drop/> [/if]<drop/> [/if]<drop/> [/if]<drop/> [/for]<drop/> [/for]<drop/>	

[/for]<drop/>
</table><drop/>
[else][/if]<drop/>
</fragment><drop/>

8.14.1.14 Insert Enums Fragment (DD only)

<fragment name='insertEnums' importedBundles='commons;gmf;papyrus'><drop/>
<arg name='dt' type='uml::DataType'/><drop/>

<Word section heading> [dt.name/]

Qualified Name: [dt.qualifiedName/]

[for (co:Comment | dt.ownedComment)]<drop/>
<dropEmpty>[co._body.clean()/]</dropEmpty>
[/for]<drop/>

Applied stereotypes:
[for (st:Stereotype | dt.getAppliedStereotypes())]<drop/>
- [st.name/]

[/for]<drop/>

Contains Enumeration Literals:
[for (e:EnumerationLiteral|dt.oclAsType(Enumeration).ownedLiteral)]<drop/>
- [e.name/]:
  - [for (co:Comment | e.ownedComment)]<drop/>
  - <dropEmpty>[co._body.clean()/]
  - </dropEmpty>[/for]<drop/>
  - Applied stereotypes:
    - [for (st:Stereotype | e.getAppliedStereotypes())]<drop/>
    - [st.name/]
    - [/for]<drop/>

[/for]<drop/>
</fragment><drop/>

8.14.1.15 Insert Primitive Ttypes Fragment (DD only)

<fragment name='insertPrimitiveTypes' importedBundles='commons;gmf;papyrus'><drop/>
<arg name='dt' type='uml::DataType'/><drop/>

<Word section heading> [dt.name/]

Qualified Name: [dt.qualifiedName/]

[for (co:Comment | dt.ownedComment)]<drop/>
<dropEmpty>[co._body.clean()/]</dropEmpty>
[/for]<drop/>

Applied stereotypes:
[for (st:Stereotype | dt.getAppliedStereotypes())]<drop/>

- [st.name/]

[/for]<drop/>
</fragment><drop/>

8.14.2 Examples using Gendoc Fragments

8.14.2.1 Introduction

This clause provides examples of usage of the Gendoc fragments provided in the previous clause.

8.14.2.2 Use of Insert Standard Diagram Fragment

[for(p:Package|Package.allInstances())]<drop/>
Inserts the diagram identified in first quotes with the title identified in second quotes <drop/>
[p.insertStandardDiagram(‘<diagram name in Papyrus>’, ’<diagram title in Word>’)/]
[/for]<drop/>

8.14.2.3 Use of Insert Small Diagram Fragment

[for(p:Package|Package.allInstances())]<drop/>
Inserts the diagram identified in first quotes with the title identified in second quotes <drop/>
[p.insertSmallDiagram(‘<diagram name in Papyrus>’, ’<diagram title in Word>’)/]
[/for]<drop/>

8.14.2.4 Simple use of Insert Class Fragment for specific class list

[for (cl:Class | Class.allInstances()->sortedBy(name))]<drop/>
[if (cl.qualifiedName.contains(‘<package name>’))]<drop/>
[if (cl.name.contains(‘<class 1 name>’) or cl.name.contains(‘<class 2 name>’))]<drop/>

<Word section heading> [cl.name/]

Inserts the details (qualified name and applied comment) of the class in first quotes from the package in second quotes <drop/>
[cl.insertClass(cl.name,’<package name>’)/]
[/if]<drop/>
[/if]<drop/>
[/for]<drop/>

8.14.2.5 Simple use of Insert Attribute Table Brief Fragment for specific class list

[for (cl:Class | Class.allInstances()->sortedBy(name))]<drop/>
[if (cl.qualifiedName.contains(‘<package name>’))]<drop/>
[if (cl.name.contains(‘<class 1 name>’) or cl.name.contains(‘<class 2 name>’))]<drop/>

<Word section heading> [cl.name/]

Inserts the attributes (name, lifecycle state, applied comment) of the class <drop/>
[cl.insertAttributeTableBrief ()/]
[/if]<drop/>
[/if]<drop/>
[/for]<drop/>

8.14.2.6 Simple use of Insert Ten Specified Attribute Table Brief Fragment for specific class list

Will generate a table per class listed and will insert in the table that attributes listed up to 10
<drop/>
[for (cl:Class | Class.allInstances()->sortedBy(name))]<drop/>
[if (cl.qualifiedName.contains('<package name>'))]<drop/>
[if (cl.name.contains('<class name>'))]<drop/>

