

OPEN NETWORKING
FOUNDATION

Papyrus Guidelines

Version 1.1
November 30, 2015

ONF TR-515



ONF Document Type: Technical Recommendation

ONF Document Name: Papyrus Guidelines V1.1

Disclaimer

THIS SPECIFICATION IS PROVIDED “AS IS” WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.

Any marks and brands contained herein are the property of their respective owners.

Open Networking Foundation
2275 E. Bayshore Road, Suite 103, Palo Alto, CA 94303
www.opennetworking.org

©2015 Open Networking Foundation. All rights reserved.

Open Networking Foundation, the ONF symbol, and OpenFlow are registered trademarks of the Open Networking Foundation, in the United States and/or in other countries. All other brands, products, or service names are or may be trademarks or service marks of, and are used to identify, products or services of their respective owners.

Content

| | | |
|----------------|---|-----------|
| 1 | <i>Introduction</i> | 6 |
| 2 | <i>References</i> | 6 |
| 3 | <i>Abbreviations</i> | 6 |
| 4 | <i>Documentation Overview</i> | 7 |
| 5 | <i>Getting Papyrus Running</i> | 8 |
| 5.1 | Downloading Eclipse | 9 |
| 5.2 | Installing Papyrus | 10 |
| 5.3 | Importing a Model | 15 |
| 5.4 | Deleting a Project | 19 |
| 6 | <i>Information Model on GitHub</i> | 20 |
| 6.1 | ONInfoModel Structure on GitHub | 20 |
| 6.2 | GitHub Work Flow | 21 |
| 6.3 | Downloading a Model from github for “Read Only Use” | 37 |
| 7 | <i>Using Papyrus</i> | 39 |
| 7.1 | Illustrative Profile and Model | 39 |
| 7.2 | Papyrus File Structure | 40 |
| 7.3 | Model Splitting | 41 |
| 7.4 | Team UML Model Development | 42 |
| 7.5 | Developing a Sub-Model | 45 |
| 8 | <i>Extracting Data from a Papyrus model</i> | 47 |
| 8.1 | Gendoc Plugin | 48 |
| 8.2 | Installing Gendoc | 48 |
| 8.3 | Using Gendoc | 50 |
| 8.4 | Papyrus Table | 50 |
| 9 | <i>Importing RSA Models into Papyrus</i> | 54 |
| 9.1 | Import RSA Model into Papyrus | 54 |
| 9.2 | Replace RSA Profile by Papyrus Profile | 56 |
| 9.3 | Remove the “old” RSA files | 57 |
| Annex A | <i>Using Gendoc</i> | 58 |
| A.1 | Template usage | 58 |
| A.2 | Basic template | 59 |
| A.3 | Cover, contents, closing text etc | 60 |

| | | |
|-------------|---|-----------|
| A.4 | Figures from the model with interleaved text | 60 |
| A.5 | Figure in alphabetical order with no interleaved specific text | 62 |
| A.6 | Further explanation of the script | 62 |
| A.7 | Test template for printing diagrams and associated text | 62 |
| A.8 | Data Dictionary template overview | 63 |
| A.9 | Adding the class and its stereotypes | 63 |
| A.10 | Adding properties and stereotypes in tabular form | 65 |
| A.11 | Adding complex data types | 67 |
| A.12 | Adding other data types | 67 |
| A.12.1 | Enumeration Types | 67 |
| A.12.2 | Primitive Types | 68 |
| A.13 | Example complete template | 69 |
| A.14 | Extending the template | 69 |
| A.15 | Known issues | 69 |

List of Figures

| | |
|---|----|
| Figure 4-1: Specification Architecture | 9 |
| Figure 5-1: Eclipse Download Page | 10 |
| Figure 5-2: Content of the Eclipse Folder after Extracting the Zip-file | 10 |
| Figure 5-3: Initial Welcome Page of Eclipse | 11 |
| Figure 5-4: Installing Papyrus (1) | 12 |
| Figure 5-5: Installing Papyrus (2) | 12 |
| Figure 5-6: Installing Papyrus (3) | 13 |
| Figure 5-7: Proxy Configuration | 14 |
| Figure 5-8: Open Papyrus Perspective | 15 |
| Figure 5-9: Required Display Setting | 16 |
| Figure 5-10: Papyrus Project Explorer / Model Explorer | 16 |
| Figure 5-11: Papyrus Model Structure | 17 |
| Figure 5-12: Importing a Model (1) | 18 |
| Figure 5-13: Importing a Model (2) | 19 |
| Figure 5-14: Open a Model | 20 |
| Figure 5-15: Delete a Project | 21 |
| Figure 6-1: Initial ONFInfoModel Structure on GitHub | 22 |
| Figure 6-2: GitHub Work Flow | 23 |
| Figure 6-3: Open Git Perspective | 24 |

| | |
|---|----|
| Figure 6-4: Add Repository Choices | 24 |
| Figure 6-5: Location of the Repository Address..... | 25 |
| Figure 6-6: Source Git Repository Window..... | 26 |
| Figure 6-7: Branch Selection Window..... | 26 |
| Figure 6-8: Local Destination Window | 27 |
| Figure 6-9: develop Branch Cloned to Local PC..... | 28 |
| Figure 6-10: develop branch shown in Papyrus Project Explorer (snapshot)..... | 28 |
| Figure 6-11: NbiTopologyModule Shown in Papyrus Model Explorer | 29 |
| Figure 6-12: Importing UML Primitive Types | 29 |
| Figure 6-13: Importing Core Model Artifacts | 30 |
| Figure 6-14: Selecting Core Model Artifacts | 31 |
| Figure 6-15: Imported Core Model Artifacts..... | 32 |
| Figure 6-16: Unstaged Changes in Git Staging | 33 |
| Figure 6-17: Add Files to Git Stage..... | 33 |
| Figure 6-18: Staged Changes in Git Staging | 33 |
| Figure 6-19: Push Updated Branch to Remote Repository | 34 |
| Figure 6-20: Push Confirmation Window | 35 |
| Figure 6-21: Compare in Modeler's Remote Repository | 36 |
| Figure 6-22: Detailed Comparison in Modeler's Remote Repository..... | 36 |
| Figure 6-23: Pull Request in Modeler's Remote Repository..... | 37 |
| Figure 6-24: Pull Request in Administrator's Remote Repository..... | 37 |
| Figure 6-25: Download ONFInfoModel Repository..... | 38 |
| Figure 6-26: Extract ONFInfoModel Repository to Worksoace..... | 38 |
| Figure 6-27: Making the ONFInfoModel visible in Papyrus (1) | 39 |
| Figure 6-28: Making the ONFInfoModel visible in Papyrus (2) | 39 |
| Figure 7-1: Illustrative UML Profile..... | 40 |
| Figure 7-2: Illustrative Core Model..... | 41 |
| Figure 7-3: Profile Associated to the Model | 41 |
| Figure 7-4: Papyrus File Structure | 41 |
| Figure 7-5: Papyrus File Structure after Splitting | 42 |
| Figure 7-6: Imported UML Artifacts..... | 43 |
| Figure 7-7: Information Model File Structure | 44 |
| Figure 7-8: Importing an Existing Project into Papyrus..... | 44 |

| | |
|--|----|
| Figure 7-9: Project and Model Explorer View after Import into Papyrus..... | 45 |
| Figure 7-10: Modeling Process over Time..... | 46 |
| Figure 7-11: Example Sub-Model A (highlighted in red)..... | 47 |
| Figure 7-12: Updated Sub-Model A Files (highlighted in blue)..... | 48 |
| Figure 8-1: Installing Gendoc (1) | 49 |
| Figure 8-2: Installing Gendoc (2) | 50 |
| Figure 8-3: Installing Gendoc (3) | 51 |
| Figure 8-4: Model Selection | 51 |
| Figure 8-5: Class Expansion..... | 52 |
| Figure 8-6: Create new empty Table | 53 |
| Figure 8-7: Artifact Selection..... | 53 |
| Figure 8-8: Creation of Excel Sheet (1) | 54 |
| Figure 8-9: Creation of Excel Sheet (2) | 54 |
| Figure 9-1: Installing Papyrus Component “RSA Model Importer” | 55 |
| Figure 9-2: : Importing .emx Model..... | 56 |
| Figure 9-3: Associated Papyrus Profile..... | 57 |

List of Tables

None.

Document History

| Version | Date | Description of Change |
|---------|----------------|-----------------------|
| 1.0 | March 13, 2015 | Initial version |
| 1.1 | Nov. 30, 2015 | Version 1.1 |

1 Introduction

This Technical Recommendation 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 technology or management protocol. Although the examples used in the document are often ONF related, they can also be used by all other SDOs using Papyrus as their UML tool.

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 section on Gendoc added.

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] ONF TR-514 “UML Modeling Guidelines 1.0” (https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/UML_Modeling_Guidelines_V1.0.pdf)
- [5] Open Model Profile (<https://github.com/OpenNetworkingFoundation/ONFInfoModel>)

3 Abbreviations

| | |
|-------|--|
| API | Application-Programming-Interface |
| ARO | Association Resources Online™ |
| ASCII | American Standard Code for Information Interchange |
| DS | Data Schema |
| IDE | Integrated Development Environment |
| 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 series of Technical Recommendations. The location of this document within the documentation architecture is shown in Figure 4.1 below:

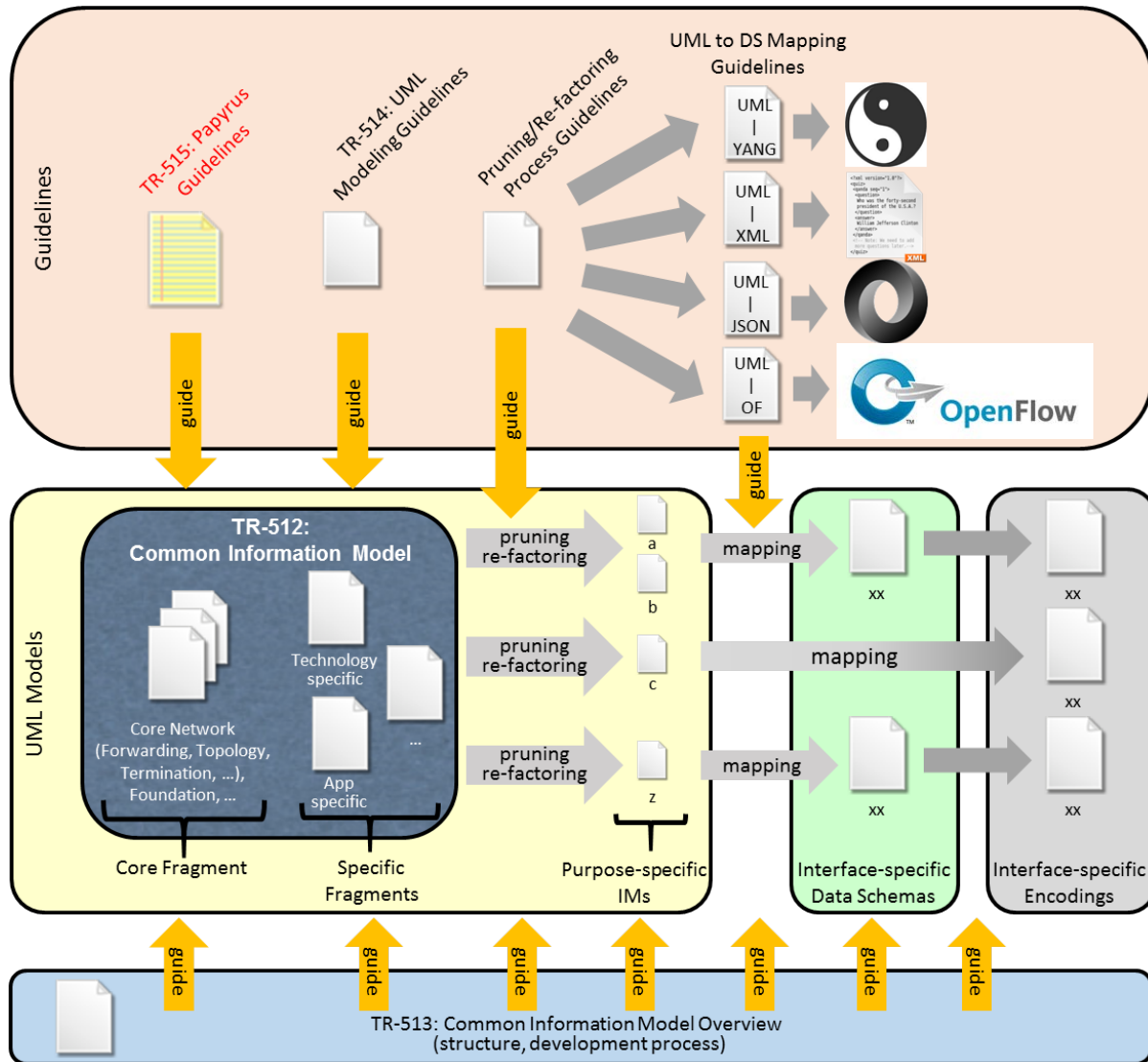


Figure 4-1: Specification Architecture

5 Getting Papyrus Running

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

Current tool versions:

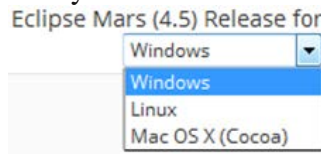
- Eclipse version 4.5.x “Mars” (4.5.1)
- Papyrus version 1.1.x (1.1.1).

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

5.1 Downloading Eclipse

Eclipse can be downloaded from here: <https://eclipse.org/downloads/>.

First you have to choose on which platform you want to run eclipse:



Eclipse offers pre-assembled packages for various use cases. For Papyrus you need the *Eclipse Modeling Tools* package.

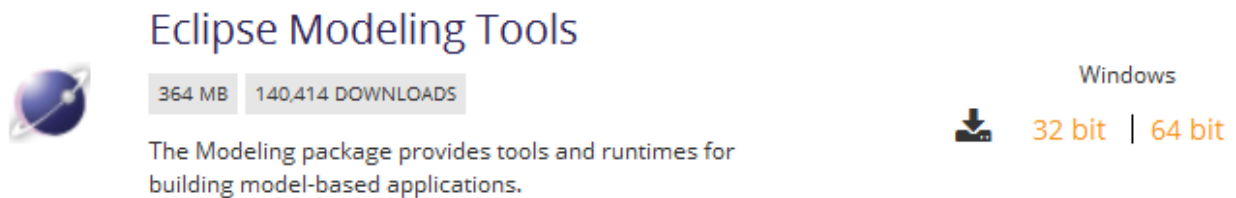


Figure 5-1: Eclipse Download Page

You cannot “install” Eclipse on the PC; just extract the zip-file into a new folder:

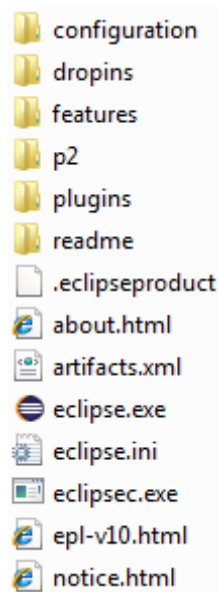
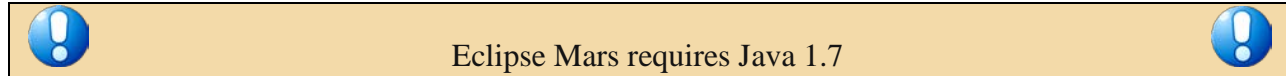


Figure 5-2: Content of the Eclipse Folder after Extracting the Zip-file

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



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

workspace
 .metadata . 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 start of Eclipse.

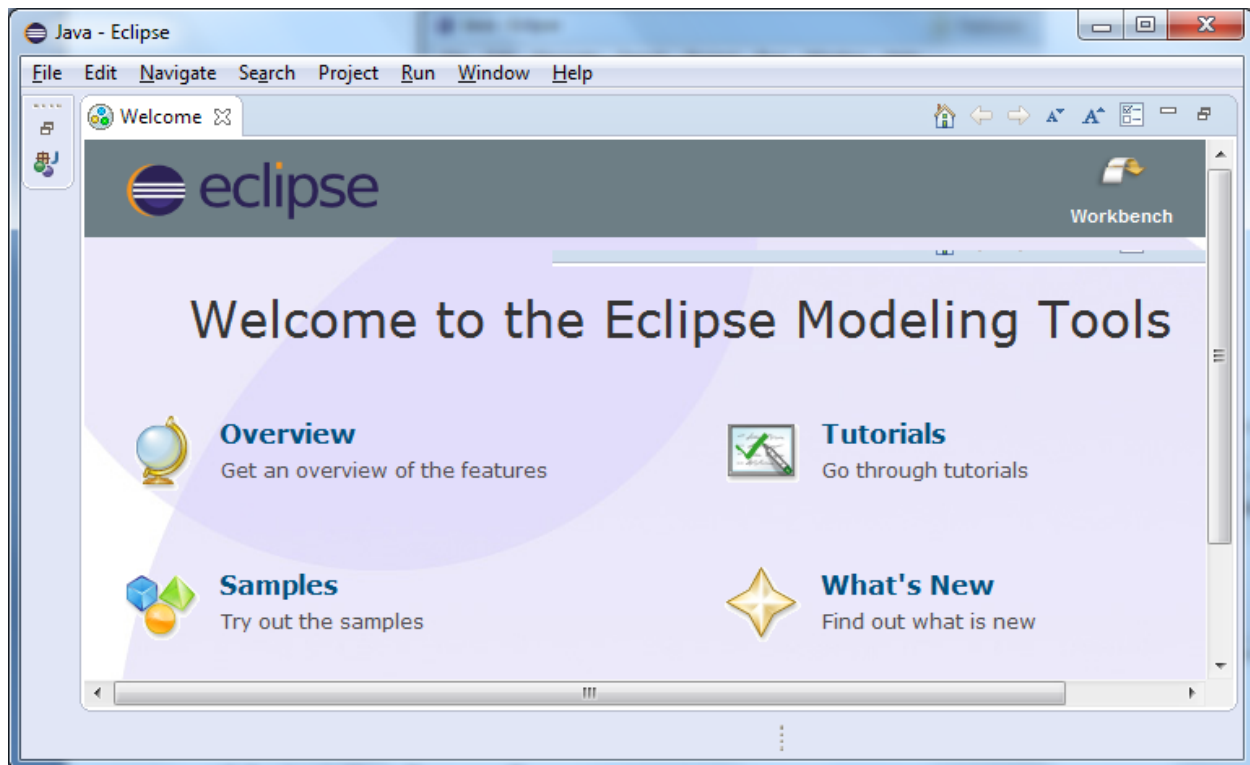


Figure 5-3: Initial Welcome Page of Eclipse

Close the Welcome tab at the upper left corner. Eclipse is now ready for use.

5.2 Installing Papyrus

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

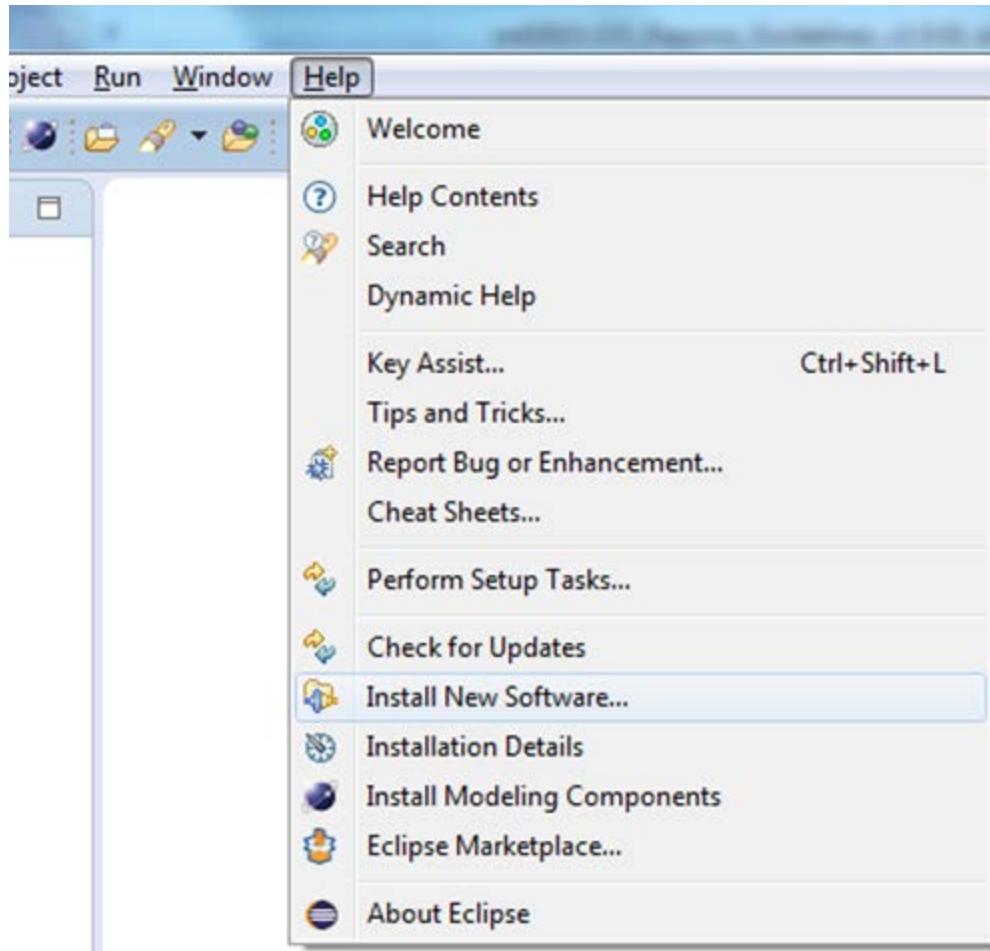


Figure 5-4: Installing Papyrus (1)

Click and enter the Papyrus 1.1 update site:

