OSETS

Open source ECOA Toolsets

Usage Manual

Version History

Date	Version	Description	Author
15-09-2017	1.0	Initial Release	Venu Madhav P
20-11-2017	2.0	Corrections to address customer comments.	Stuart Palin

References

S.No	Document Name	Document Version

Distribution

Name	Company	Designation

Contents

1 Introduction 5

1.1 Disclaimer 5

1.2 Scope 5

1.3 Pre-requisites 6

1.4 Installation 6

2 Project Setup 7

2.1 Initiation of Wizards 10

3 Tree Editors 11

3.1 Types Editor 11

3.1.1 Scenario 11

3.1.2 Other Definitions 20

3.2 Services Editor 28

3.2.1 Scenario 28

3.2.2 Other Definitions 33

3.3 Component Definition Editor 36

3.3.1 Scenario 37

4 Graphical Editors 47

4.1 Initial Assembly Editor 47

4.1.1 Scenario 47

4.2 Component Implementation Editor 54

4.2.1 Scenario 55

4.2.2 Other Definitions 67

4.3 Final Assembly Editor 68

4.3.1 Scenario 68

4.4 Logical Systems Editor 72

4.4.1 Scenario 72

4.5 Deployment Editor 76

4.5.1 Scenario 76

4.5.2 Other Definitions 81

5 Additional Features 84

5.1 Edit and Remove on Tree Editors 84

5.1.1 Edit 84

5.1.2 Remove 85
5.2 Generation of XML 87
5.3 Generation of API Codegen 89
5.4 Generation of Platform Codegen 89
5.5 Export as Image 90
5.6 Refresh of Editors 92
5.6.1 Refresh of Component Implementation from Definition 92
5.6.2 Refresh of Final Assembly from Initial Assembly 93
5.6.3 Refresh of Deployment from Logical System and Final Assembly 95
6 Epilogue 97
6.1 Identifying Toolbar Options 97
Appendix A Known Issues 98
A.1 Types Editor 98
A.2 Service Editor 98
A.3 Component Definition Editor 98
A.4 Initial Assembly Editor 98
A.5 Component Implementation Editor 98
A.6 Final Assembly Editor 99
A.7 Logical System Editor 99
A.8 Deployment Editor 99
A.9 Code Generation 99
A.10 General 100
Appendix B Dining Philosophers Example Script 101
B.1 Data Libraries 101
B.2 Services 101
B.3 Component Definition 101
B.4 Initial Assembly 101
B.5 Component Implementation 101
B.6 Logical System 102
B.7 Final Assembly & Deployment 102
B.8 ECOA XML and Code Generation 103
Appendix C ECOA XML Validation with Eclipse 104

1 Introduction

OSETS – Open Source ECOA Toolset is an Eclipse Neon GEF based plugin used to design and implement experimental systems using the ECOA approach. The ECOA Components below are implemented using GEF Editors:

- Tree Editors
  - Type Editor
  - Service Editor
  - Component Definition Editor
- Graphical Editors
  - Initial Assembly Editor
  - Component Implementation Editor
  - Final Assembly Editor
  - Logical Systems Editor
  - Deployment Editor

1.1 Disclaimer

OSETS supports ECOA Architecture Specification version 5 (http://www.ecoa.technology/) with the following exceptions:

- Hierarchical Data Libraries,
- Quality of Service,
- PINFO.

OSETS is provided "as is", without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and noninfringement. In no event shall the authors or copyright holders be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the software or the use or other dealings in the software.

OSETS and its outputs are not claimed to be fit or safe for any purpose. Any user should satisfy themselves that this software or its outputs are appropriate for its intended purpose.

1.2 Scope

The scope of this document is to demonstrate the usage of the plug-in. Each of the above editors will be explained using the Dining Philosophers problem.

NOTE: To assist with re-creating the example a text script of the steps is given in Appendix B.