<Word section heading> **[cl.name/]**

Inserts the attributes (name, lifecycle state, applied comment) of the class identified in the list
<drop/>
[cl.insertTenSpecifiedAttributeTableBrief ('<attribute 1 name>', '<attribute 2 name>', '3', '4', '5', '6', '7', '8', '9', '10')/]
[/if]<drop/>
[/if]<drop/>
[/for]<drop/>

8.14.2.7 Use of Insert Start Attribute Table and Insert Attribute Row Fragments for specific class list (previous method is preferred)

[for (cl:Class | Class.allInstances()->sortedBy(name))]<drop/>
[if (cl.qualifiedName.contains('<package name>'))]<drop/>
[if (cl.name.contains('<class 1 name>') or cl.name.contains('<class 2 name>'))]<drop/>

<Word section heading> **[cl.name/]**

Inserts the identified attributes (name, lifecycle state, applied comment) of the identified classes
<drop/>
[if cl.ownedAttribute->notEmpty()]<drop/>

Table 8: Attributes for [cl.name/]

<table><drop/>
[cl.insertAttributeTableHeader ()/]
[for (p:Property|cl.ownedAttribute)]<drop/>
[if (p.name.contains('<attribute 1 name>') or p.name.contains('<attribute 2 name>'))]<drop/>
[p.insertAttributeRowBrief ()/]
[/if]
[/for]<drop/>
</table><drop/>
[/if]<drop/>
[/if]<drop/>
[/if]<drop/>
[/for]<drop/>

8.14.2.8 Intertwining of Insert Class Fragment and Insert Attribute Table Brief Fragment for a specific class

[for (cl:Class | Class.allInstances()->sortedBy(name))]<drop/>
[if (cl.qualifiedName.contains('<package name>'))]<drop/>

[if (cl.name.contains('<class 1 name>'))]<drop/>

<Word section heading> [cl.name/]

Inserts the details of the class (qualified name, applied comment) in first quotes from the package in second quotes <drop/>
[cl.insertClass(cl.name,'<package name>')/]
Inserts the attributes (name, lifecycle state, applied comment) of the class <drop/>
[cl.insertAttributeTableBrief ()/]
[/if]<drop/>
[/if]<drop/>
[/for]<drop/>

8.14.2.9 Intertwining of Insert Class Fragment and Insert Attribute Table Brief Fragment for a list of specific classes with fragments of text after each table.

[for (cl:Class | Class.allInstances()->sortedBy(name))]<drop/>
[if (cl.qualifiedName.contains('CoreNetworkModel'))]<drop/>
[if (cl.name.contains('FcSwitch') or cl.name.contains('FcRoute'))]<drop/>

<Word section heading> [cl.name/]

Inserts the details of the class (qualified name, applied comment) in first quotes from the package in second quotes <drop/>
[cl.insertClass(cl.name,'<package name>')/]
Inserts the attributes (name, lifecycle state, applied comment) of the class <drop/>
[cl.insertAttributeTableBrief ()/]
[if (cl.name.contains('<class 1 name>'))]<drop/>
<Text 1 to be inserted after the table>
[else] <drop/>
[if (cl.name.contains('<class 2 name>'))]<drop/>
<Text 2 to be inserted after the table>
[/if]<drop/>
[/if]<drop/>
[/if]<drop/>
[/if]<drop/>
[/for]<drop/>
End of document

</gendoc><drop/>

8.15 Example complete template

The following embedded file contains script which when modified to select the right model (several places in the file) and output locations will print three diagrams and/or all diagrams in the model (follow instruction to control the options) and will then provide data dictionary clauses listing classes, complex data types, enumerations and primitive data types.


gdFullModelDocumentationExample.docx

Note that when using a template figure numbers, table numbers, references, tables of contents/figures/tables etc will need to be updated in the output document to provide a completed document.

8.16 Extending the template

Work is ongoing to extend the capability for documentation. These guidelines will be updated as additional relevant capabilities are identified.

8.17 Known issues

Known issues:

- Some header/footer content may prevent document production
- "[" (often used in references) shall not be used in the Word text anywhere
- The "importedFragments" statement does not allow a space after the semicolon; e.g., importedFragments='<name of first fragment>; <name of second fragment>

9 Importing RSA Models into Papyrus

9.1 Prerequisite

This clause describes the steps to be followed when Models "written" in RSA (TM Forum and ITU-T are using this UML tool from IBM) need to be imported to Papyrus.