<http://download.eclipse.org/modeling/mdt/papyrus/updates/releases/mars>

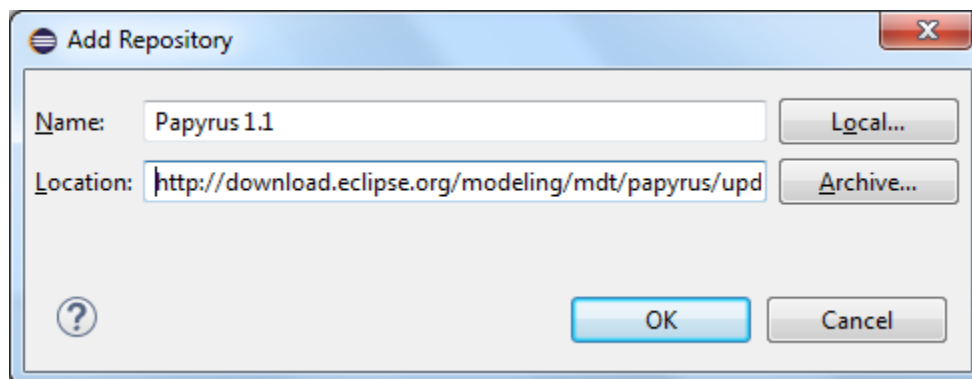


Figure 5-5: Installing Papyrus (2)

You need to select at least Papyrus:

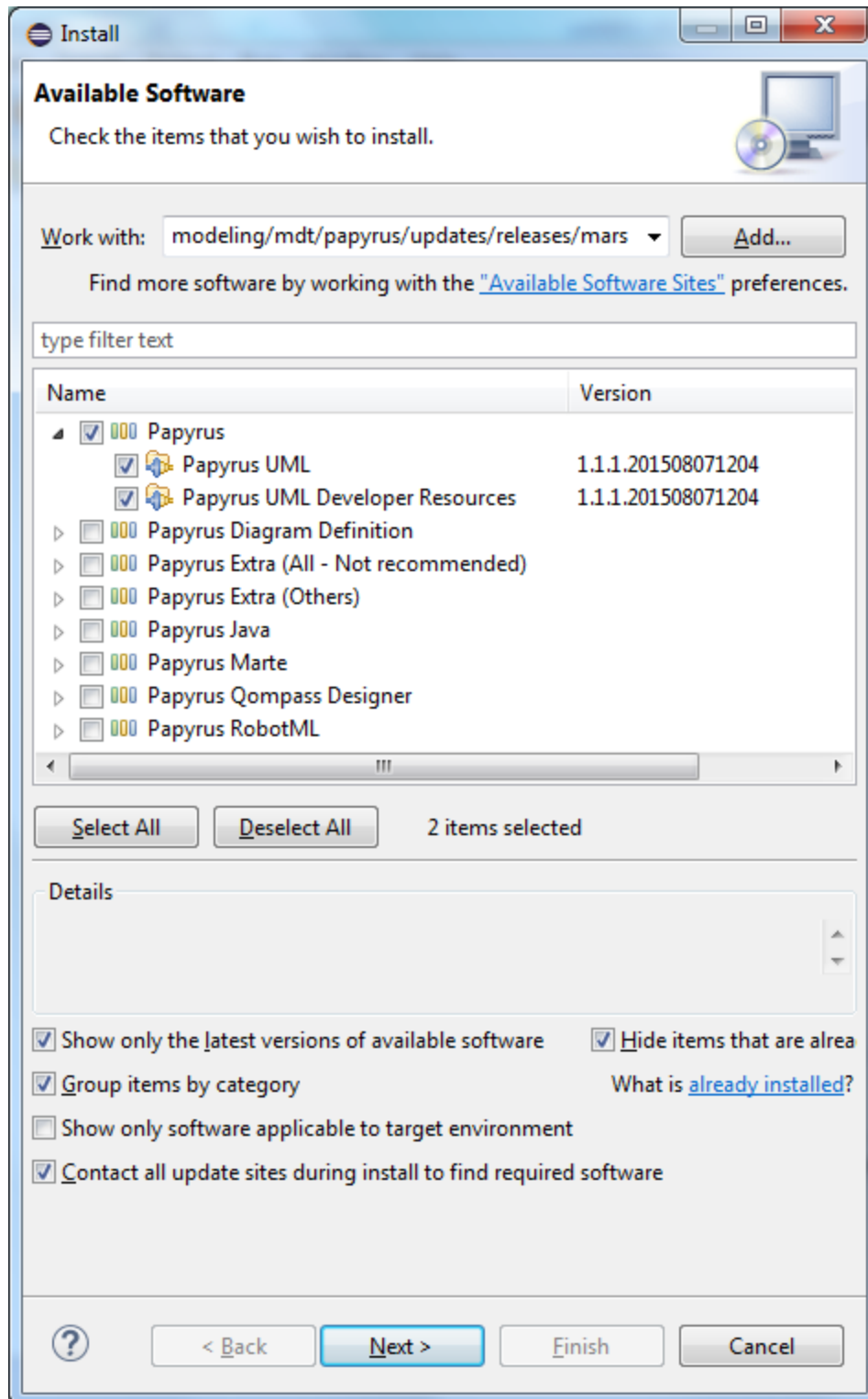


Figure 5-6: Installing Papyrus (3)

Then click **Next >** and follow the instructions.

Only if necessary (usually it is not), you can configure a proxy at menu **Window**, **Preferences** :

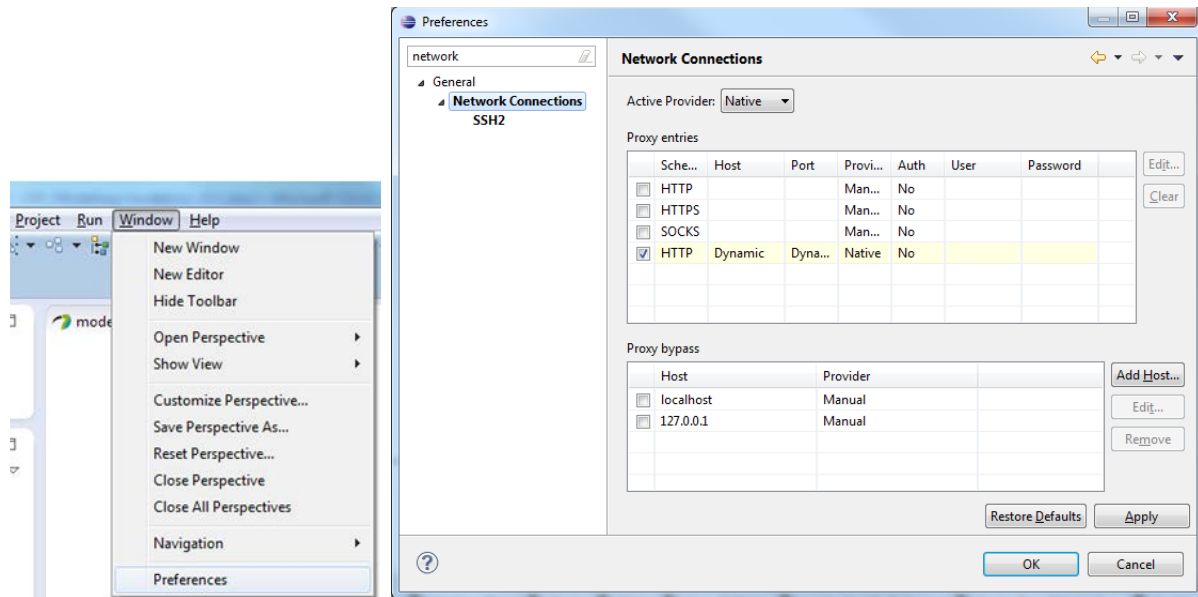
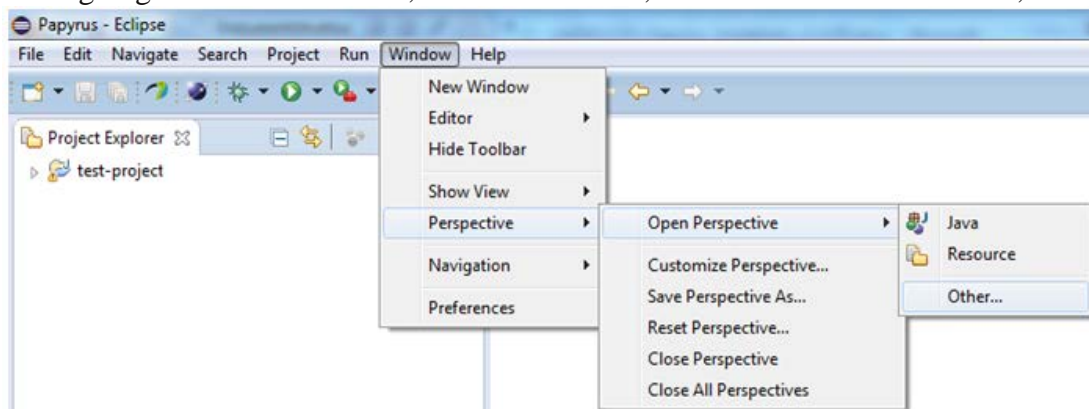


Figure 5-7: Proxy Configuration


After restarting Eclipse you need to switch to the **Papyrus Perspective** 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 :

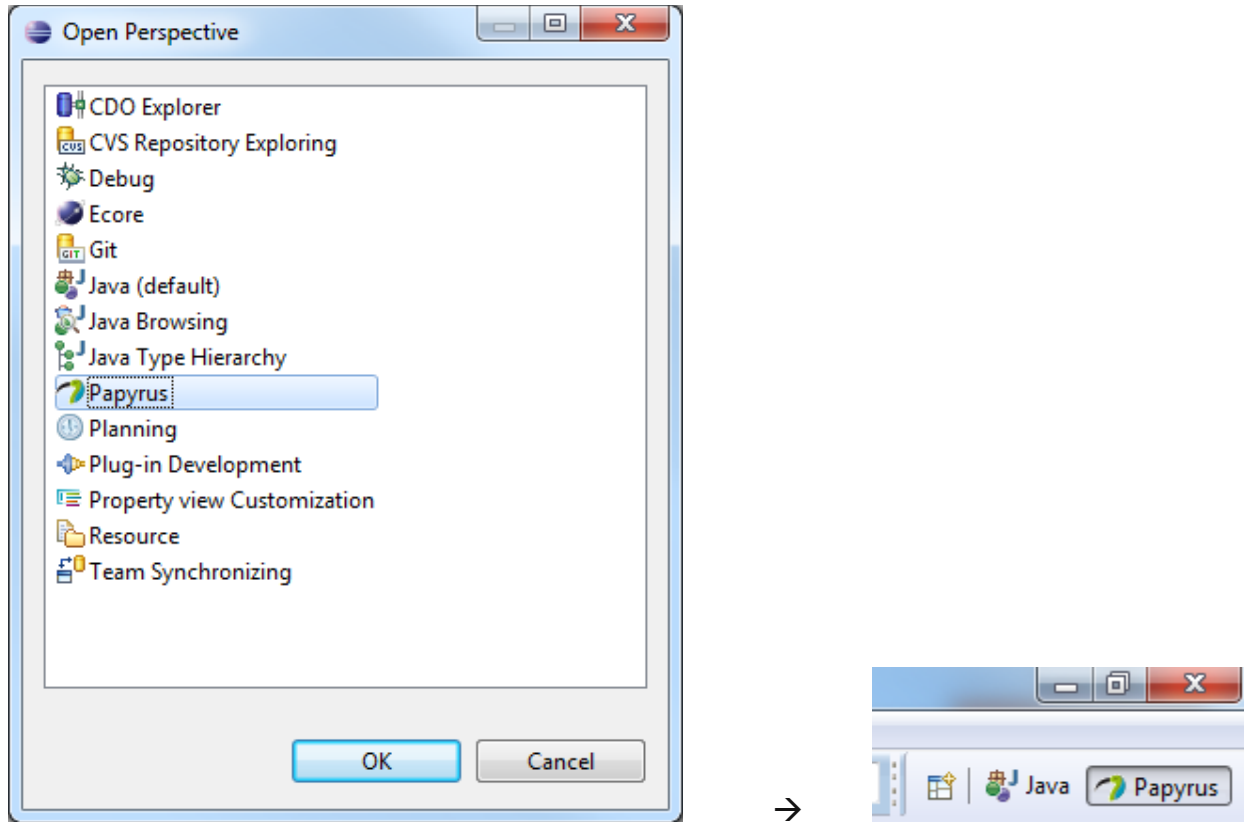




Figure 5-8: Open Papyrus Perspective

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

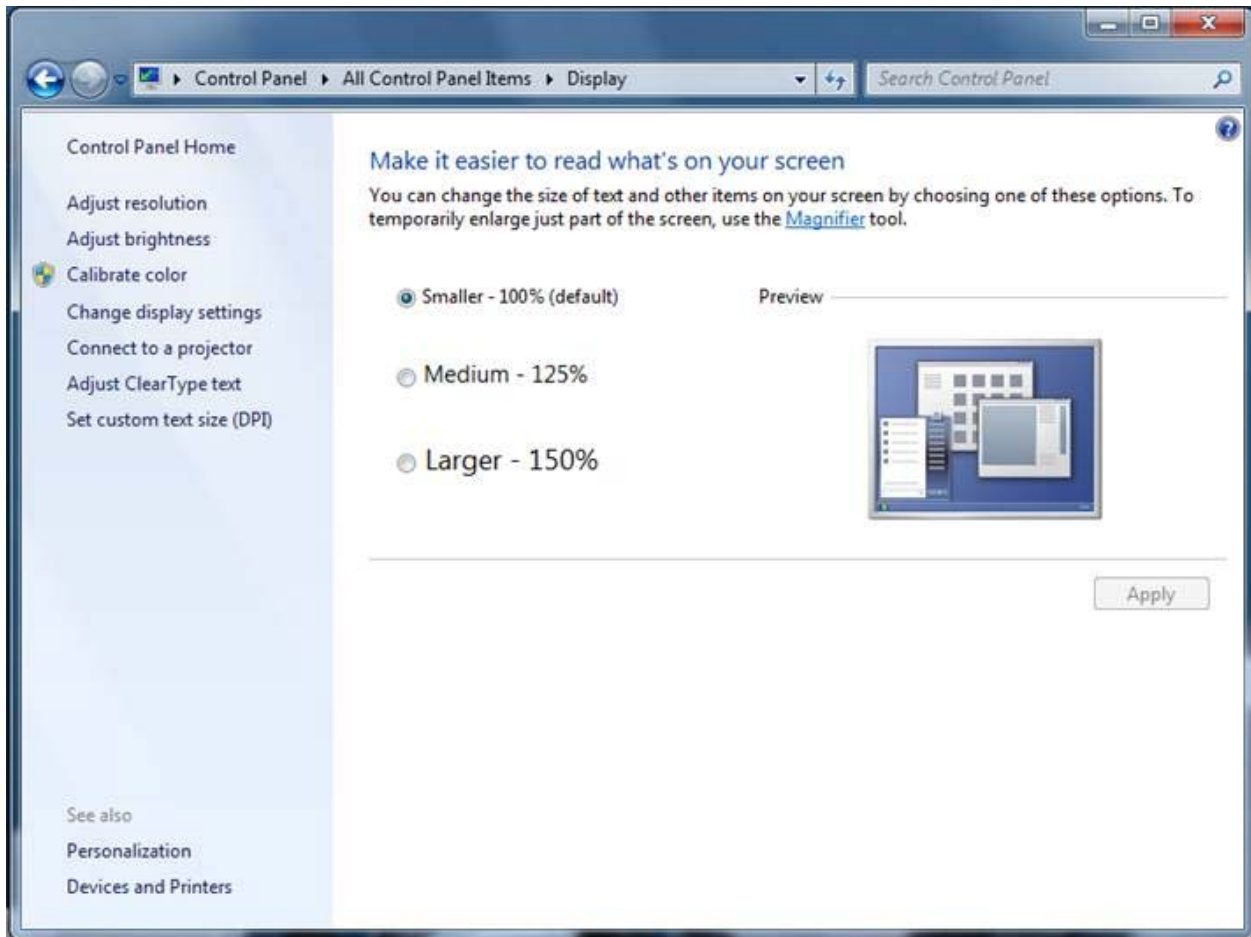


Figure 5-9: Required Display Setting

5.3 Importing a Model

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

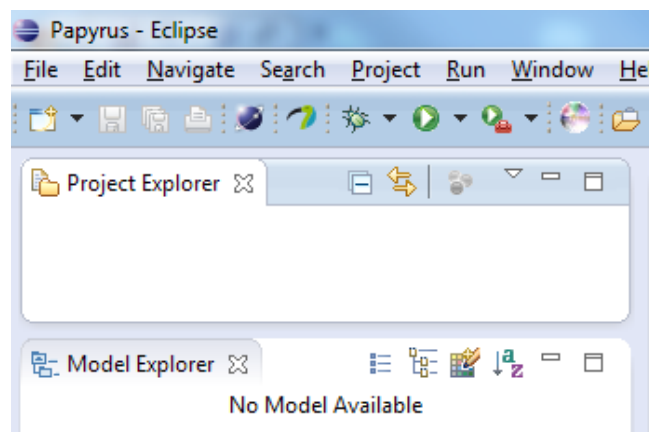








Figure 5-10: Papyrus Project Explorer / Model Explorer

Notes:

Models cannot exist on their own. Every model needs to be contained in a project. A project can contain 0 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 show (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).

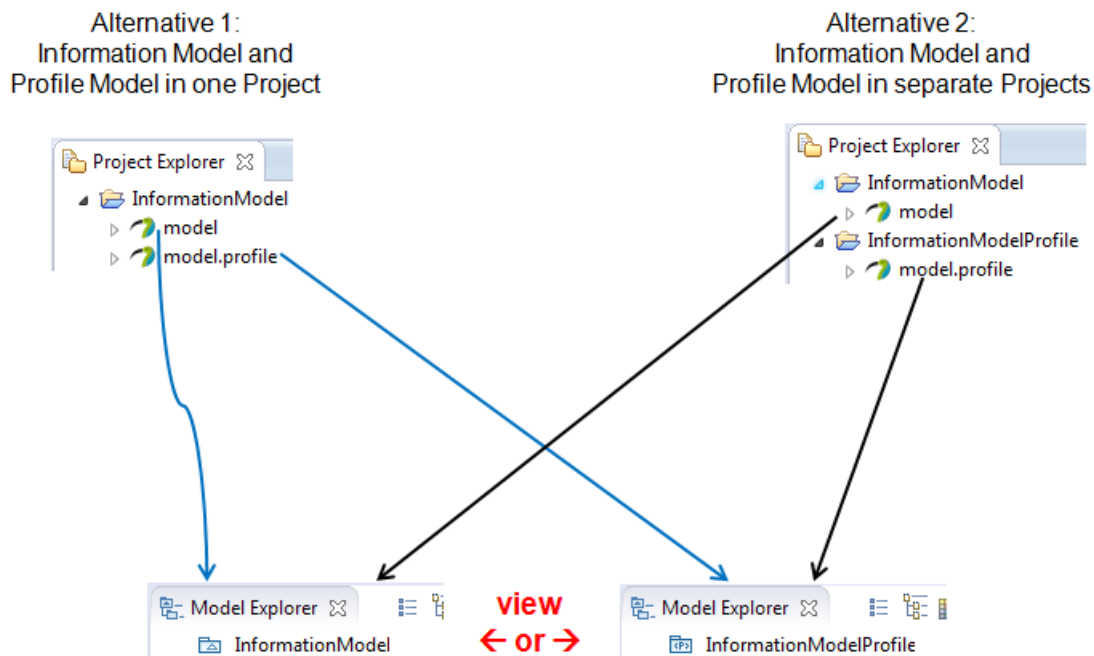




Figure 5-11: Papyrus Model Structure

Note:

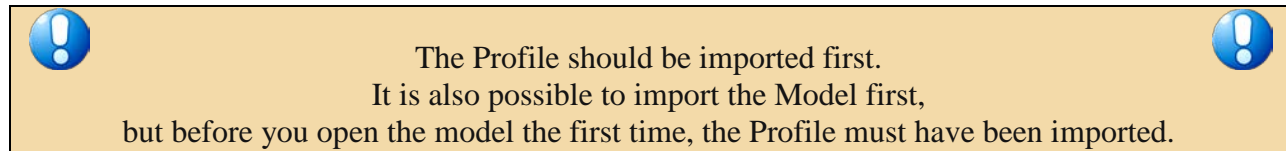
The following model import description uses the ONF Model as an example. The steps are similar for any other model.

The current version of the ONF Model ([CoreInformationModel.V1.0.zip](#); the latest revisions are available on ARO) consists of an Information Model folder and a separate Profile Model folder (i.e., following *Alternative 2* in Figure 5.11):

-  OnfModel
-  OnfProfiles

Each folder contains a **.project**, **.di**, **.notation** and **.uml** files.

The next step is to import the Profiles files and Model files into Papyrus.