1.3 Pre-requisites

The below are the expected pre-requisites for usage of the toolsets

1) Minimal understanding of ECOA Concepts
2) Eclipse Neon for C/C++ Development

1.4 Installation

The below are the installation steps:

1) Download Eclipse for C/C++ development
2) Copy the release plugin file[1] osets-eclipse-plugin_<version>.jar into <ECLIPSE_HOME>/plugins directory
3) Start eclipse by running **<ECLIPSE_HOME>/eclipse –clean**
4) Observe and verify that the OSETS Toolbar Actions are visible

Further instructions for incorporating XML validation against the ECOA XML schema are provided in Appendix C.

Addition to footnote 1: Generating the plug-in requires the **Eclipse Plug-in Development Environment**, and the **Eclipse GEF Application Development Framework**, both of which are available using **Help | Install New Software** within Eclipse.

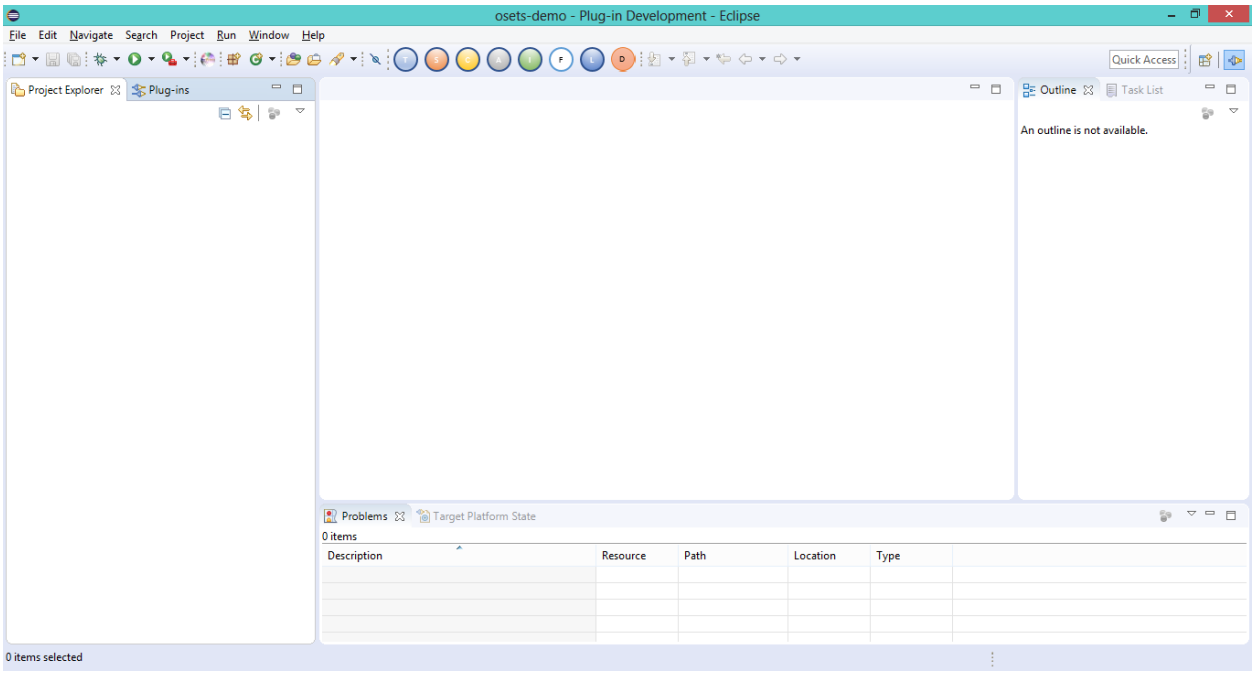
[1] The OSETS source is configured as an Eclipse Plug-In Development Project which can be used to generate the OSETS plug-in. If starting from the OSETS source project, contained within a zip file, the plug-in is generated by the following steps:

a) Start Eclipse from launcher (or terminal window) and select normal workspace;
b) If necessary delete pre-existing versions of the OSETS project:
   i) Right click project and select **Delete** | **Delete project contents on disk** | **OK**
   ii) Using a host file-manager delete the folder <*name*>.zip_expanded (<*name*> is the name of the original OSETS imported zip file) from the Eclipse workspace area.
   *NOTE: Leave the **.metadata** file - this keeps previous project work using OSETS*
c) From Eclipse toolbar (top right) select File | **Open Projects from File System** | **Archive**
d) Select OSETS zip file | **OK**
e) Deselect <*name*>.zip_expanded (leaving <*name*>.zip_expanded/oset-eclipse-plugin), and click **Finish**
f) Select **Window**| **Perspective** | **Open Perspective** | **Other** | **Plug-In Development** | **OK**
g) Right click on **osets-eclipse-plugin** project;
h) Select **Export**;
i) Select **Deployable plug-ins and fragments** and **Next**;
j) Ensure **osets-eclipse-plugin** is selected in the **Available Plug-Ins and Fragments**;
k) Select a Directory to receive the plug-in and **Finish**;
l) At the end of the generation process the Directory will hold a file **osets-eclipse-plugin_#.#.#.jar** (#.#.# indicates the plug-in version) within a plugins directory.

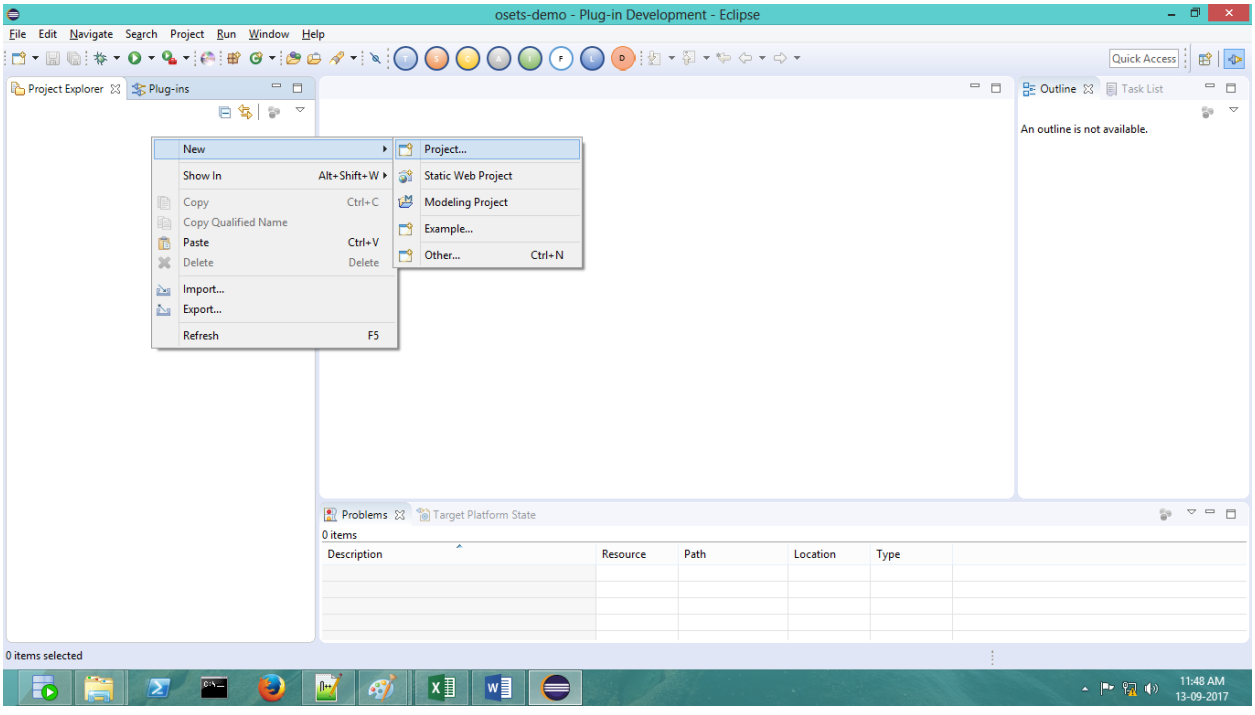
2 Project Setup

Once eclipse is open post installation - Ref 3), follow the below steps to create the empty project setup:

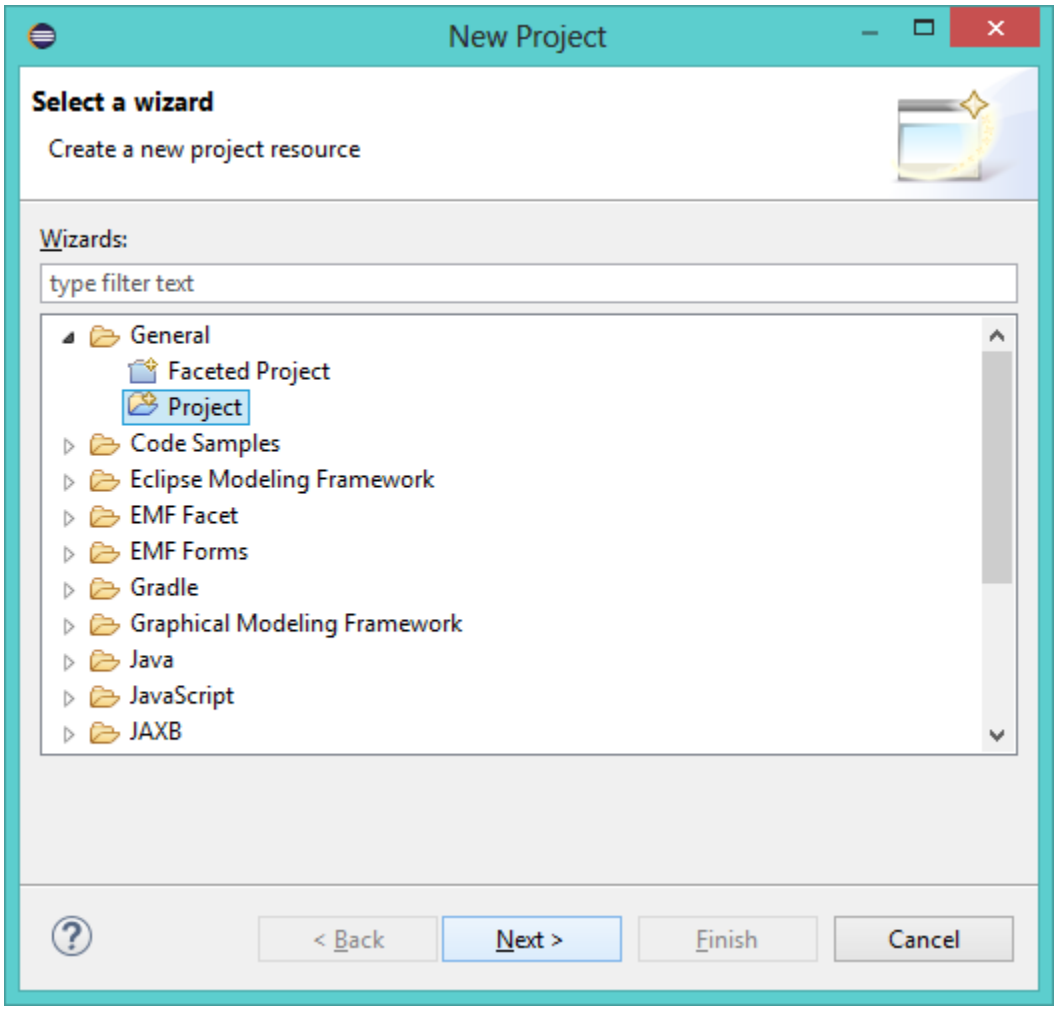
1) The initial page will look something like below:



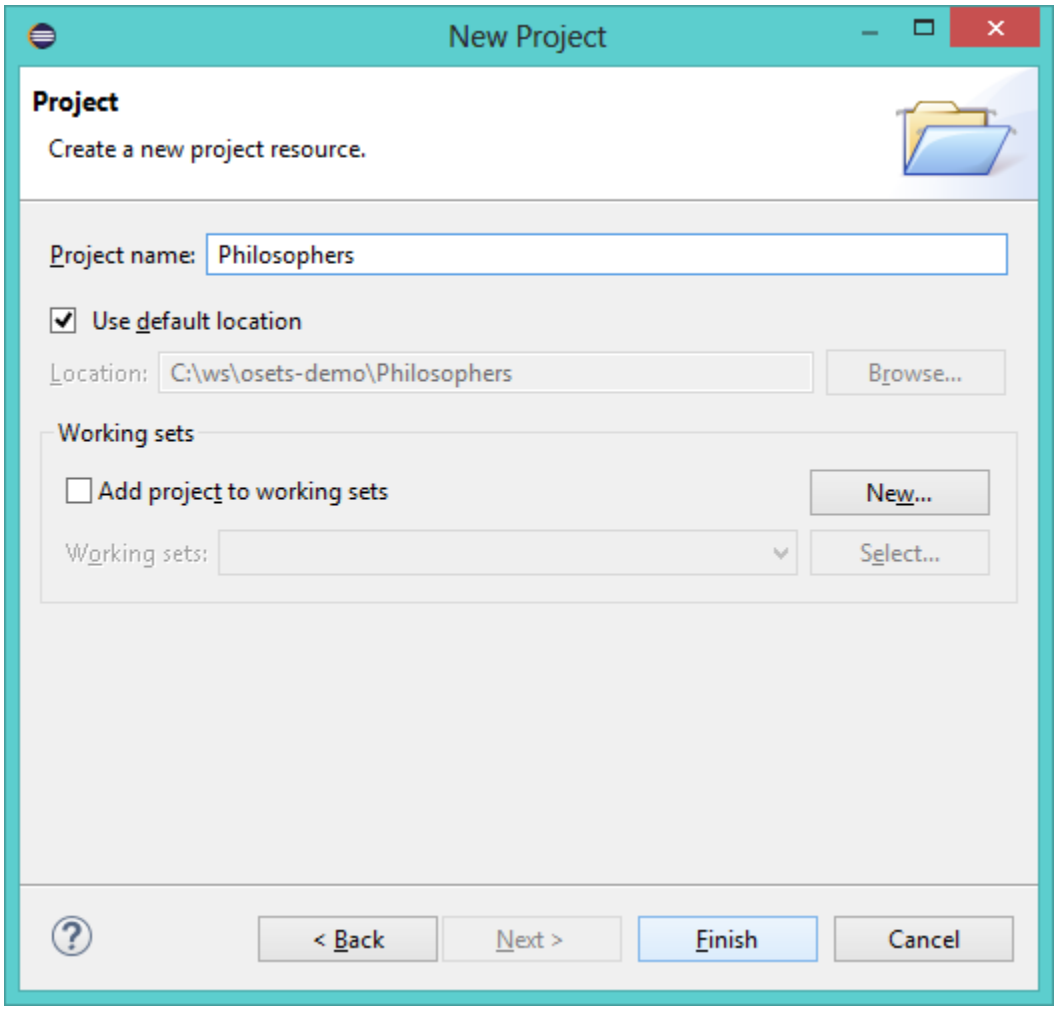
2) Right click on the project explorer and select New -> Project