Prerequisite for doing this is that the additional Papyrus component "RSA Model Importer" is installed. Additional Papyrus components can be installed via

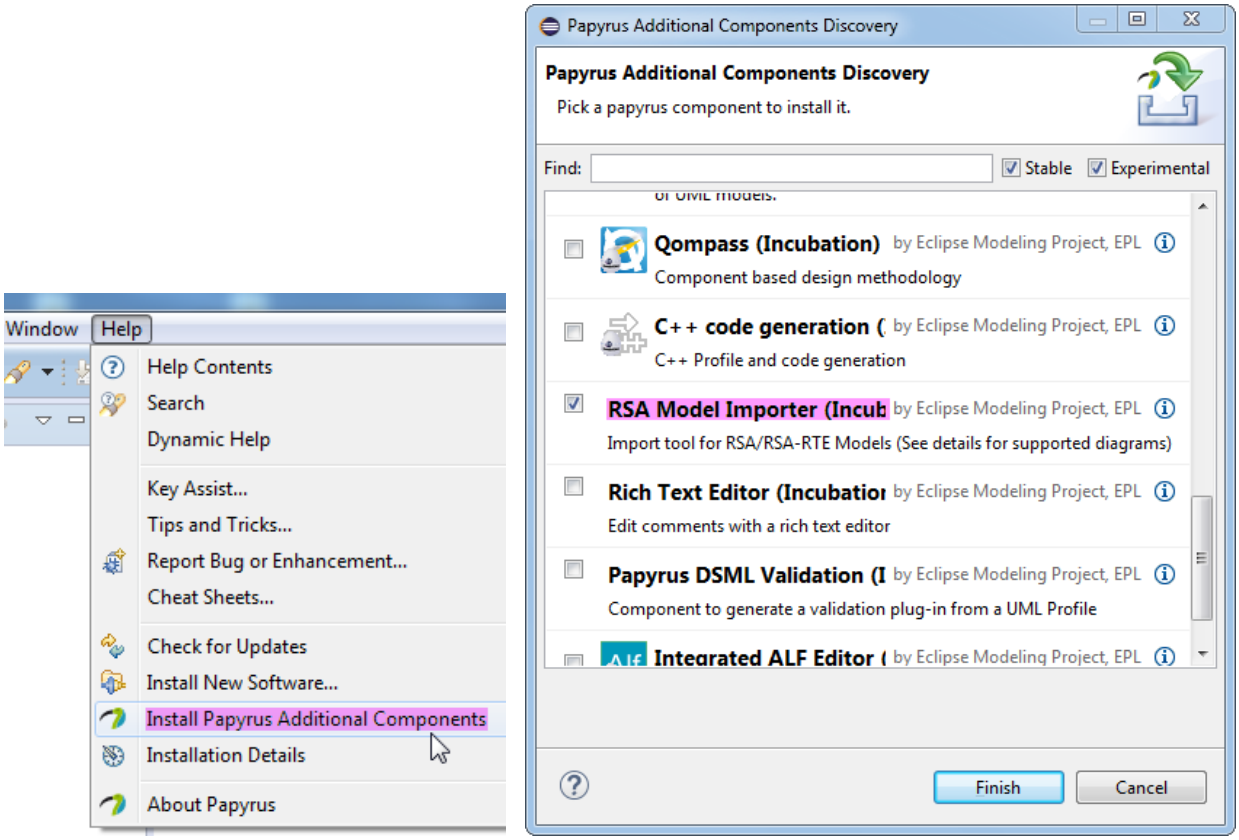
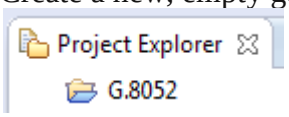


Figure 9-1: Installing Papyrus Component “RSA Model Importer”

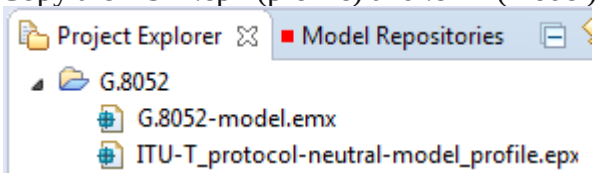
9.2 Import RSA Model into Papyrus

Notes: Each step identifies the tool that is used to execute it. Any ASCII editor can be used instead of Notepad++.
The import of the ITU-T G.8052 model is used here as an example.