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

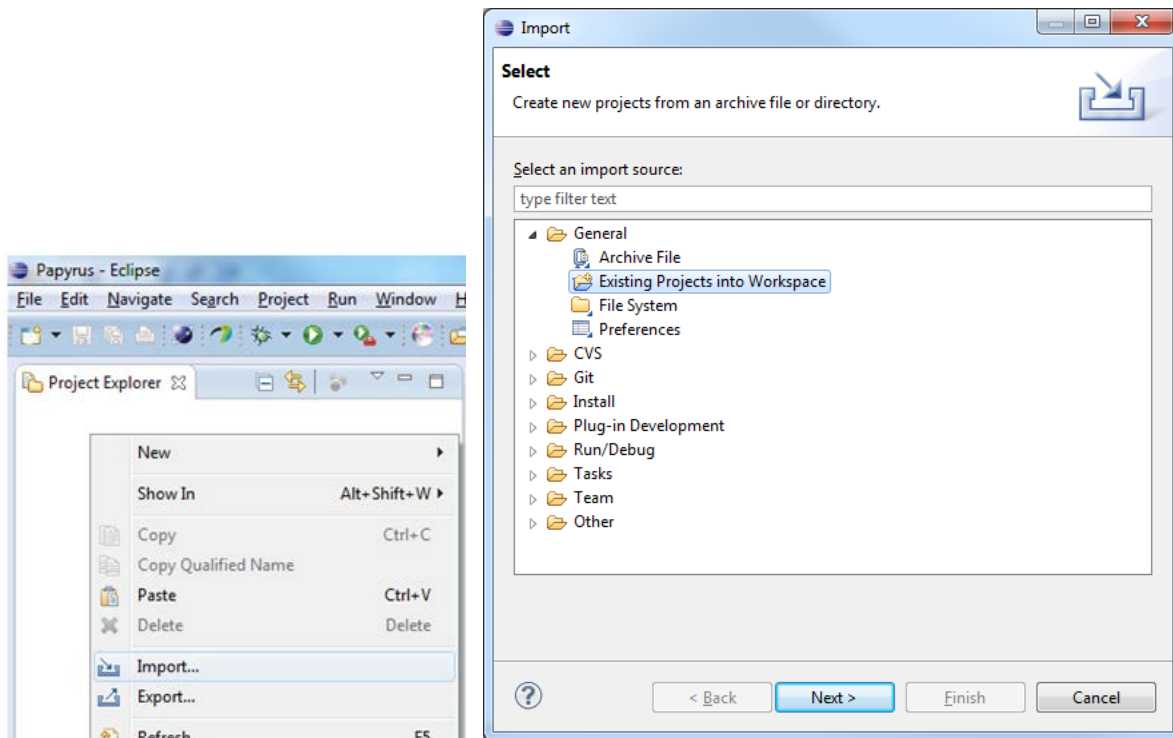



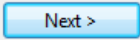
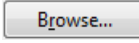


Figure 5-12: Importing a Model (1)

You need to select the  Existing Projects into Workspace option when the profile - that you want to import - contains a  .project file. Otherwise you need to create a new project and then import the profile using the  File System option.

Click  and then point via  to the folder containing the extracted Profile files.

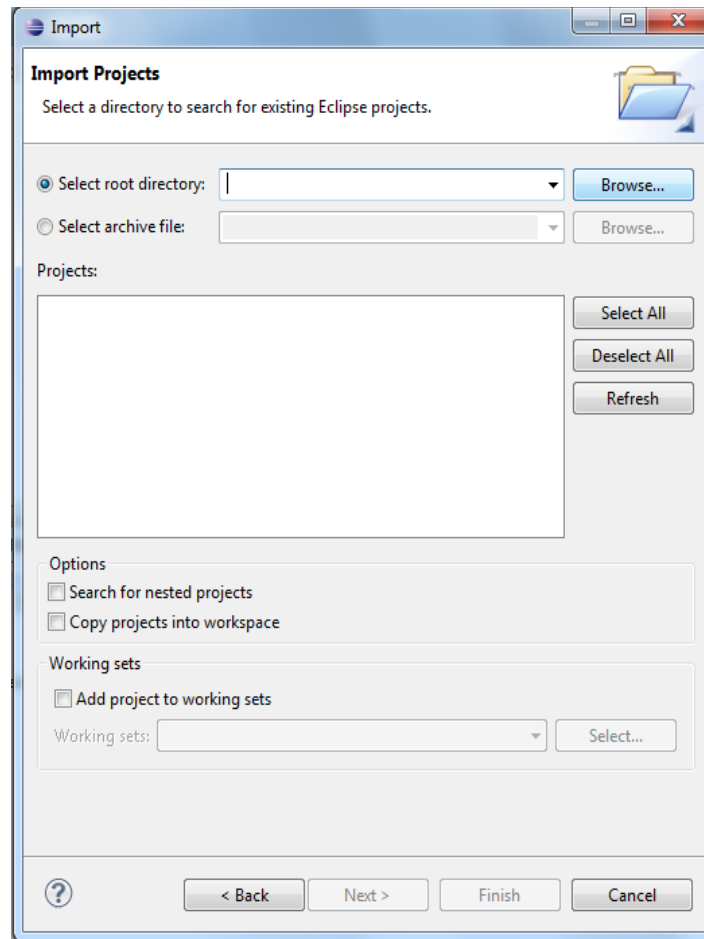
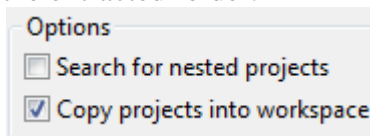


Figure 5-13: Importing a Model (2)

THEN select the option **Copy projects into workspace** if you want the Profile files copied into your workspace, otherwise Papyrus only creates a pointer and works with the files contained in the extracted folder:

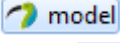




Click .

Note:

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

The Model is imported in the same way as the Profile.

A double click e.g., on  in the  **Project Explorer** opens the ONF_InformationModel in the  **Model Explorer**:

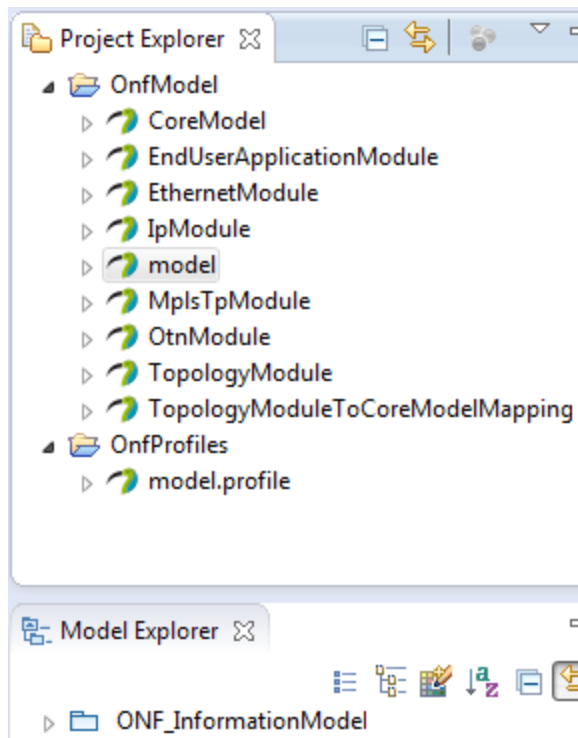






Figure 5-14: Open a Model

Now you can start working with the model.

5.4 Deleting a Project

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

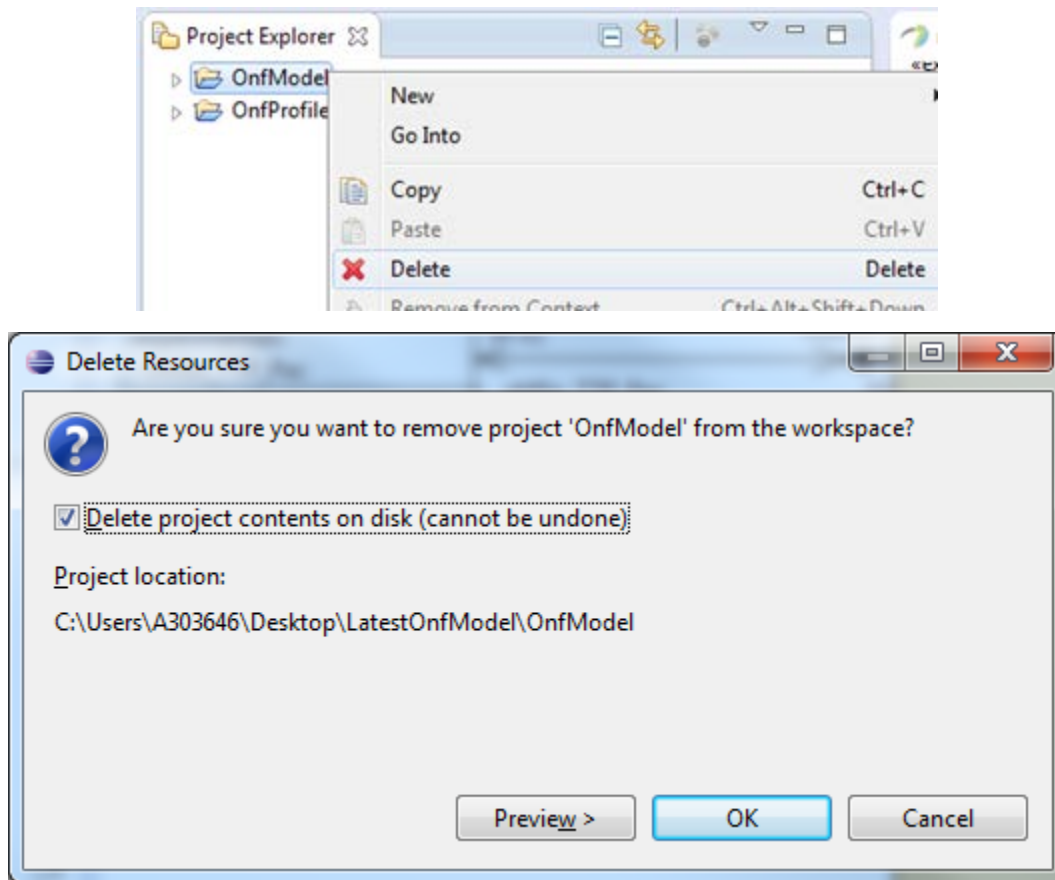


Figure 5-15: Delete a Project

6 Information Model on GitHub

Note:

The following GitHub usage description is using the ONF Model as an example. The steps are similar for any other model.

The ONF Information Model is stored in an ONF-specific area of GitHub. The name of the repository is “[ONFInfoModel](#)”.

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 ONF Information Model.

6.1 ONFInfoModel Structure on GitHub

The ONF Information Model is contained in a “[master branch](#)” on GitHub. A copy of this master branch is provided in the “[develop branch](#)”.

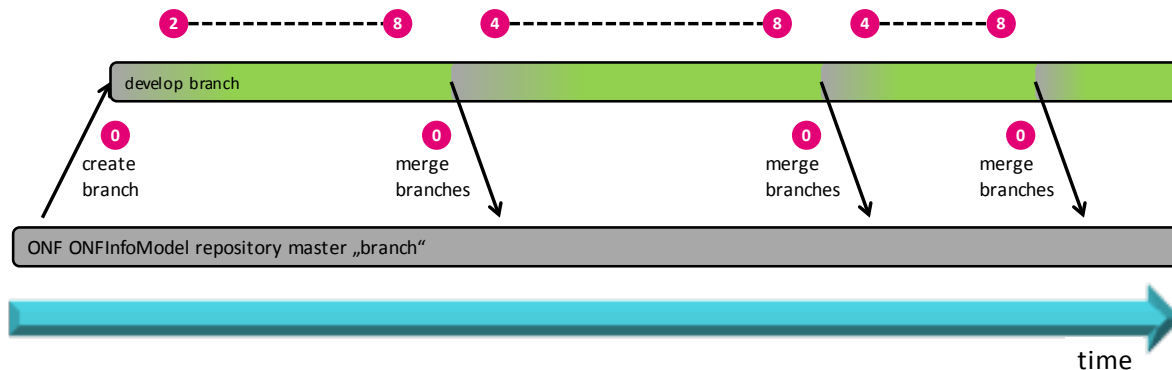


Figure 6-1: Initial ONFInfoModel Structure on GitHub

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 time¹.

6.2 GitHub Work Flow

The following steps describe the work flow that a modeler has to follow to establish an individual infrastructure for developing a piece of the ONF Information Model.

Section **Fehler! Verweisquelle konnte nicht gefunden werden.** describes a more easy way of getting the ONF Information Model 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 steps correspond with the numbers **1** in Figure 6.2.

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

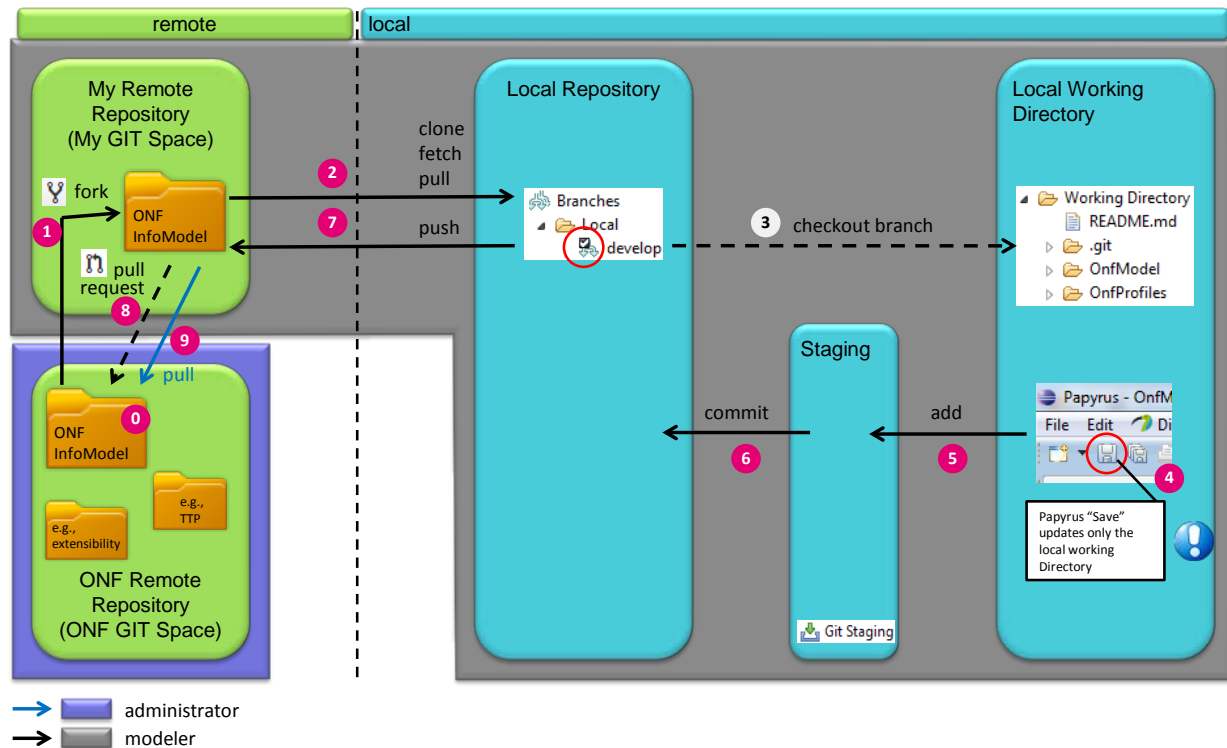





Figure 6-2: GitHub Work Flow

0. The administrator has established the ONFInfoModel repository (containing the master and develop branches) in the ONF git space under the following URL:
<https://github.com/OpenNetworkingFoundation/ONFInfoModel>



1. The modeler needs to copy the repository from the ONF git space into its own git space; by clicking  :



2. 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 contain already the OpenModelProfile and OnfModel projects.
 

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.

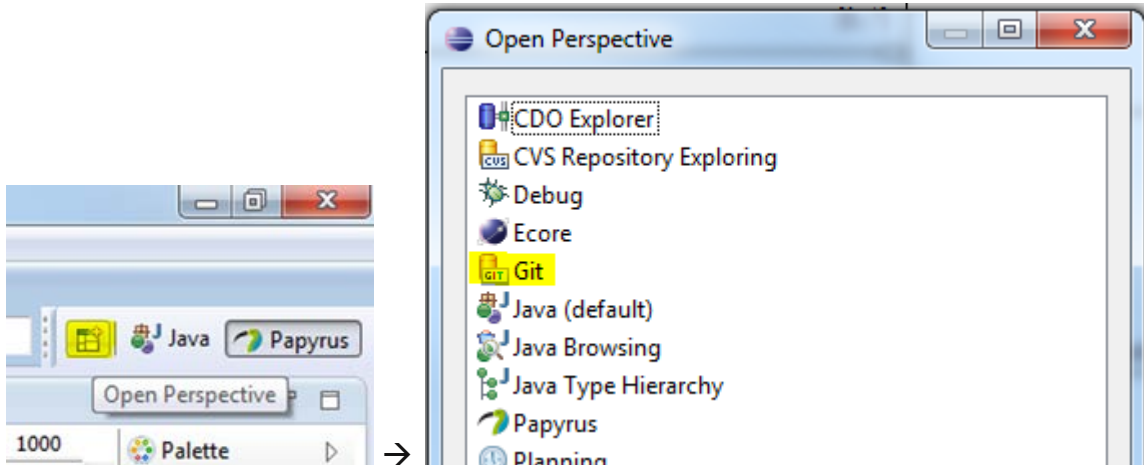


Figure 6-3: Open Git Perspective

In the  Git Repositories  window click on  [Clone a Git repository](#) .

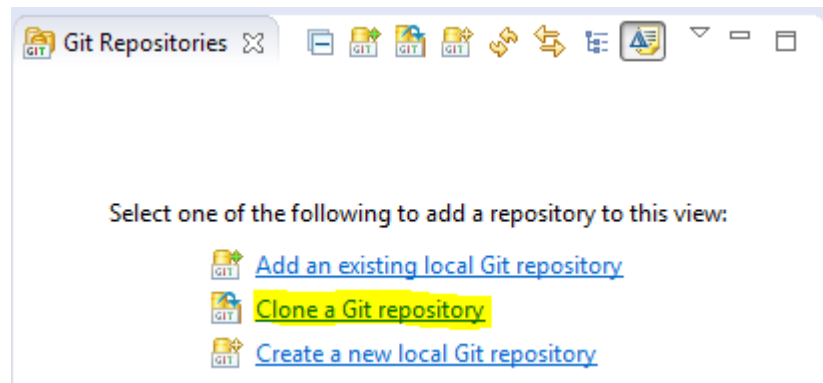



Figure 6-4: Add Repository Choices

Copy and paste the address that is provided on the web page of the ONFInfoModel in the

modeler’s git space  [bzeuner / ONFInfoModel](#) PRIVATE
forked from OpenNetworkingFoundation/ONFInfoModel :

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

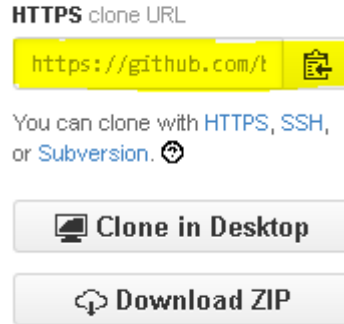


Figure 6-5: Location of the Repository Address

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

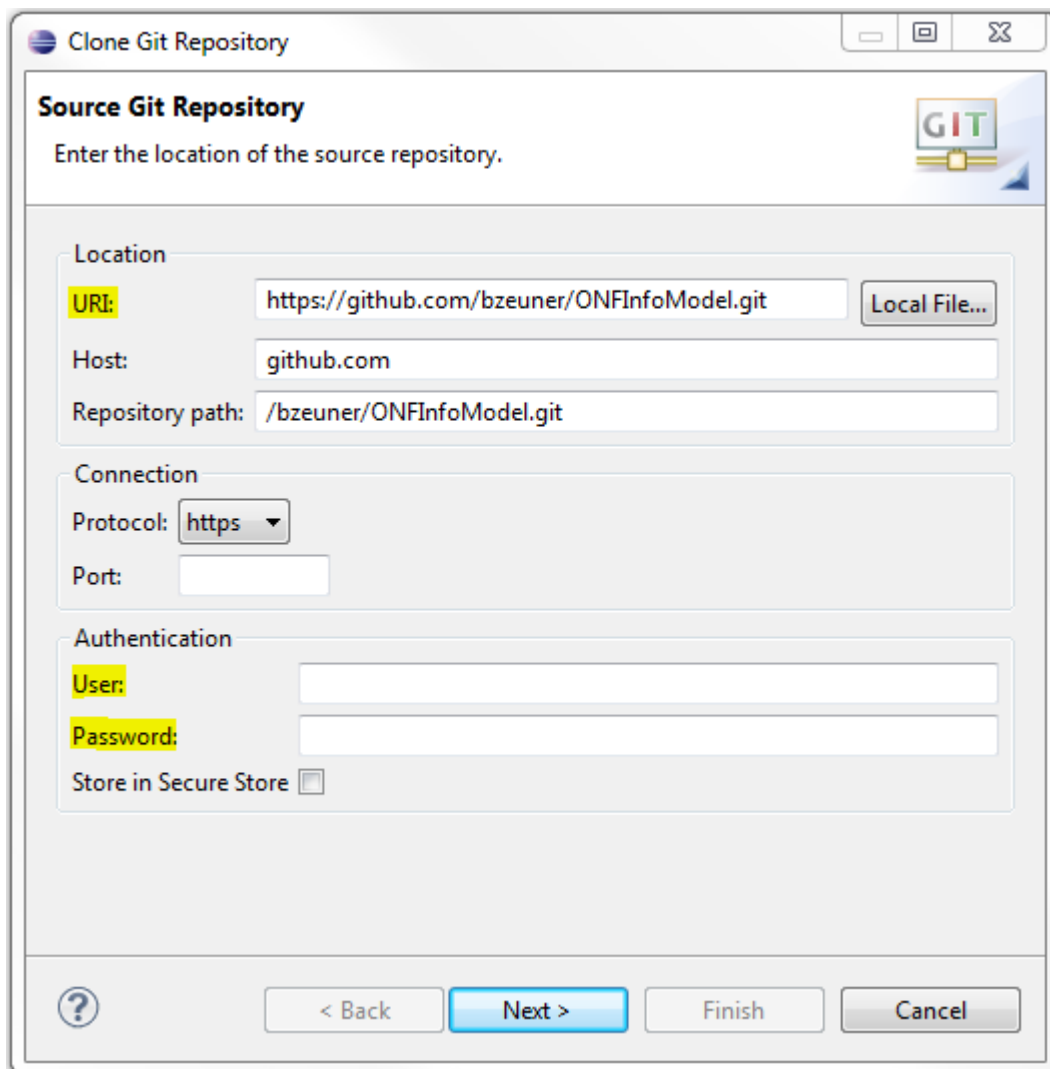


Figure 6-6: Source Git Repository Window

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

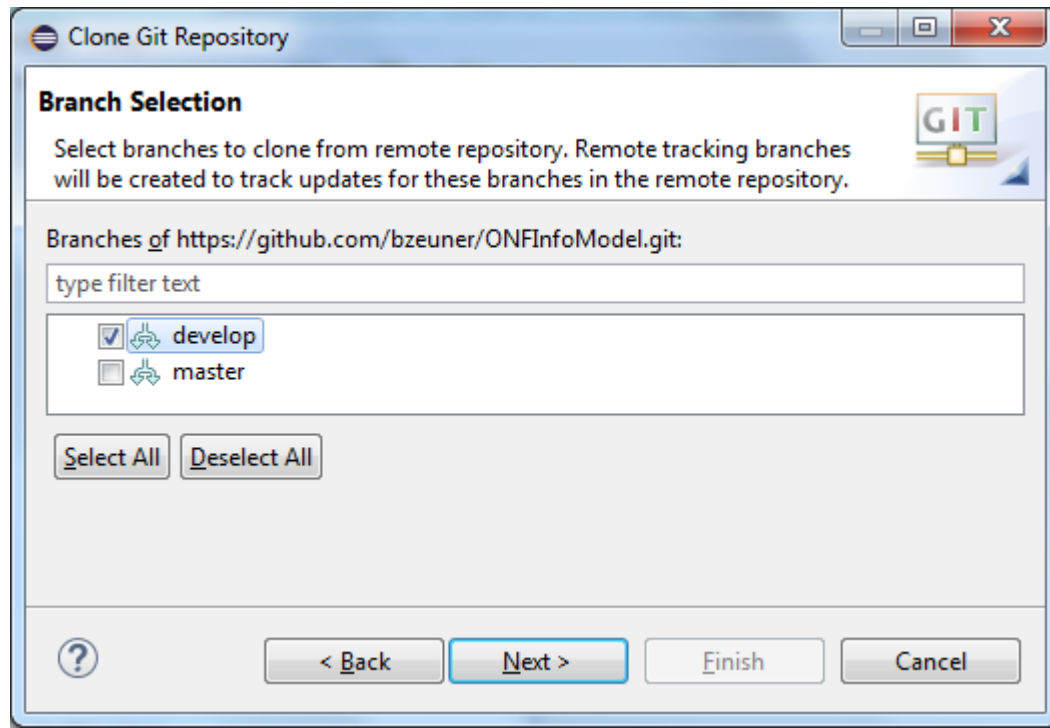


Figure 6-7: Branch Selection Window

Click  and insert the destination directory on your local disk where you want the model to be stored.

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

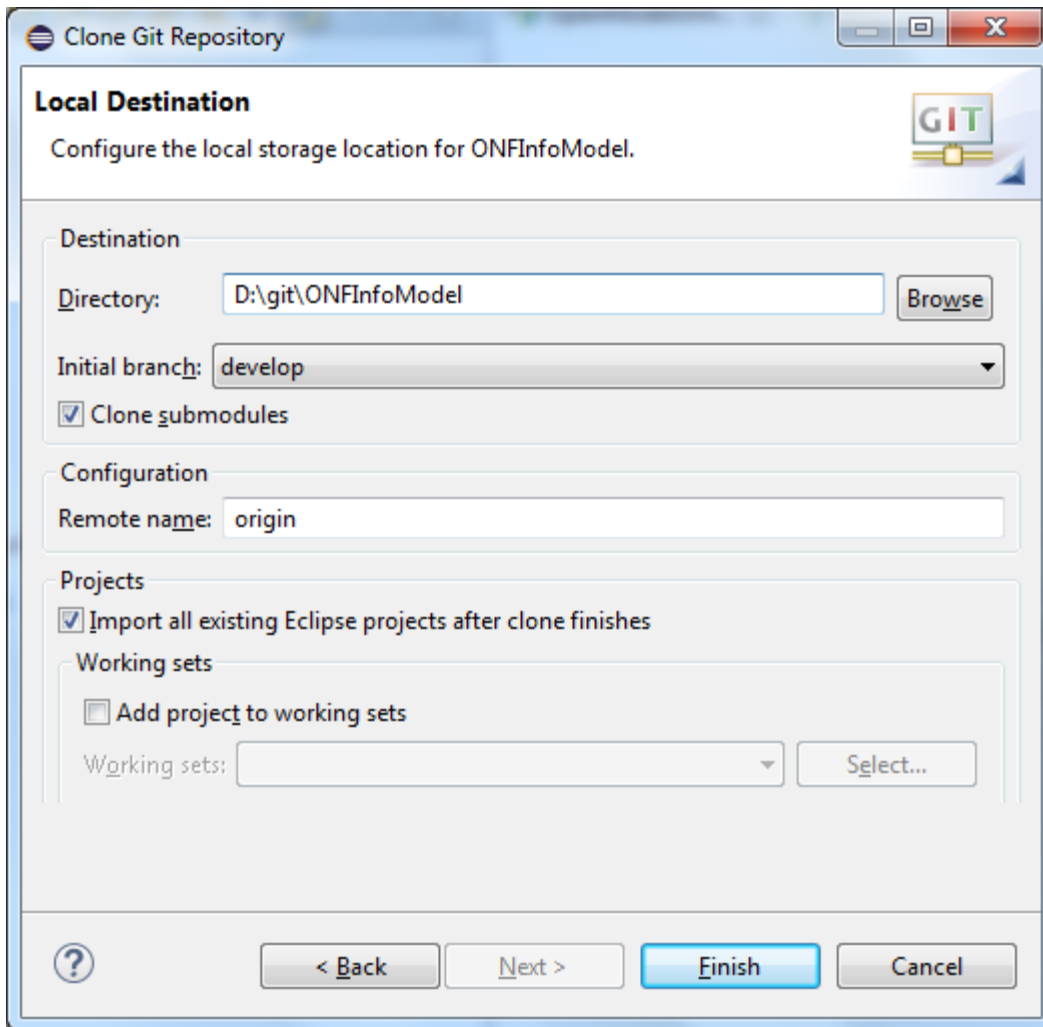
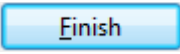




Figure 6-8: Local Destination Window

Click . The selected branch of the ONFInfoModel is now downloaded to your local PC.

The  Git Repositories  window should then contain the following files:

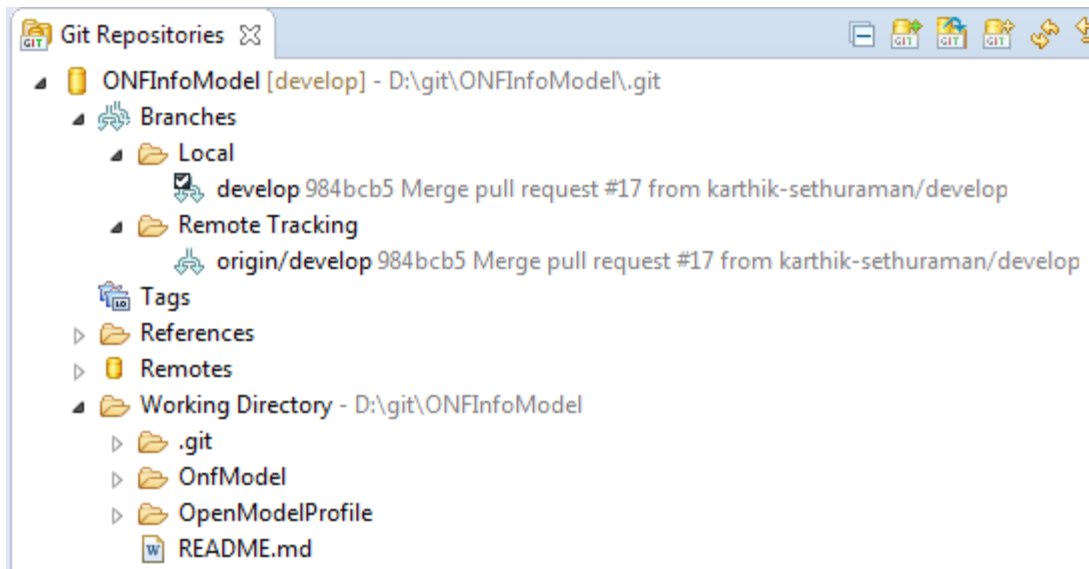


Figure 6-9: develop Branch Cloned to Local PC

- Since you have checked the **Import all existing projects after clone finishes** checkbox, the OpenModelProfile and OnfModel projects should automatically appear in the **Project Explorer** window in the Papyrus perspective. You may double verify this at the **Papyrus** perspective.

Note:

The complete model – including all sub-modules – is imported.

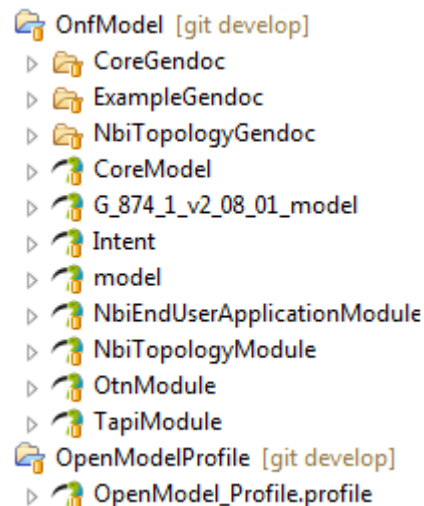





Figure 6-10: develop branch shown in Papyrus Project Explorer (snapshot)

Editor's note: Write protection of modules needs more investigation.

4. A double-click on the module of interest starts the modeling in Papyrus. E.g., a double-click on  `NbiTopologyModule` in the  `Project Explorer` window opens the `NbiTopologyModule` model in the  `Model Explorer` window.

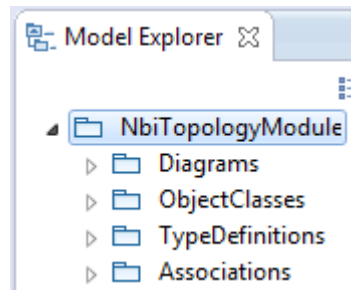





Figure 6-11: NbiTopologyModule Shown in Papyrus Model Explorer

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 package  `NbiTopologyModule` then going to  `Import` /  `Import Registered Package` and then select `UMLPrimitiveTypes` :

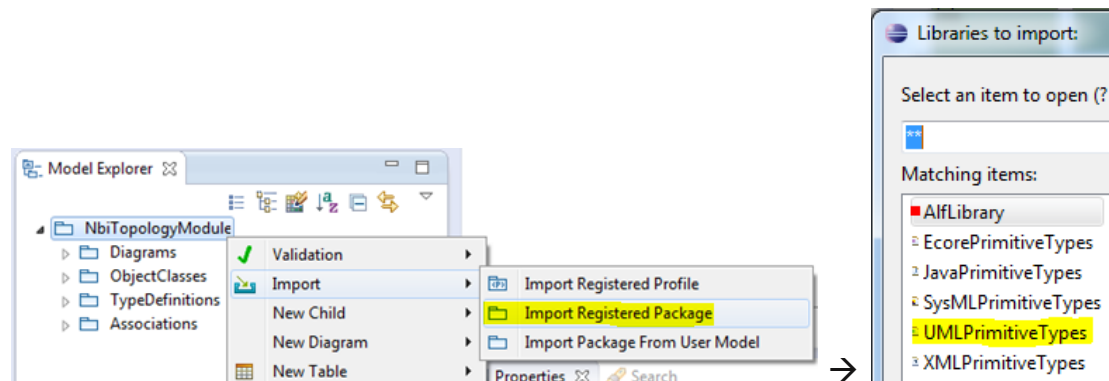





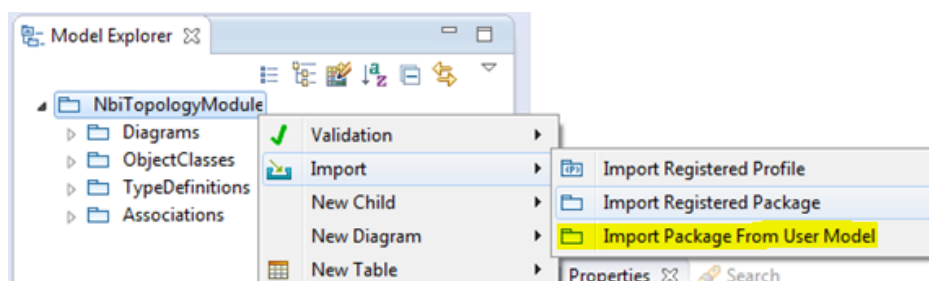


Figure 6-12: Importing UML Primitive Types

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  `NbiTopologyModule` then via  `Import` and  `Import Package From User Model` select  `OnfModel [ONFInfoModel north_bound_interface_develop]` /  `CoreModel.uml` .



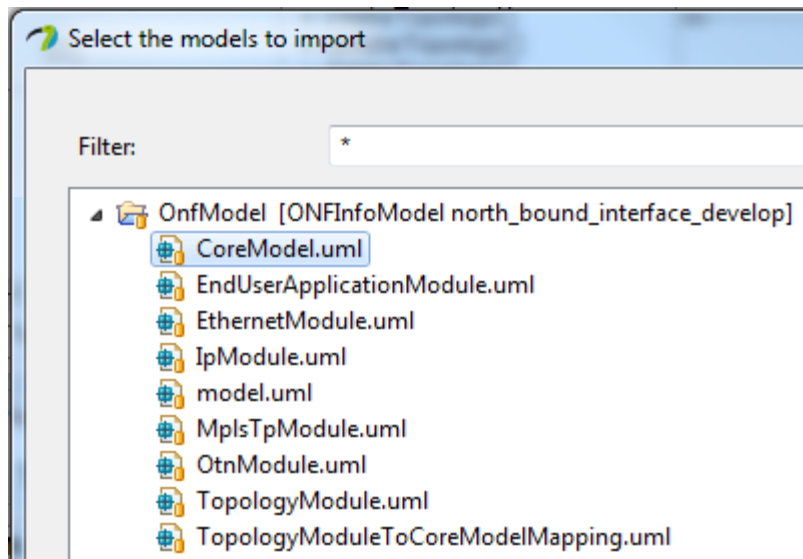


Figure 6-13: Importing Core Model Artifacts

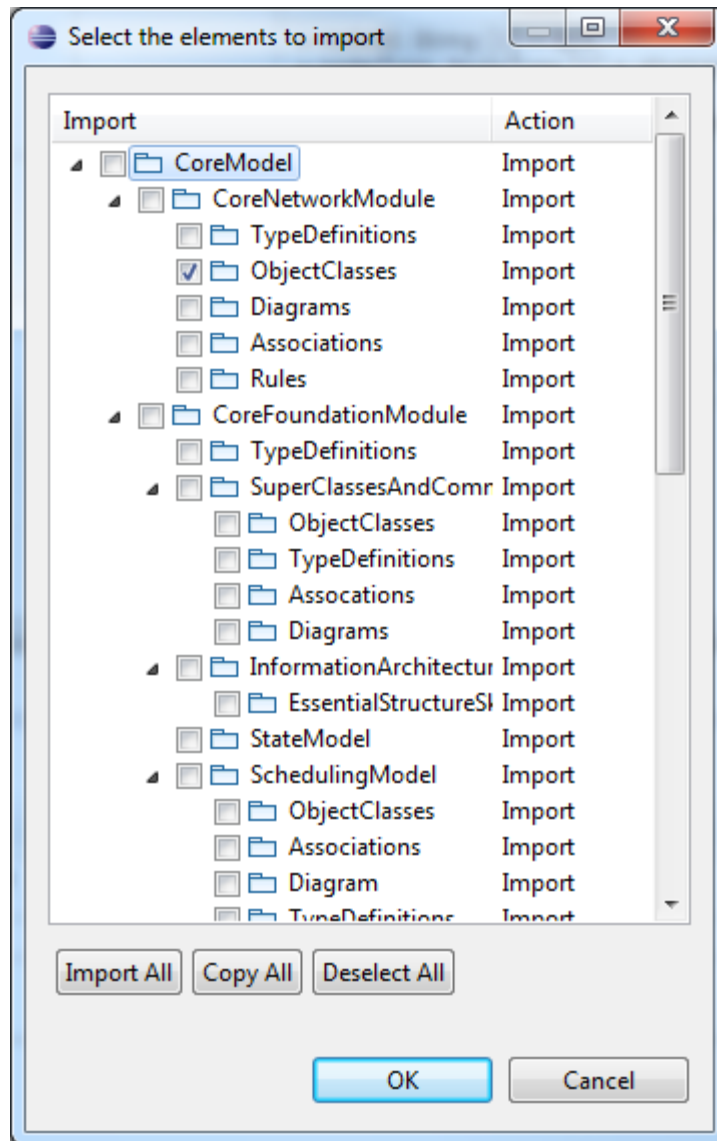


Figure 6-14: Selecting Core Model Artifacts

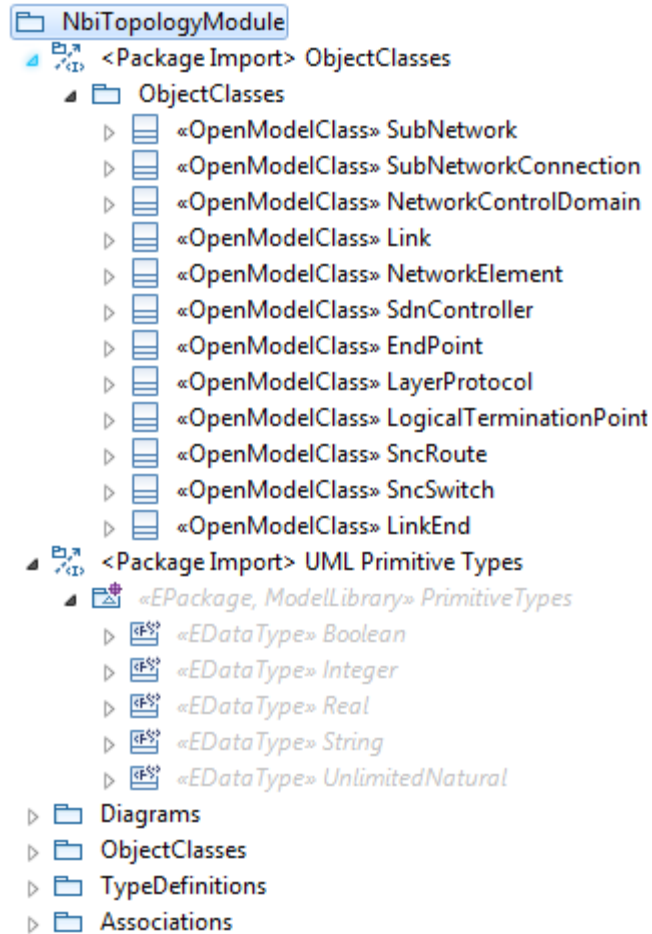
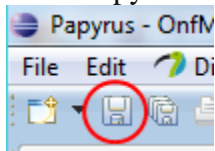




Figure 6-15: Imported Core Model Artifacts

The designer of the NbiTopologyModule can now develop the model. The allowed actions are described in section 7.5.

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



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

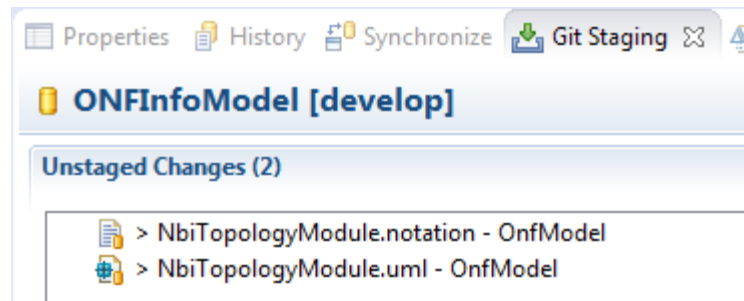


Figure 6-16: Unstaged Changes in Git Staging

Select all TopologyModule files, right-click and select **Add to Index** :

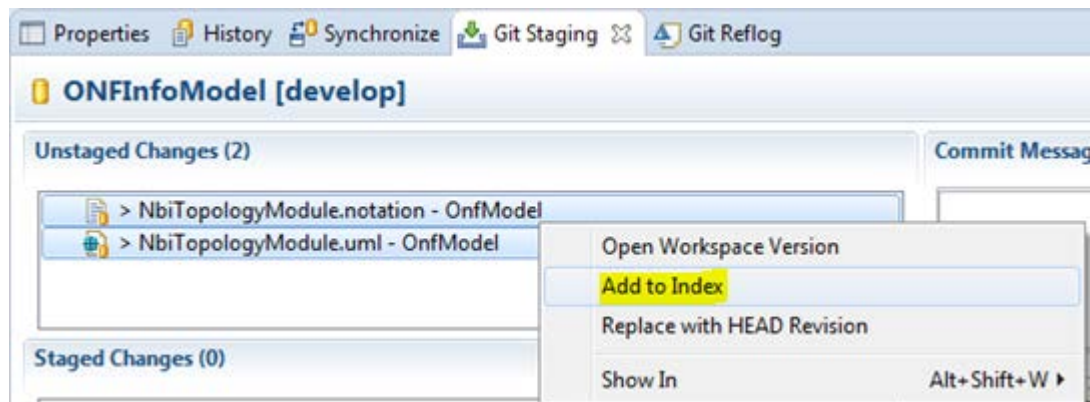


Figure 6-17: Add Files to Git Stage

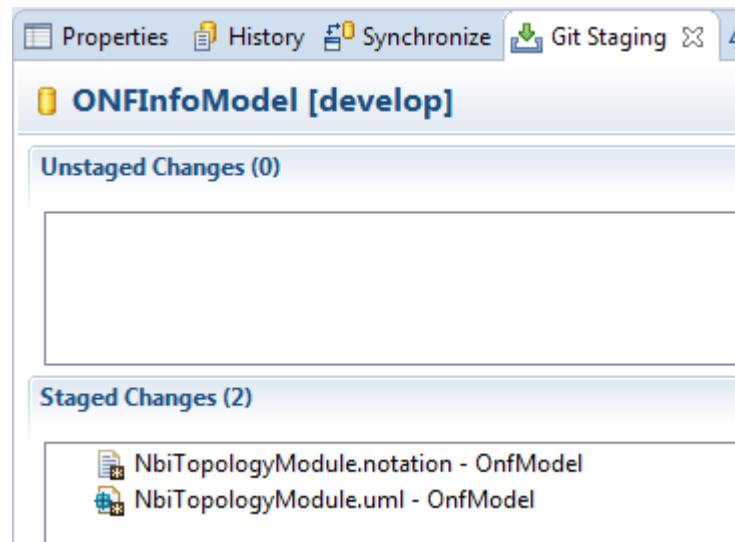






Figure 6-18: Staged Changes in Git Staging

6. To commit the staged changes to your local repository you need to provide a commit message describing the changes that were done and then click on  **Commit** .