1. Create a new, empty general project (i.e., not a Papyrus project).



2. Copy the RSA .epx (profile) and .emx (model) files into the empty project folder.



3. Import the RSA model (.emx file) into Papyrus by right-click on the .emx file and then select “Import EMX model”:

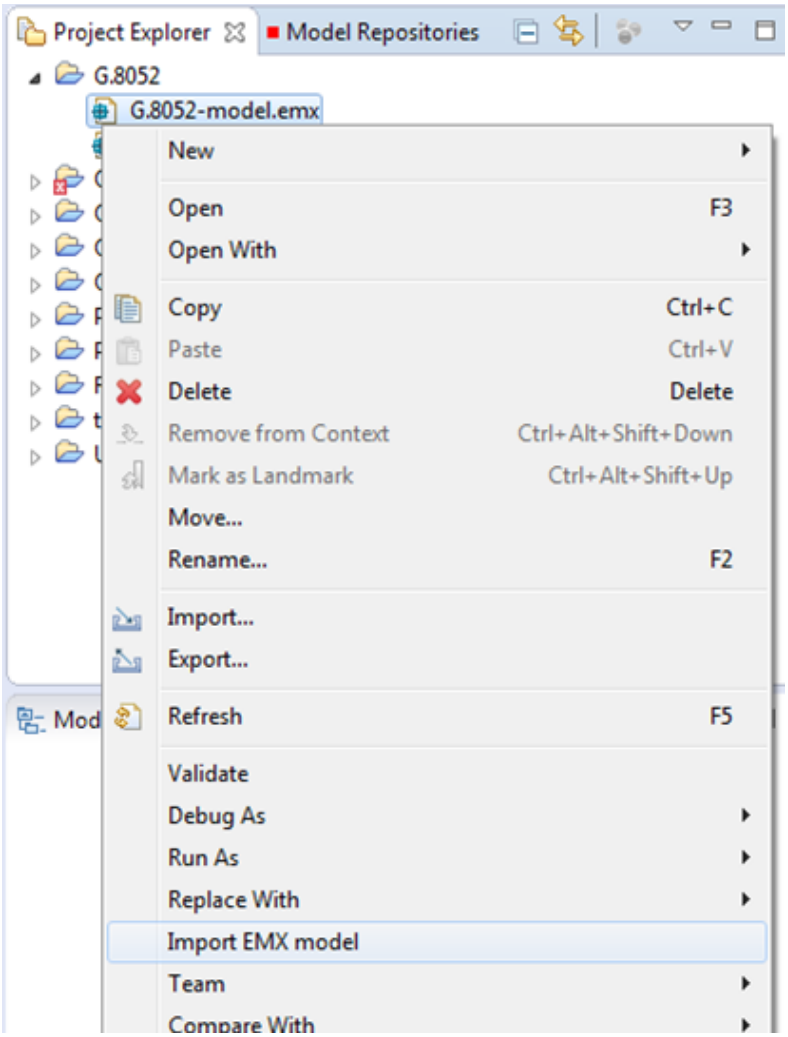
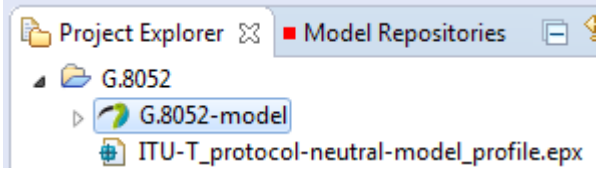


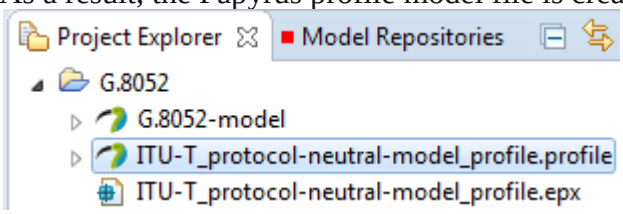
Figure 9-2: Importing .emx Model

As a result, the RSA .emx file is replaced by the Papyrus model file:



4. Import the RSA Profile model (.epx file) into Papyrus by right-click on the .epx file and then select “Import EMX model” (same as previous step).