7. 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 you need to right-click on the repository in the  Git Repositories  tab and select  Push Branch 'develop'...

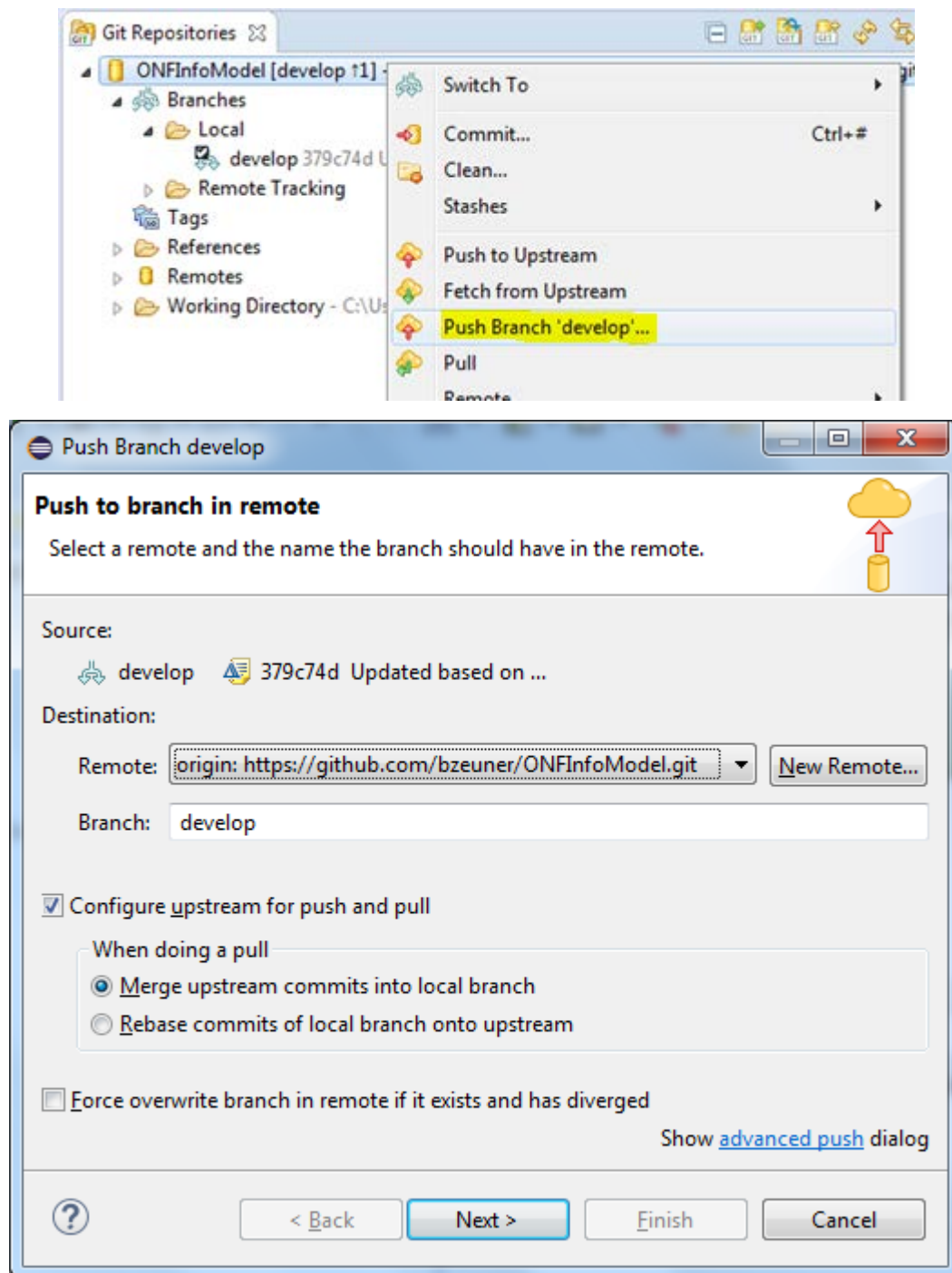


Figure 6-19: Push Updated Branch to Remote Repository

Make sure you have checked **Configure upstream for push and pull** and **Merge upstream commits into local branch**.

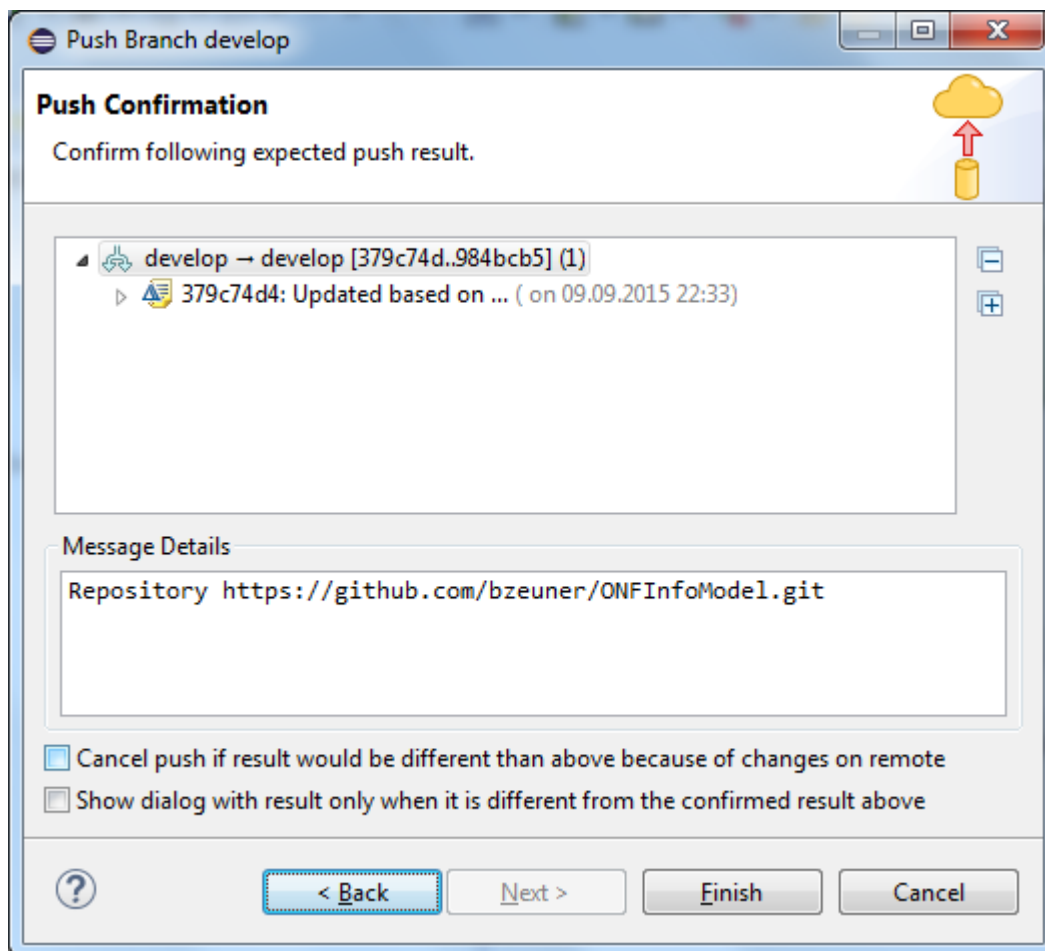


Figure 6-20: Push Confirmation Window

Finally click .

- Steps 3 – 7 can be done as often as necessary. Once all updates of the sub-model for the next release of the ONF Information Model 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 .

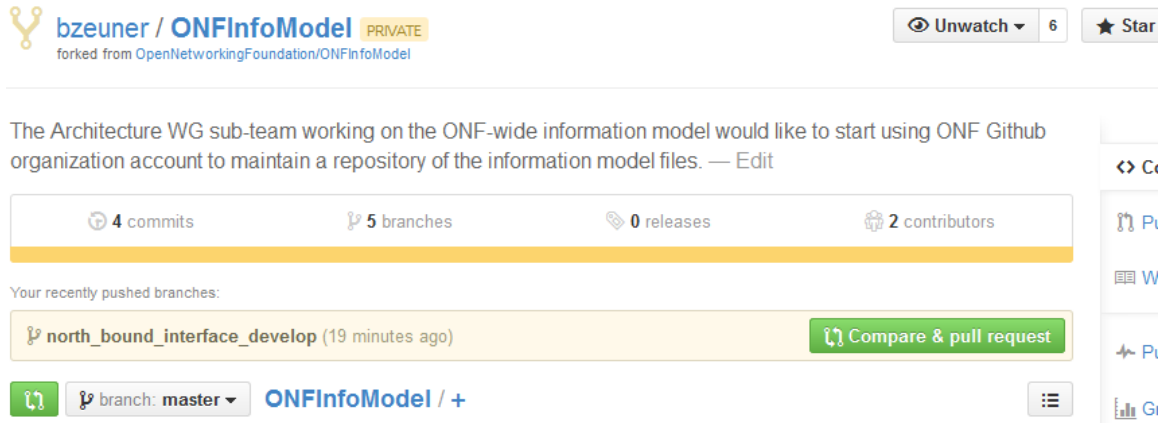


Figure 6-21: Compare in Modeler's Remote Repository

After click on [Compare & pull request](#) or [git](#) provides a detailed comparison of all changes done in all files.

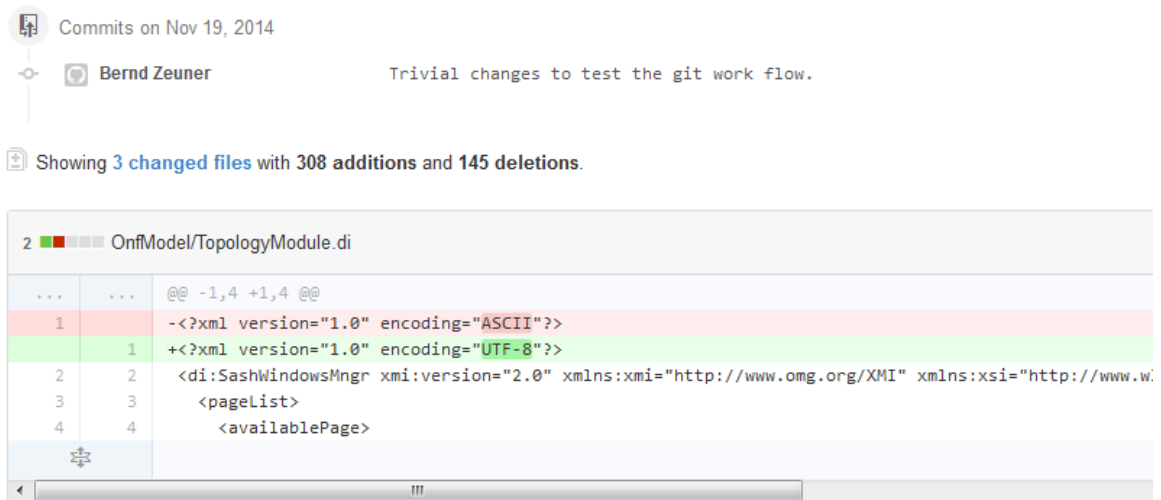


Figure 6-22: Detailed Comparison in Modeler's Remote Repository

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 [Create pull request](#).

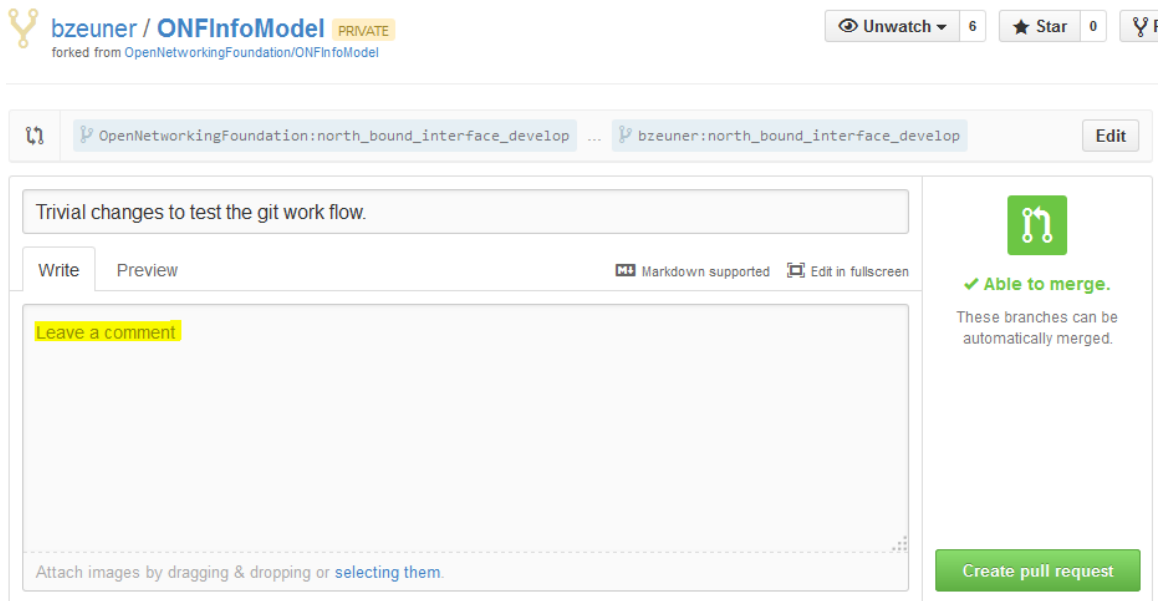


Figure 6-23: Pull Request in Modeler’s Remote Repository

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

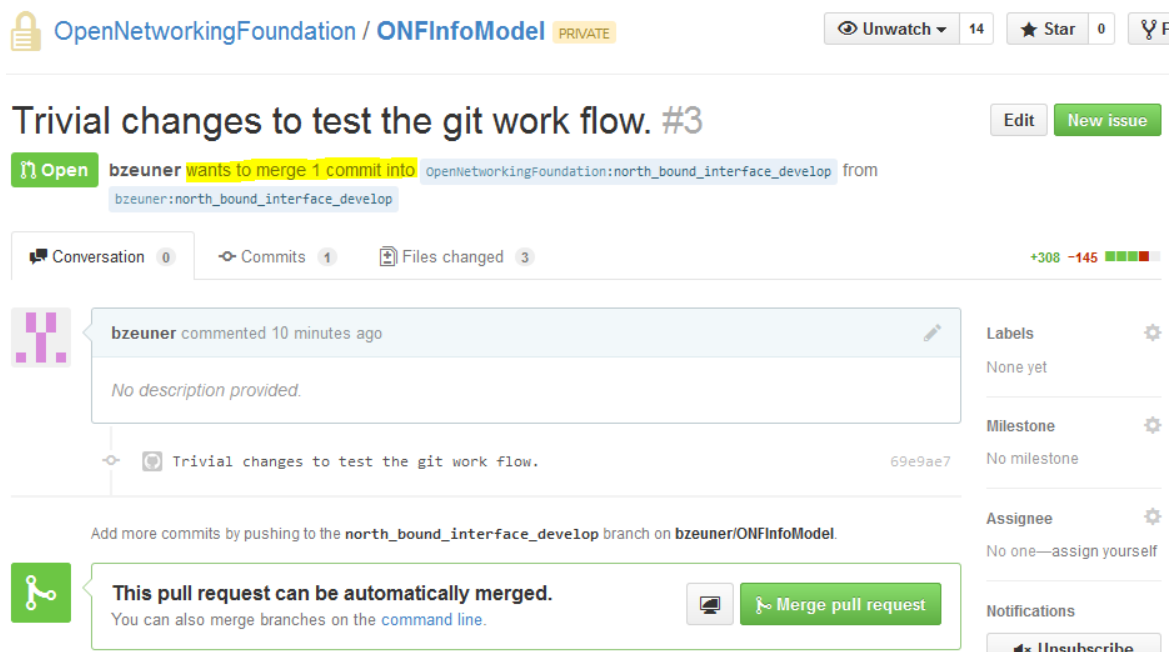



Figure 6-24: Pull Request in Administrator’s Remote Repository

6.3 Downloading a Model from github for “Read Only Use”

This section describes a more easy way of getting the ONF Information Model 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.

0. The ONFInfoModel repository is located in the ONF git space under the following URL: <https://github.com/OpenNetworkingFoundation/ONFInfoModel>.

1. Click the  button on the bottom right corner of the github web page to download the repository to your local PC.

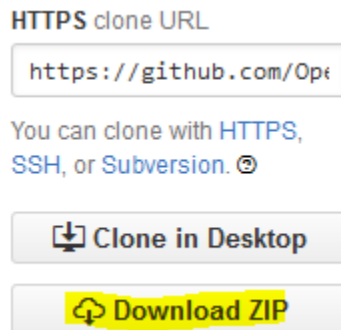


Figure 6-25: Download ONFInfoModel Repository

2. Extract the zip-file to the desired Eclipse Workspace.

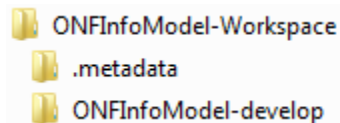





Figure 6-26: Extract ONFInfoModel Repository to Worksoace

3. Make the ONFInfoModel visible in the Papyrus Project Explorer by importing the model and profile projects.

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

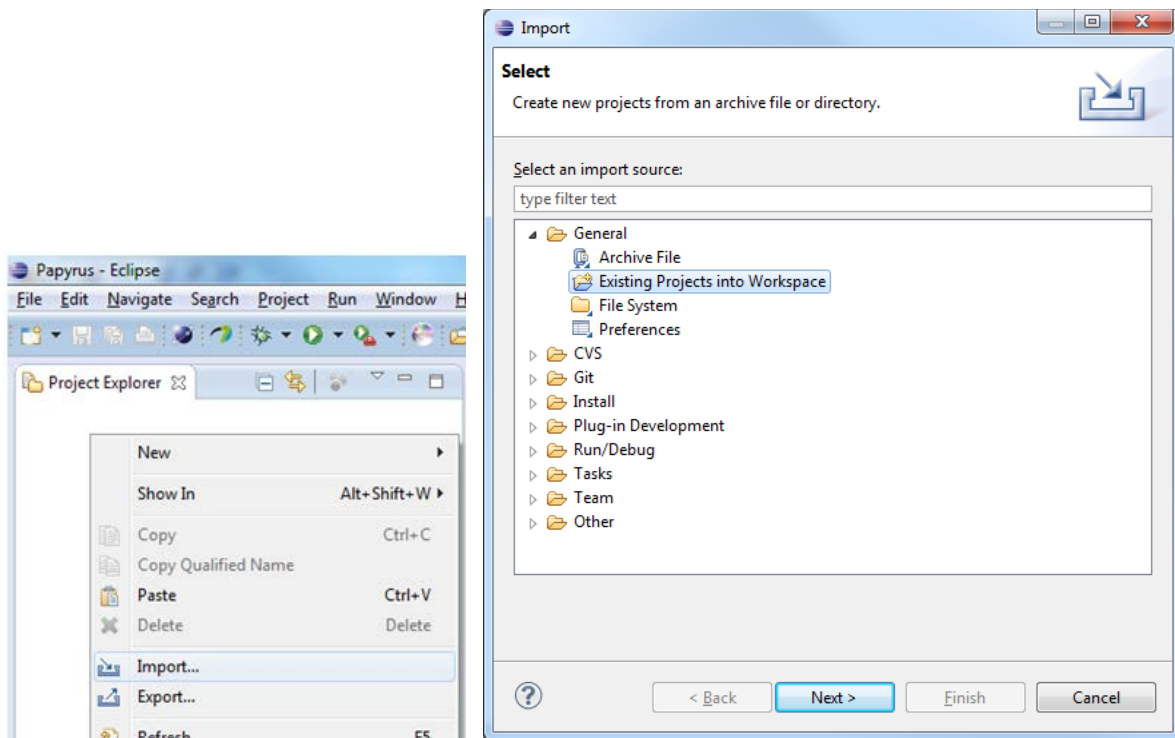


Figure 6-27: Making the ONFInfoModel visible in Papyrus (1)

You need to select the **Existing Projects into Workspace** option since the downloaded repository contains already the `.project` files for the model and profile.

Click and then point via to the folder in your workspace containing the extracted repository files; i.e., **ONFInfoModel-develop**.

OnfModel and OpenModelProfile are already selected in the Projects box.

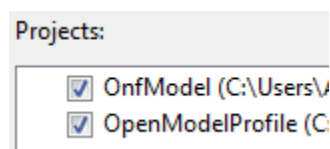
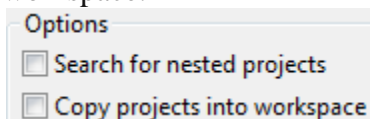


Figure 6-28: Making the ONFInfoModel visible in Papyrus (2)

Do not select the option **Copy projects into workspace** since the files are already in your workspace:



Click .

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.

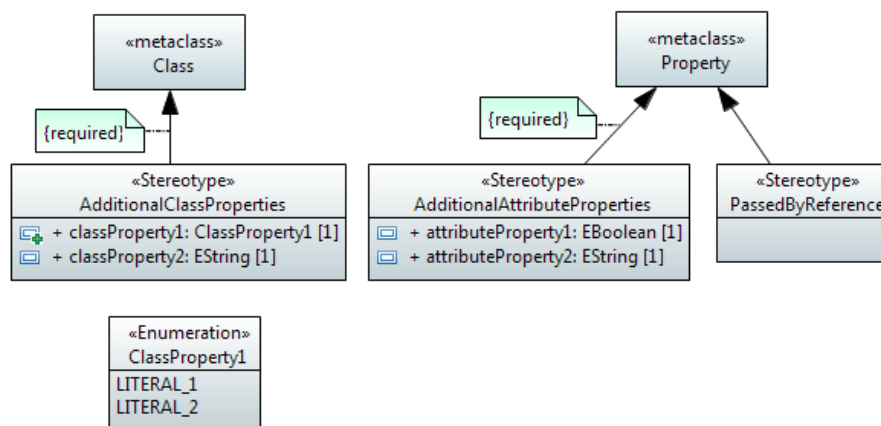


Figure 7-1: Illustrative UML Profile

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.

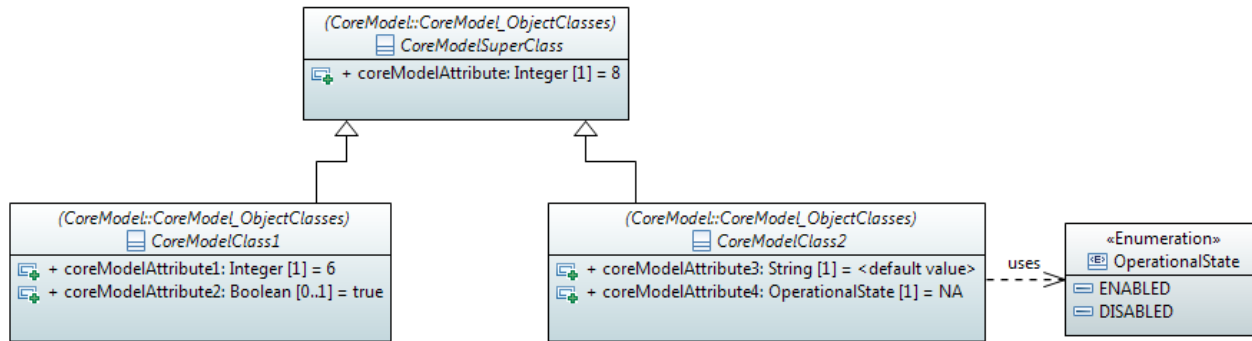


Figure 7-2: Illustrative Core Model

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.

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

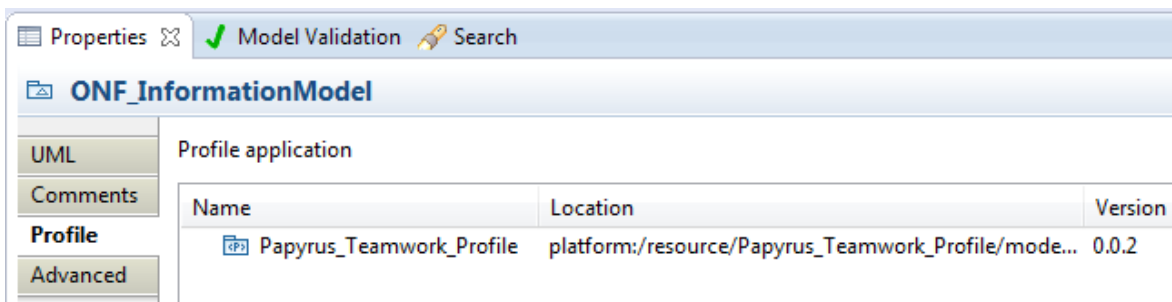
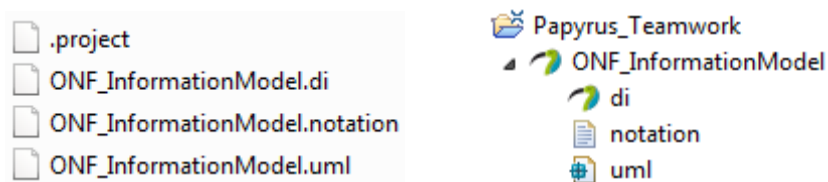


Figure 7-3: Profile Associated to the Model

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))

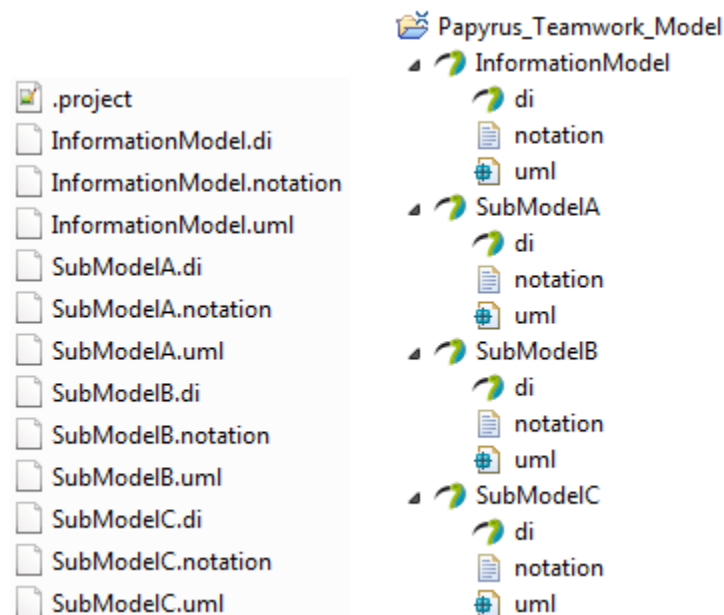
Figure 7-4: Papyrus File Structure

As already mentioned in section 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 must not be changed.


Changes in the other parts of the model are only allowed by the respective model designer.

The sub-model designer must be able to relate the sub-model object classes to the core object classes and to use the data types defined in the core-model.

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. Section 6.2 (step 4) explains how to import additional artefacts.

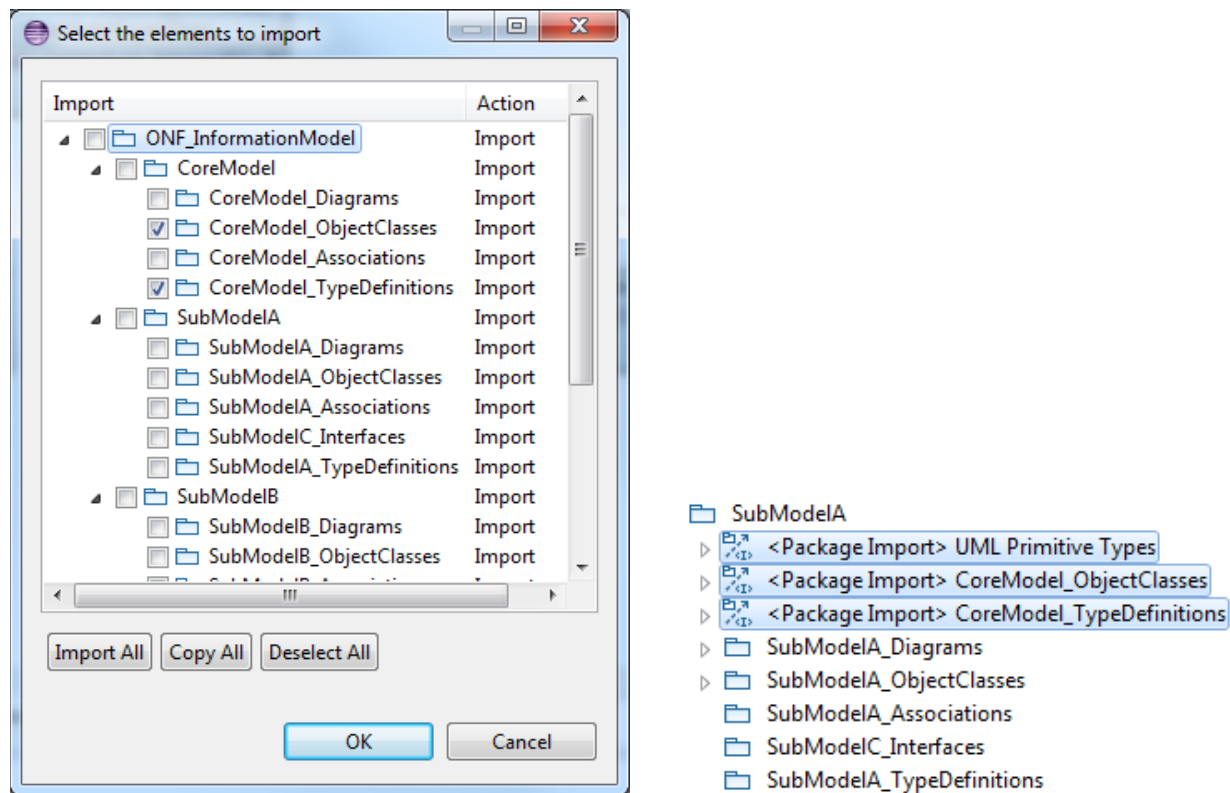


Figure 7-6: Imported UML Artifacts

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 Team UML Model Development

The ONF-wide Information Model is developed by different teams. The IMP Modeling team is responsible for the core-model and additional teams for each sub-model.

The IMP Modeling team is also responsible for the organization of the whole modeling work. It provides the basic model files for each sub-model team and merges the sub-models back to the overall ONF-wide Information Model.

The IMP Modeling team creates a zip-file per sub-model team (e.g., sub-model A) which contains the OpenModel Profile, the core-model and all existing/planned sub-models at that time. Only the specific sub-model files of sub-model A are writeable (highlighted in green in Figure 7.7), all other files are write protected.

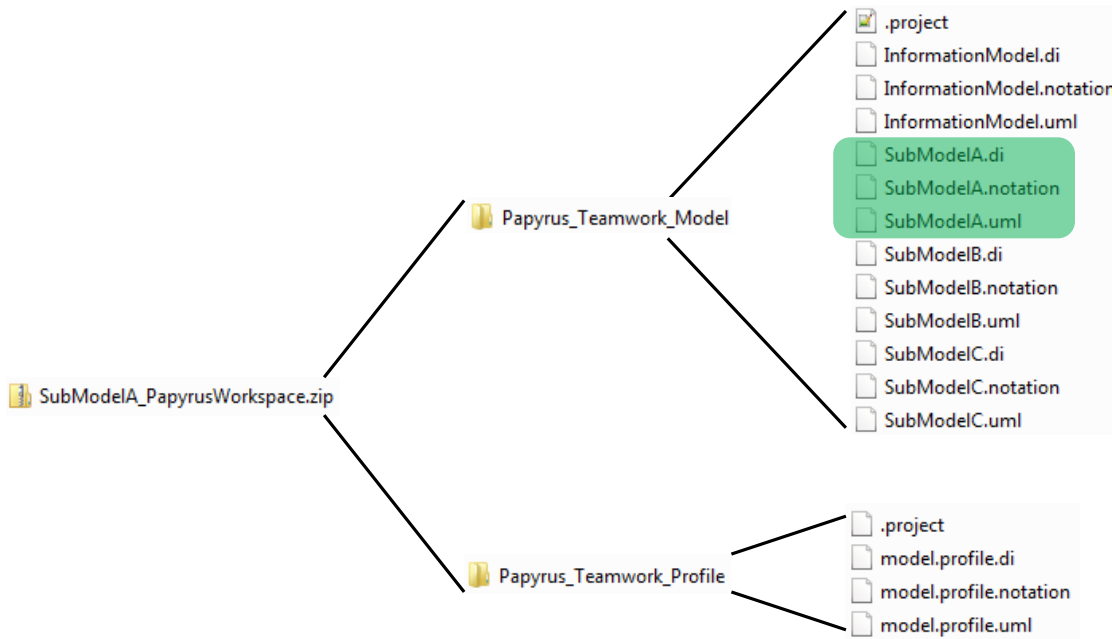


Figure 7-7: Information Model File Structure

The sub-model designer imports the Open Model Profile (here Papyrus_Teamwork_Profile) into the workspace via **Import...** **Existing Projects into Workspace**:

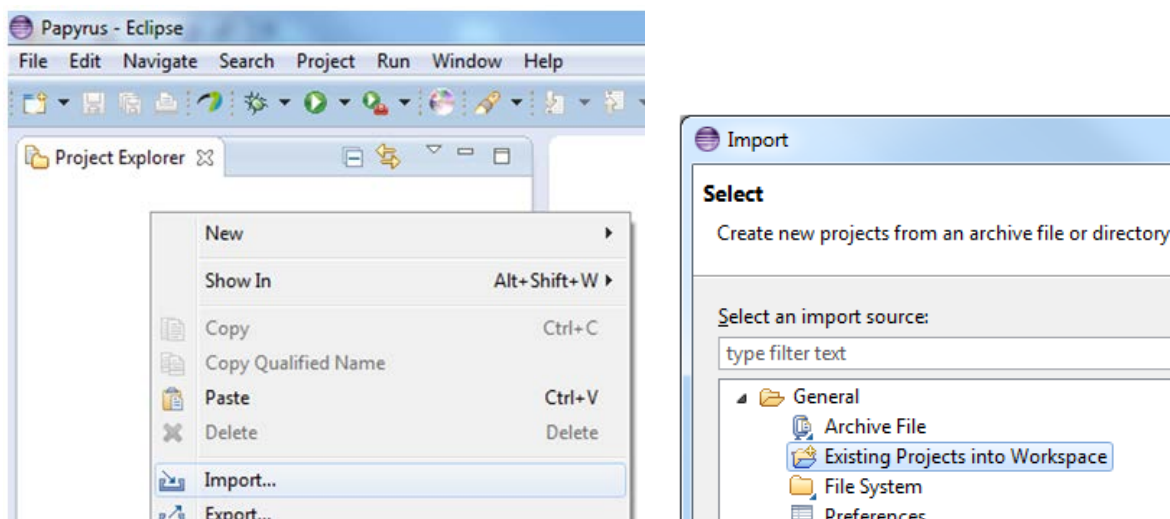




Figure 7-8: Importing an Existing Project into Papyrus

After importing the Profile, the sub-model designer imports the ONF Information Model (here Papyrus_Teamwork_Model) into the workspace via **Existing Projects into Workspace**.


 The Profile must be imported **before** the Information Model.
 Otherwise, the Profile is not associated to the Model.
 

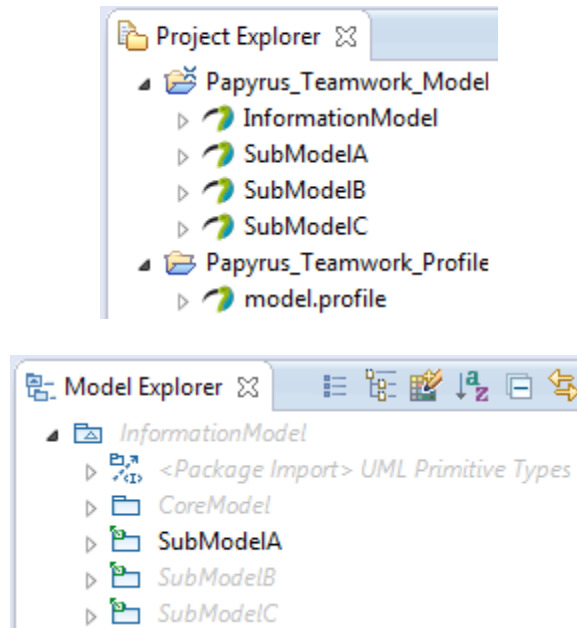








Figure 7-9: Project and Model Explorer View after Import into Papyrus

The  Model Explorer  shows the CoreModel, SubModelB and SubModelC in grey because these models are write protected.

The sub-model A designer can now develop the model.


 Sub-Model A designer **must** select the core model in the  Project Explorer .
 I.e., double click on “InformationModel”; **not** “SubModelA”.
 

The core-model must be selected after every saving of the model.

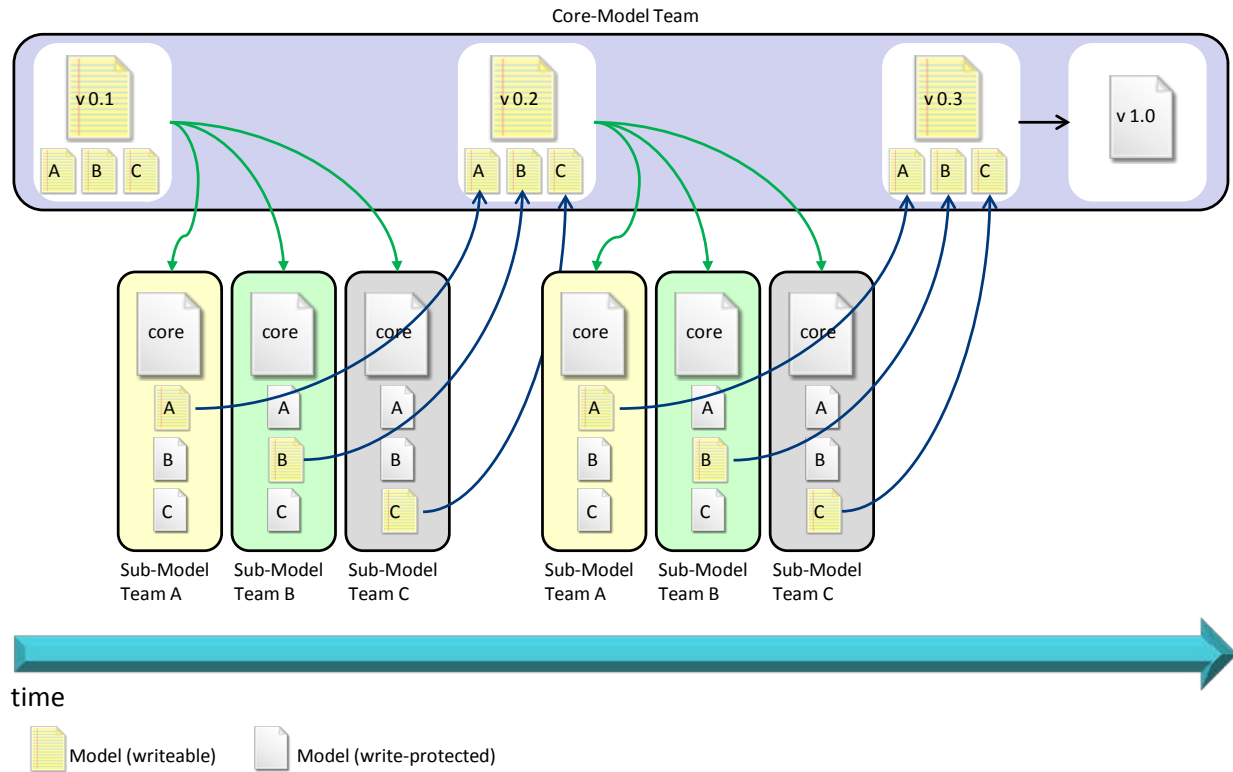


Figure 7-10: Modeling Process over Time

7.5 Developing a Sub-Model

The designer of the sub-model can now develop the model:

- Creating object classes
- Setting additional properties (defined in the Profile) of the object classes
- Adding attributes to object classes
- Setting additional properties (defined in the Profile) of the attributes
- Using data types defined in the core-model
- Inheriting sub-model object classes from core-model object classes
- Creating associations from sub-model object classes to core-model object classes
- ...