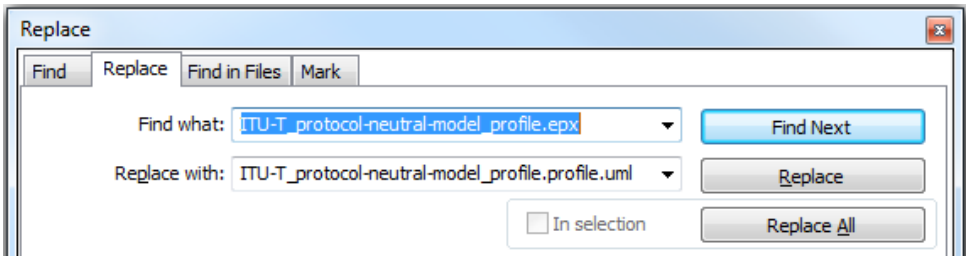
As a result, the Papyrus profile model file is created:



9.3 Replace RSA Profile by Papyrus Profile

This is done by changing the pointer from the .epx (RSA) file to the .uml (Payprus) file.

5. Close Papyrus.
   Notepad ++: Replace all occurrences of "ITU-T_protocol-neutral-model_profile.epx" by "ITU-T_protocol-neutral-model_profile.profile.uml" in the model .uml file
   (G.8052-model.uml) in the Papyrus Workspace:



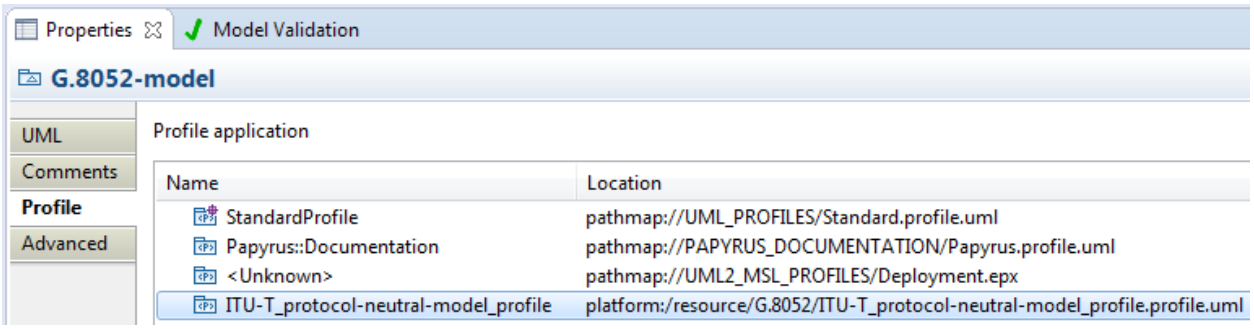


Figure 9-3: Associated Papyrus Profile

9.4 Remove the "old" RSA files

6. Windows Explorer: Delete the RSA profile .epx file
   (ITU-T_protocol-neutral-model_profile.epx) from the Papyrus Workspace.
7. Windows Explorer: Delete the RSA model .emx file (G.8052-model.emx) from the Papyrus Workspace.

10 Main Changes between Releases

10.1 Summary of main changes between version 1.0 and 1.1

The following guidelines have been changed/added:

- Eclipse: Migration from Kepler to Mars
- Github: Only one develop branch
- Github: Method for retrieving repositories using the "Download ZIP" button added; this method is just for read only users
- Documentation: New clause on Gendoc added.

10.2 Summary of main changes between version 1.1 and 1.2

- ONF specific text generalized
- Document moved to Open Source SDN
- GitHub work flow for OpenModelProfile added in clause 6.2.
- Description of extracting data from the model into Excel tables deleted.
- Use of Gendoc “fragments” added in clause 8.14.
- Constructing of the Modeling Environment for a new Model described in clause 7.4.

10.3 Summary of main changes between version 1.2 and 1.3

- Adapted to ETSI drafting rules.
- Updated for Papyrus 3.x.x (Oxygen).
- Model infrastructure files (e.g., profiles, style sheet) moved into the Papyrus project.
- Section on “Constructing Modeling Environment for a new Model” completely reworked.
- Installation of specific Eclipse version 4.7.2 “Oxygen” and Papyrus version 3.2.0 RC4 added and warning added to prevent automatic updates.
- Justification for UML package structure added in section 7.4.5.
- Papyrus installation in section 5.2.2 changed to use the update site.