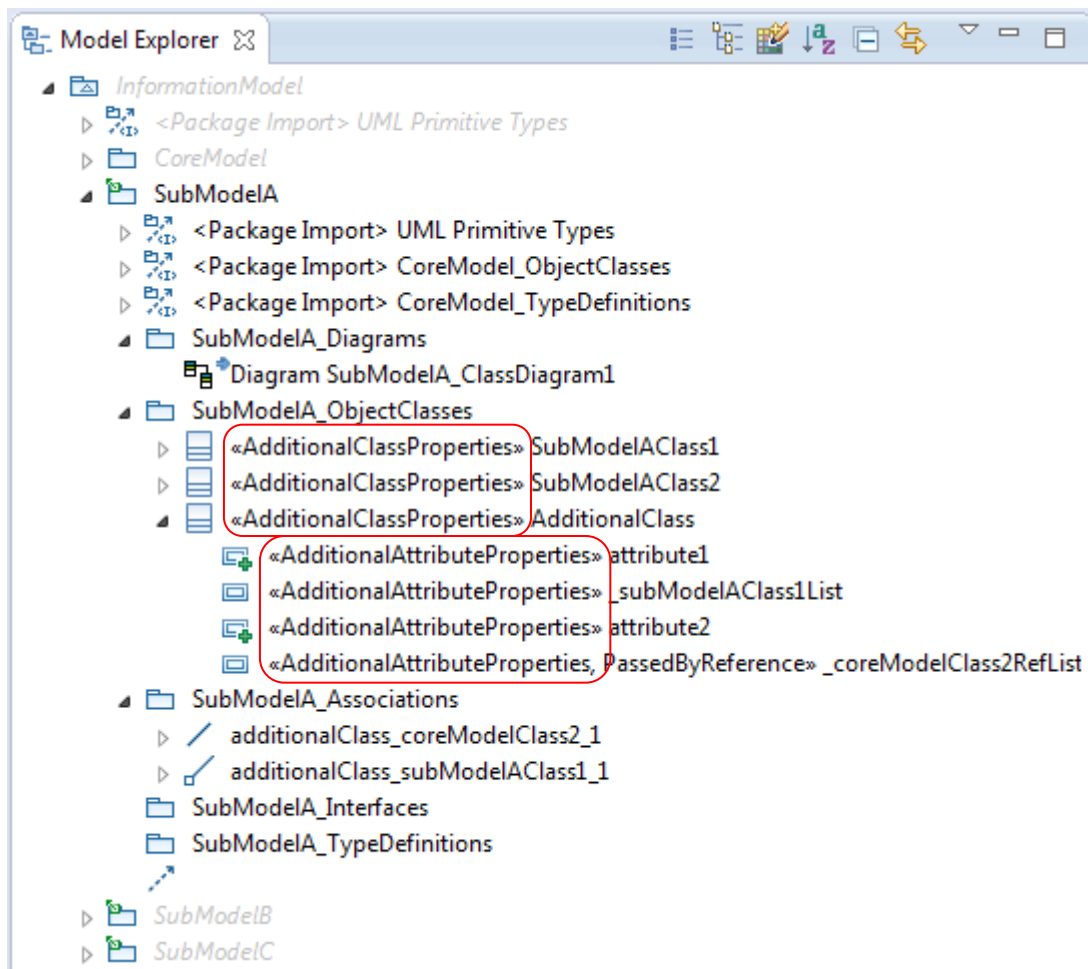
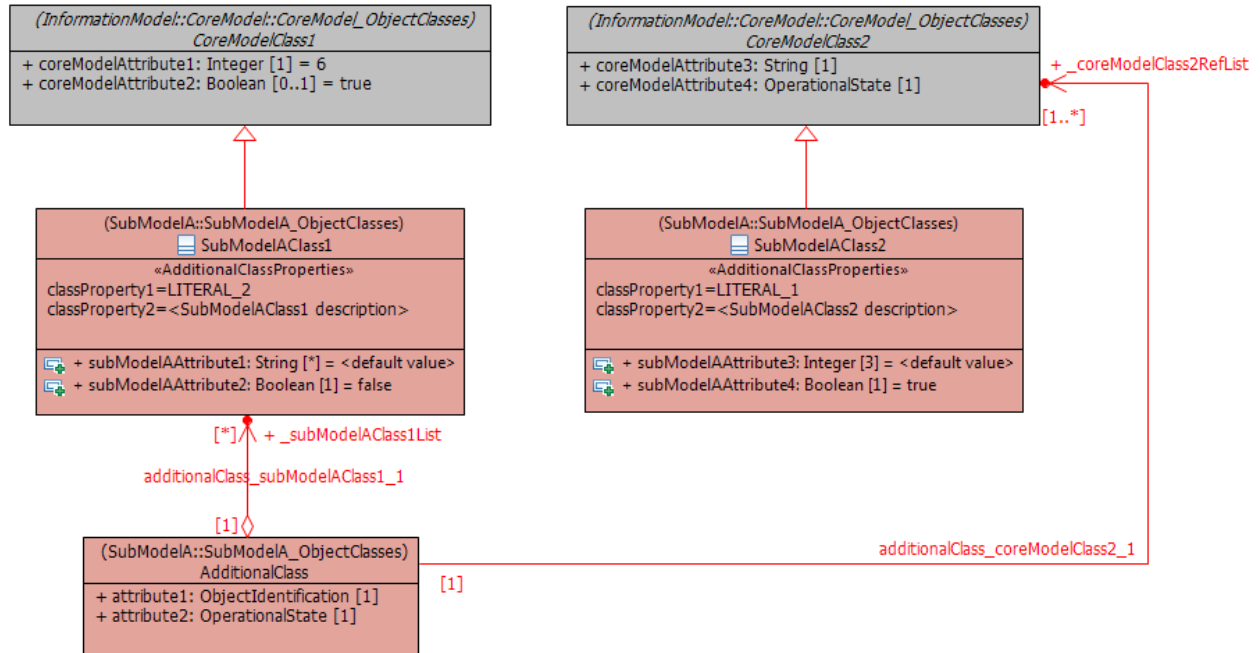
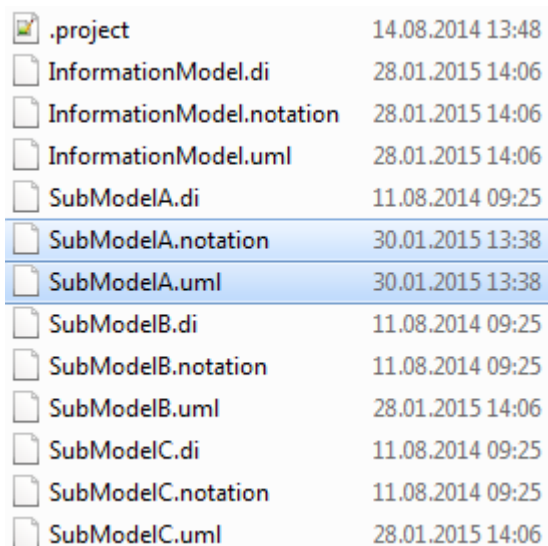


Figure 7-11: Example Sub-Model A (highlighted in red)

Note: The stereotypes in front of the class/attribute names (red boxes) indicate that the class/attribute has the additional properties illustrated in Figure 7.1.

The development of sub-model A is stored in the .uml and .notation files; i.e., these are the only files that are updated:
















| | |
|---|------------------|
|  .project | 14.08.2014 13:48 |
|  InformationModel.di | 28.01.2015 14:06 |
|  InformationModel.notation | 28.01.2015 14:06 |
|  InformationModel.uml | 28.01.2015 14:06 |
|  SubModelA.di | 11.08.2014 09:25 |
|  SubModelA.notation | 30.01.2015 13:38 |
|  SubModelA.uml | 30.01.2015 13:38 |
|  SubModelB.di | 11.08.2014 09:25 |
|  SubModelB.notation | 11.08.2014 09:25 |
|  SubModelB.uml | 28.01.2015 14:06 |
|  SubModelC.di | 11.08.2014 09:25 |
|  SubModelC.notation | 11.08.2014 09:25 |
|  SubModelC.uml | 28.01.2015 14:06 |

Figure 7-12: Updated Sub-Model A Files (highlighted in blue)

The ONF Information Model team takes these two files and overwrites the corresponding files in the original model workspace. The updated individual sub-models can now be “re-integrated” into the single ONF Information Model.

Notes:

After re-opening the model in the original model workspace, the data types (imported from the core-model) are automatically related to the core-model data types and the core-model classes used in the sub-model class diagrams are automatically related to the corresponding classes in the core-model.

Adding new data types to the type definitions in the core-model automatically adds them also to the imported type definitions in the sub-model.

8 Extracting Data from a Papyrus model

This section describes how to extract diagrams, comments and details for a Data Dictionary from a Papyrus model. There are two ways of extracting information from a Papyrus model:

- Using the Gendoc plugin
 - Recommended method for document generation
 - Provides a Microsoft Word document

- Enables extraction of diagrams as well as model content such as classes, comments etc
- Using the Table function from Papyrus
 - Alternative method for extracting model content into a table form
 - Provides a basic format

8.1 Gendoc Plugin

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 section provides instructions on how to install Gendoc followed by guidance on construction of Gendoc templates along with example fragments of templates.

8.2 Installing Gendoc

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

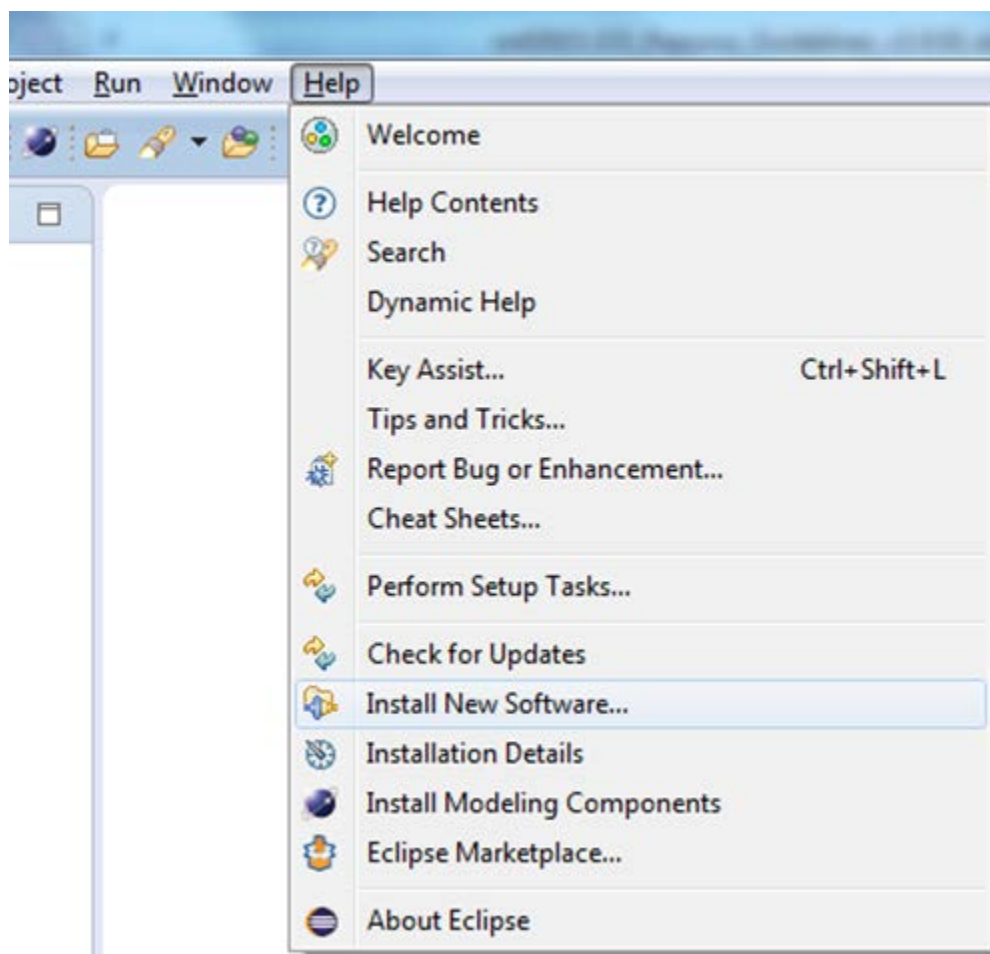


Figure 8-1: Installing Gendoc (1)

Click  and enter the Gendoc 0.5.0 update site:
<http://download.eclipse.org/gendoc/updates/releases/0.5.0/>

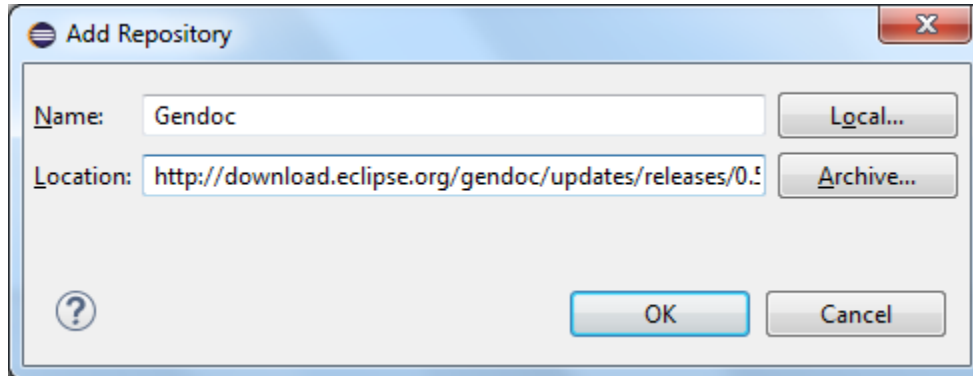


Figure 8-2: Installing Gendoc (2)

Select Gendoc:

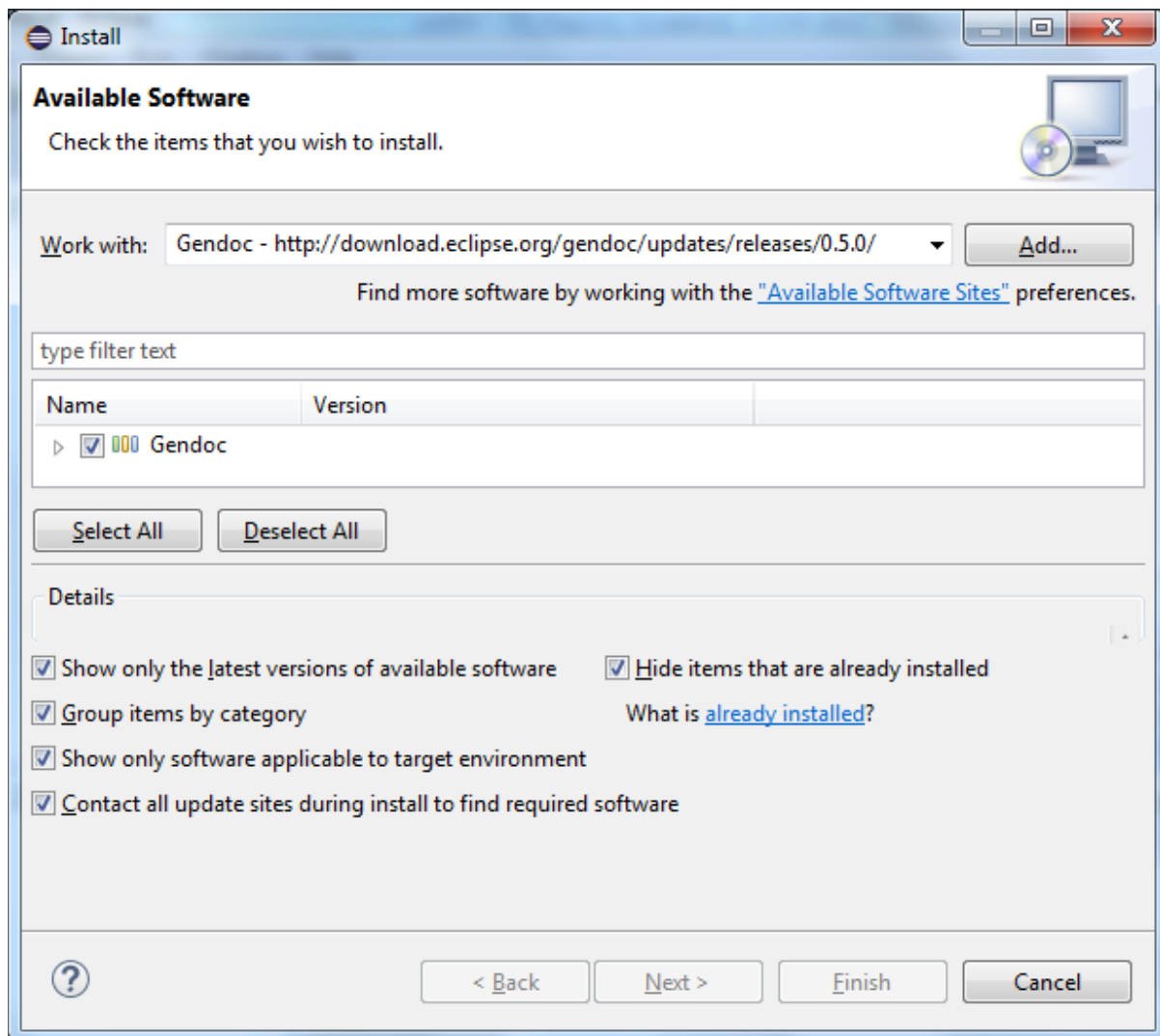


Figure 8-3: Installing Gendoc (3)

Then click  and follow the instructions.

8.3 Using Gendoc

Annex A describes how to use Gendoc.

8.4 Papyrus Table

Unlike the class diagrams which show only parts of the underlying model, a Data Dictionary contains all of the information stored in the model. Papyrus provides a function to convert the content of the model into an Excel sheet.

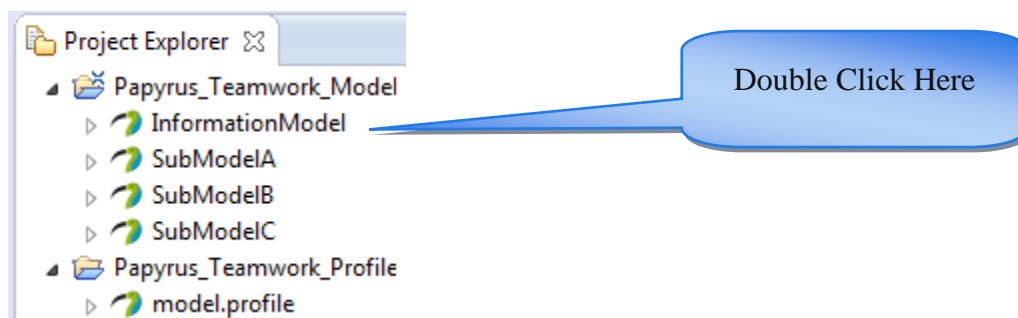


Figure 8-4: Model Selection

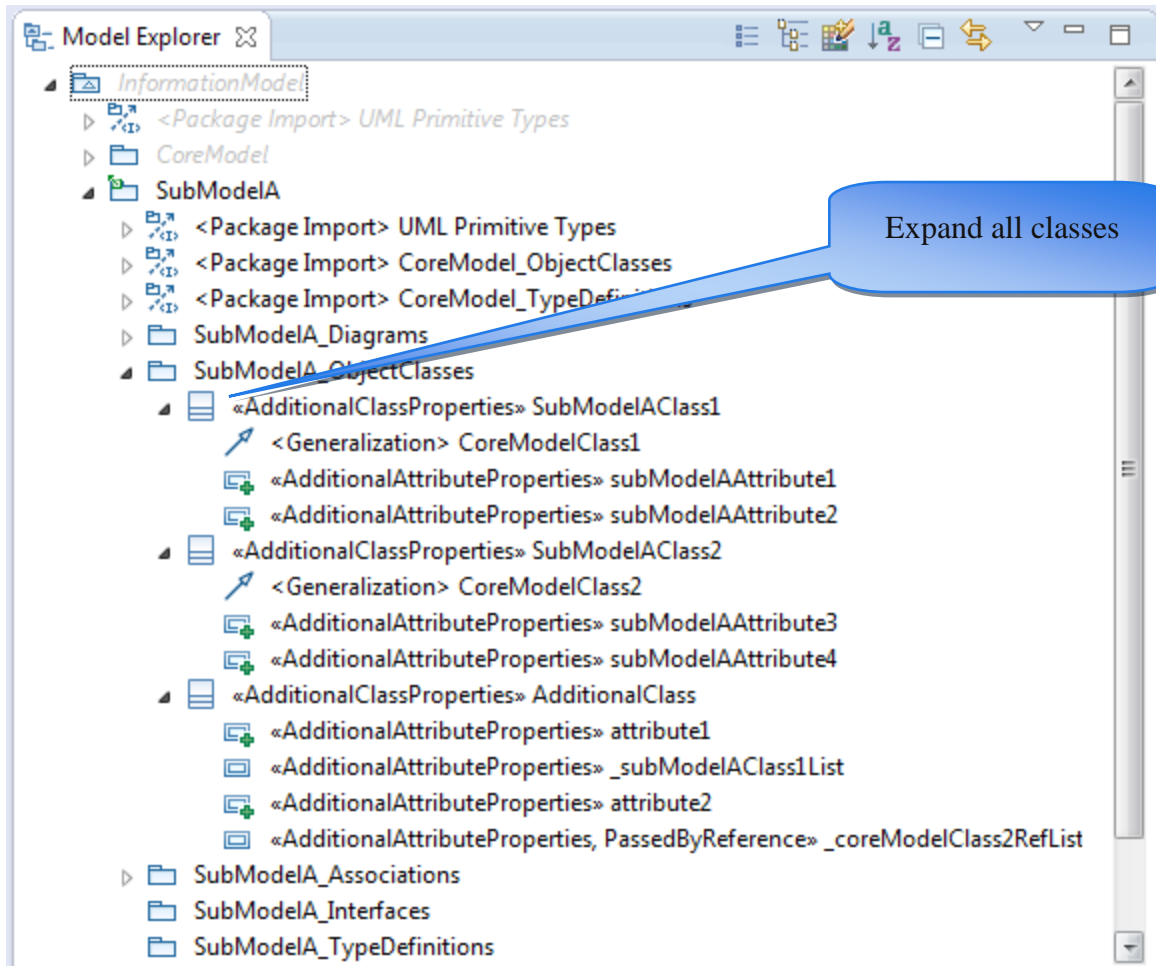


Figure 8-5: Class Expansion

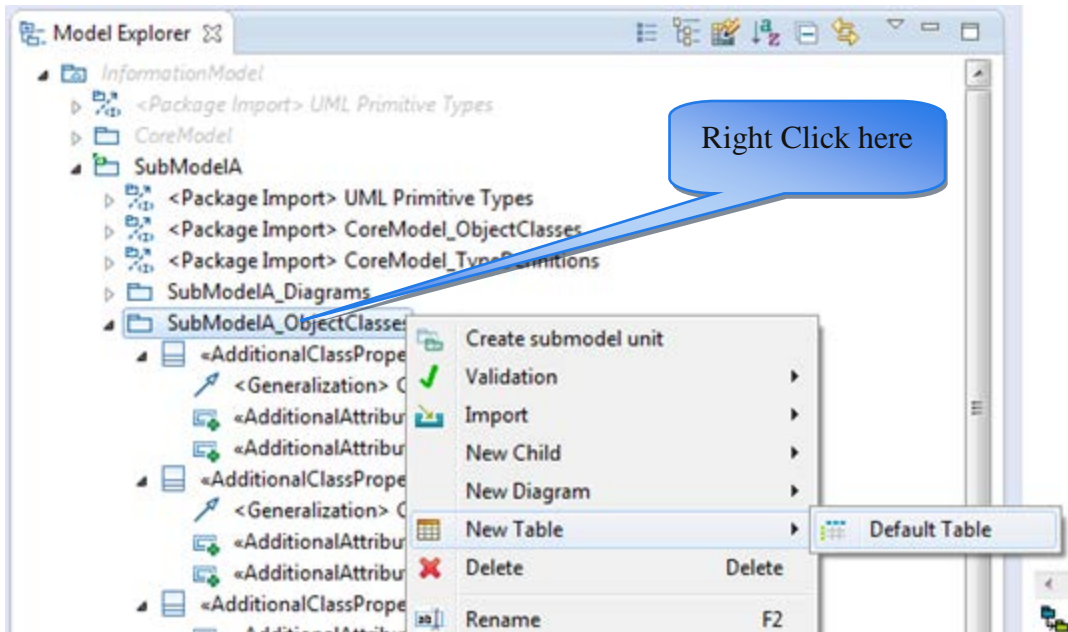


Figure 8-6: Create new empty Table

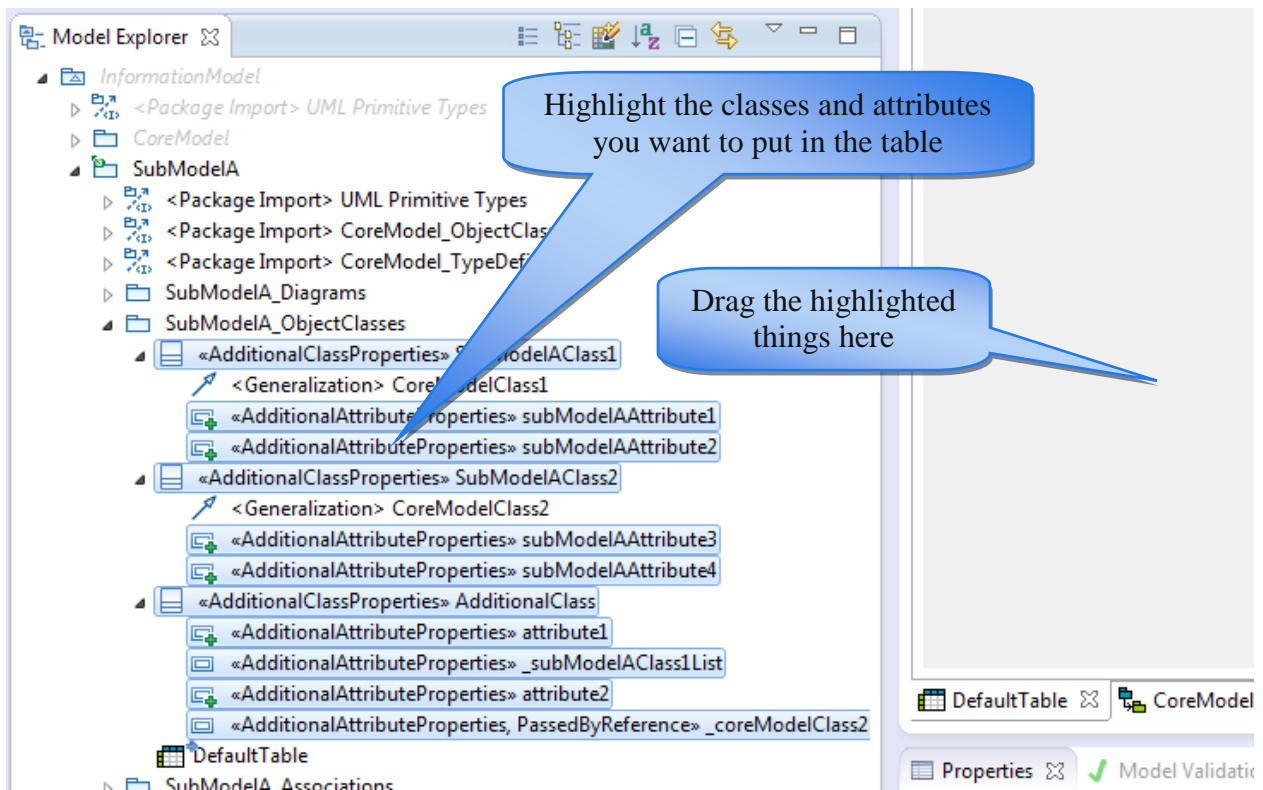


Figure 8-7: Artifact Selection

The content of the new table can be converted into an Excel sheet by selecting the required columns in the table and then copy&paste the data into an Excel sheet.

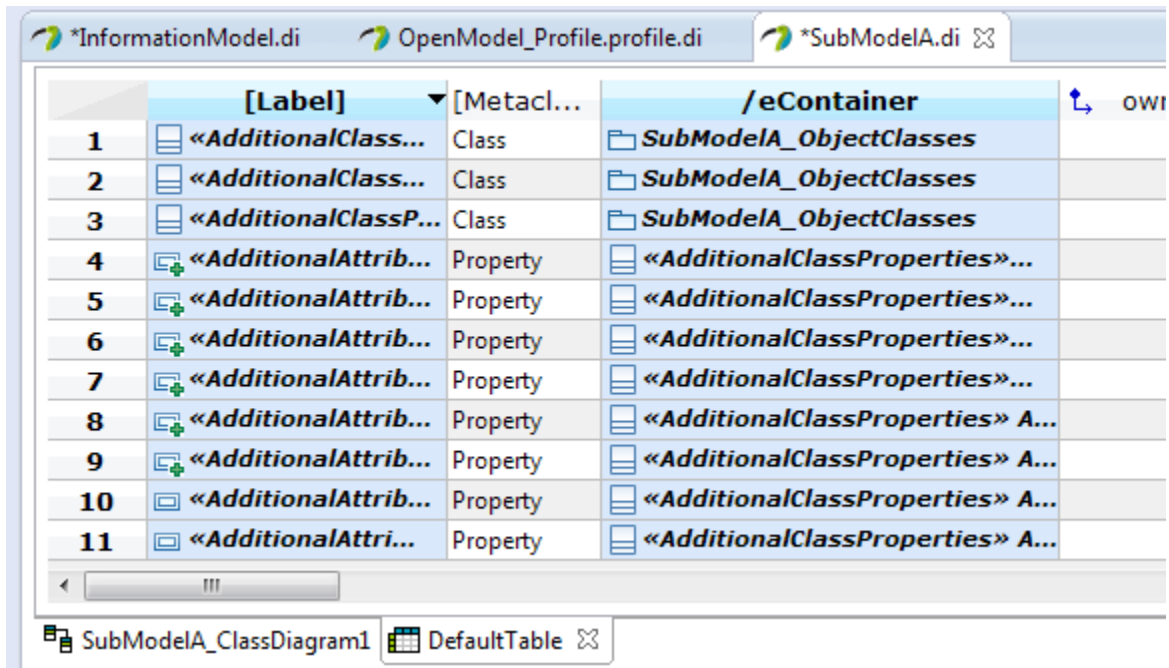


Figure 8-8: Creation of Excel Sheet (1)

| | A | B | C |
|----|--|---|--------|
| 1 | [Label] | /eContainer | ownedC |
| 2 | «AdditionalClassProperties» SubModelAClass1 | SubModelA_ObjectClasses | |
| 3 | «AdditionalAttributeProperties» subModelAAttribute1 | «AdditionalClassProperties» SubModelAClass1 | |
| 4 | «AdditionalAttributeProperties» subModelAAttribute2 | «AdditionalClassProperties» SubModelAClass1 | |
| 5 | «AdditionalClassProperties» SubModelAClass2 | SubModelA_ObjectClasses | |
| 6 | «AdditionalAttributeProperties» subModelAAttribute3 | «AdditionalClassProperties» SubModelAClass2 | |
| 7 | «AdditionalAttributeProperties» subModelAAttribute4 | «AdditionalClassProperties» SubModelAClass2 | |
| 8 | «AdditionalClassProperties» AdditionalClass | SubModelA_ObjectClasses | |
| 9 | «AdditionalAttributeProperties» attribute1 | «AdditionalClassProperties» AdditionalClass | |
| 10 | «AdditionalAttributeProperties» _subModelAClass1List | «AdditionalClassProperties» AdditionalClass | |
| 11 | «AdditionalAttributeProperties» attribute2 | «AdditionalClassProperties» AdditionalClass | |
| 12 | «AdditionalAttributeProperties, PassedByReference» _core | «AdditionalClassProperties» AdditionalClass | |
| 13 | | | |

Figure 8-9: Creation of Excel Sheet (2)

9 Importing RSA Models into Papyrus

This section 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

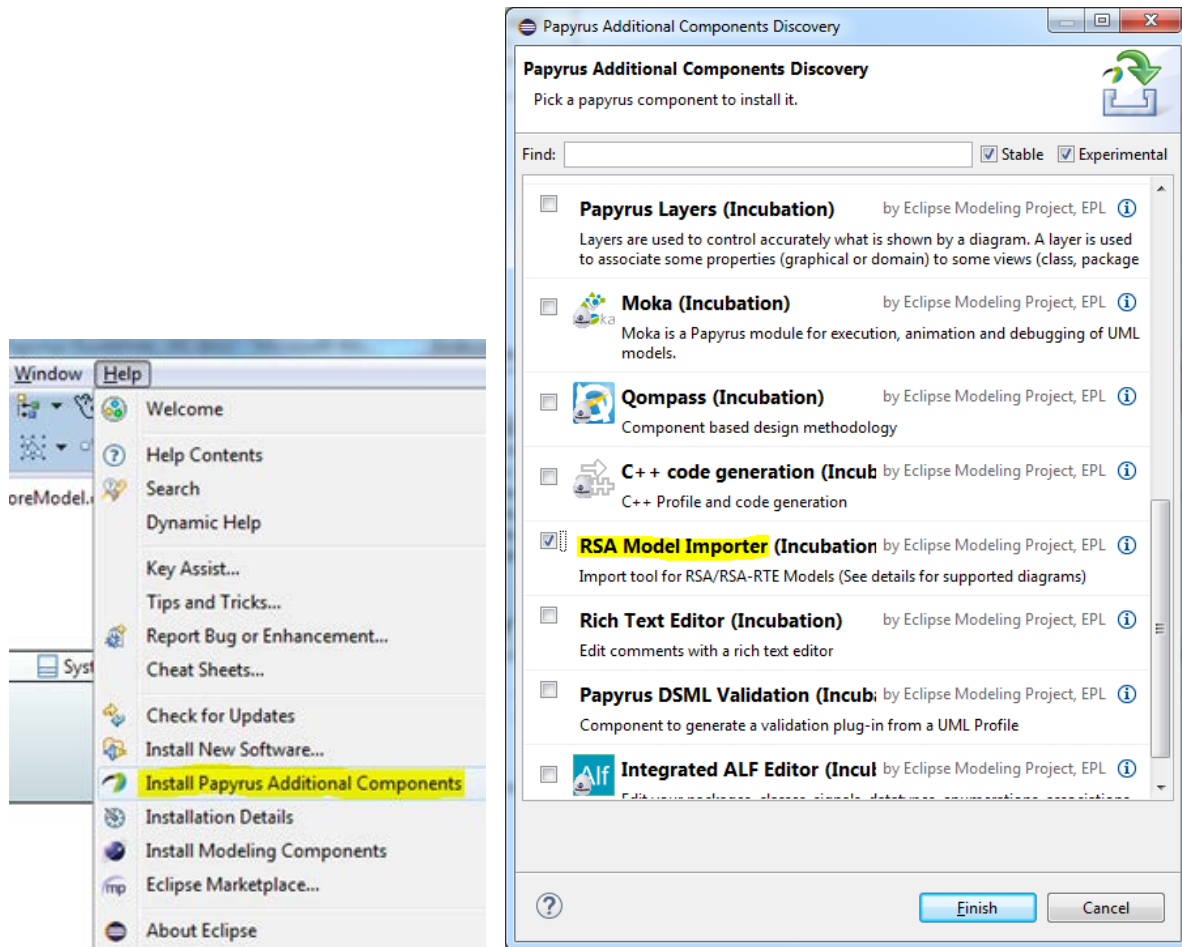


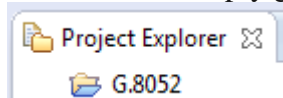
Figure 9-1: Installing Papyrus Component “RSA Model Importer”

9.1 Import RSA Model into Papyrus

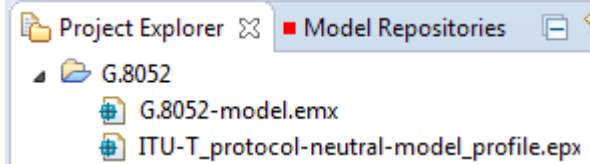
Notes: Each step identifies the tool that is used to execute it. Any ASCII editor can be used instead of Notepad++.

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).



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



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

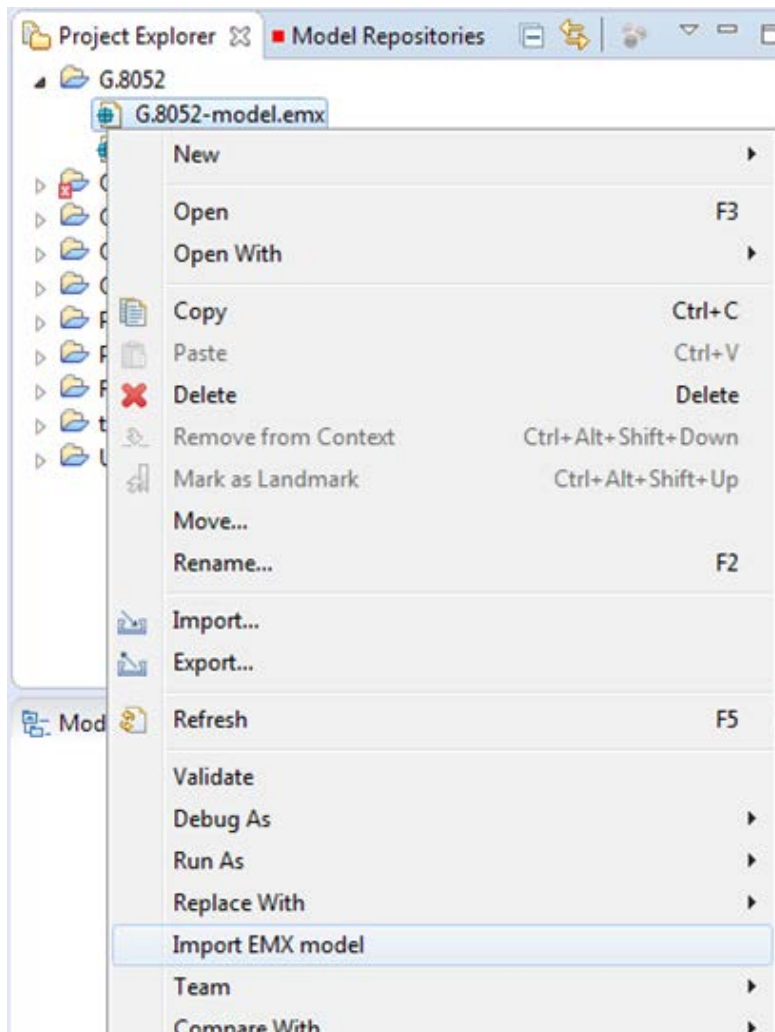
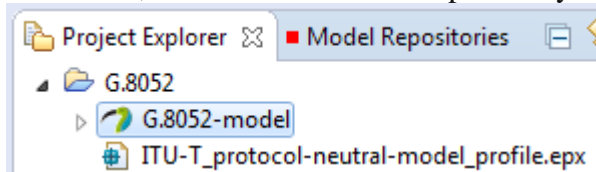


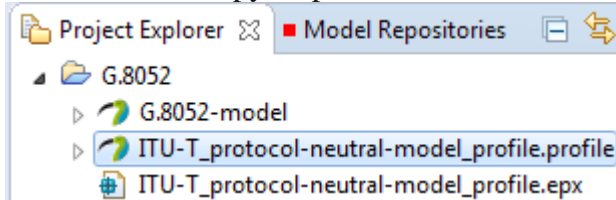
Figure 9-2: : Importing .emx Model

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:



9.2 Replace RSA Profile by Papyrus Profile

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

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

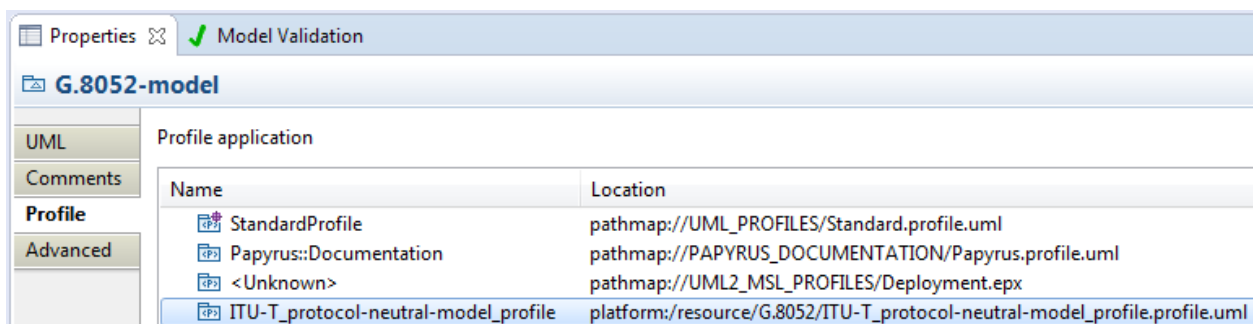
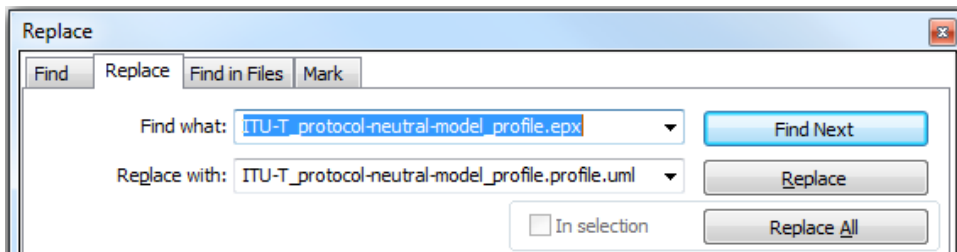




Figure 9-3: Associated Papyrus Profile

9.3 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.

Annex A Using Gendoc

Editor's notes:

This section is still a draft and will likely be changed in future versions.

Please check the known issues in section A.15 for any limitation which exists at the time.

A basic document generation tutorial is available at https://www.eclipse.org/gendoc/documentation/Gendoc_v0.5_tutorial.pdf. This provides detail in some areas but does not cover all aspects of usage. The following subsections 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 a mix of Gendoc script, normal text, Word figures, tables etc. Thus, if you want to create a word document with other non-model related information, but then have a section specifically for the model, you would insert the “gendoc” related information as part of that Word document. The following section 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 section

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 section for formal description and structure

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

A.1 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.

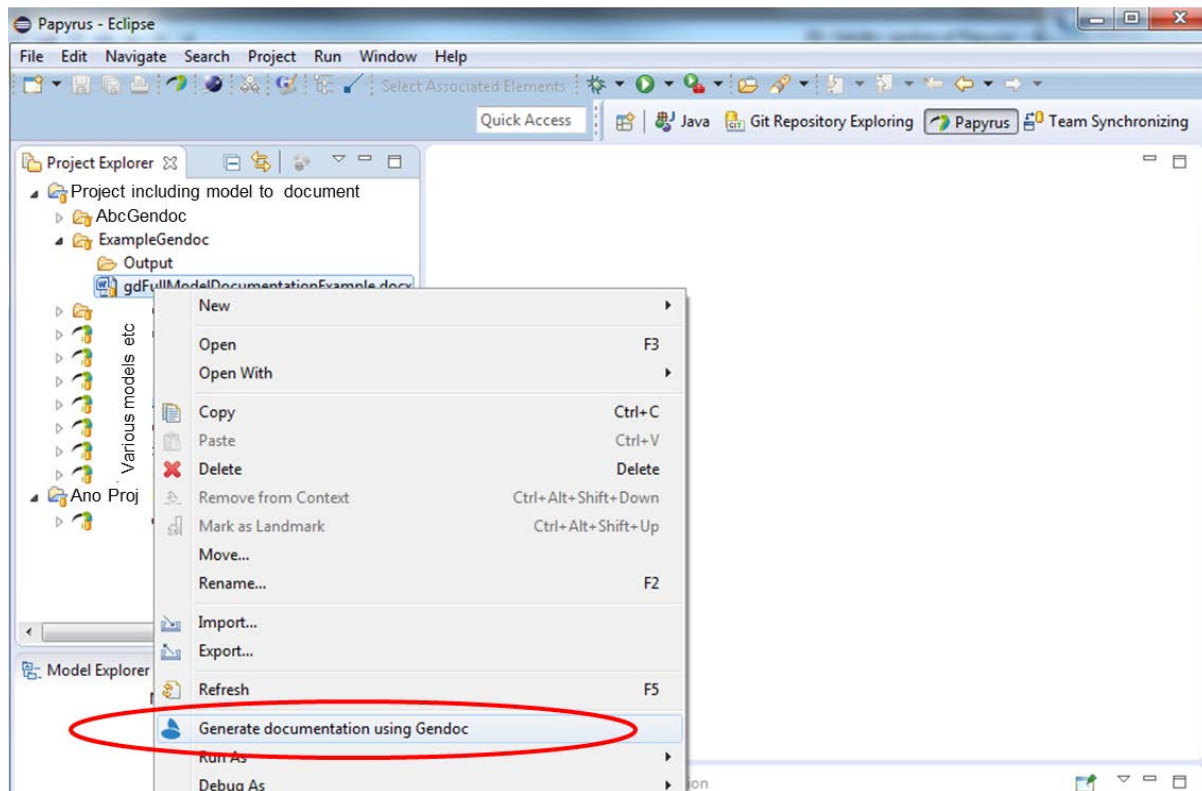


Figure A-1: Initiating Gendoc for a particular template

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

A.2 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}'`”.

A.3 Cover, contents, closing text etc

Any text and figures⁴ inserted between the `<gendoc>` to `</gendoc>` space will be produced in the output.

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

A.4 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.

A.5 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/>
```



A.6 Further explanation of the script

A majority of the script used in the previous sections 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/>. You may find many blank lines in your output. This will normally be because you have forgotten one or more <drop/> instructions.

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.

A.7 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.



A.8 Data Dictionary template overview

In the following subsections 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 [5].

A.9 Adding the class and its stereotypes

Considering the basic template described above and replacing the text “Provides access to class details” with the section 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

A.10 Adding properties and stereotypes in tabular form

The script from the previous section 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 |
|------------------|-----------------------|--|---|--|--|
| <p>[p.name/]</p> | <p>[p.type.name/]</p> | <p>[if(p.lower=p.upper)]1[else][p.lower/][if(p.upper=-1)]*[else][p.upper/][if]</p> | <p>[if(not(p.isReadOnly))]RW[else]R[if]</p> | <p>[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/> [/if]<drop/> [/for]<drop/> [/for]<drop/></p> | <p>[for (c:Comment p.ownedComment)]<drop/> [c._body.clean()/] [/for]</p> |

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

```
</gendoc><drop/>
```

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

A.11 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...

A.12 Adding other data types

A.12.1 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 sections.

```

[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/>

```

A.12.2 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/>

```

A.13 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 sections listing classes, complex data types, enumerations and primitive data types.



gdFullModelDocumen
tationExample.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.

A.14 Extending the template

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

A.15 Known issues

Known issues:

- Some header/footer content may prevent document production
- “[“ (often used in references) must not be used in the Word text anywhere
- HTML formatting must not be used in a comment/description field for any artefact in a UML model