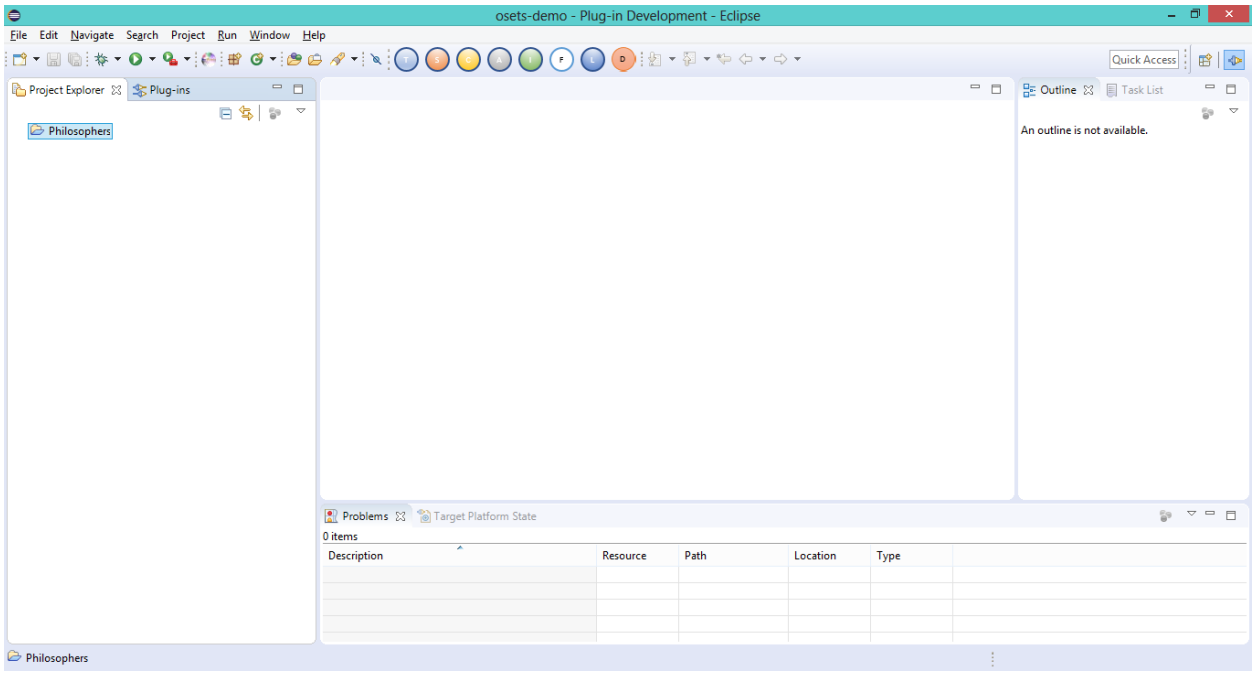
3) Select General -> Project to create a generic eclipse project



4) On Next screen, give Project Name as Philosophers and select Finish



5) This will create an empty eclipse project and the final screen will be as below



Before starting next sections please refer to section 6.1 to familiarize yourself with the Toolbar actions

2.1 Initiation of Wizards

The ECOA Construct Definition Wizards can be initiated by either:

1) Selecting the option from toolbar
2) Using the New Wizard by right clicking on the project

3 Tree Editors

This section explains about GEF Tree editors used for Types, Services and Component Definition ECOA Constructs. We have used Tree Editors for these three as their structure follows a basic Tree with Each node representing a type, service or a component definition. The fact that there are no direct links between individual definitions also enables us to move away from graphical editors.

3.1 Types Editor

ECOA has 7 Type definitions. One or Many of them form a Type Library. The idea was to create a Single file for each of the Libraries. The 7 Type Definitions are:

1) Simple
2) Enum
3) Constant
4) Array
5) Record
6) Fixed Array
7) Variant Record

3.1.1 Scenario

The Dining Philosophers problem, we have two Type Libraries. This cover

1) Simple
2) Enum

The Libraries are constructed as below:

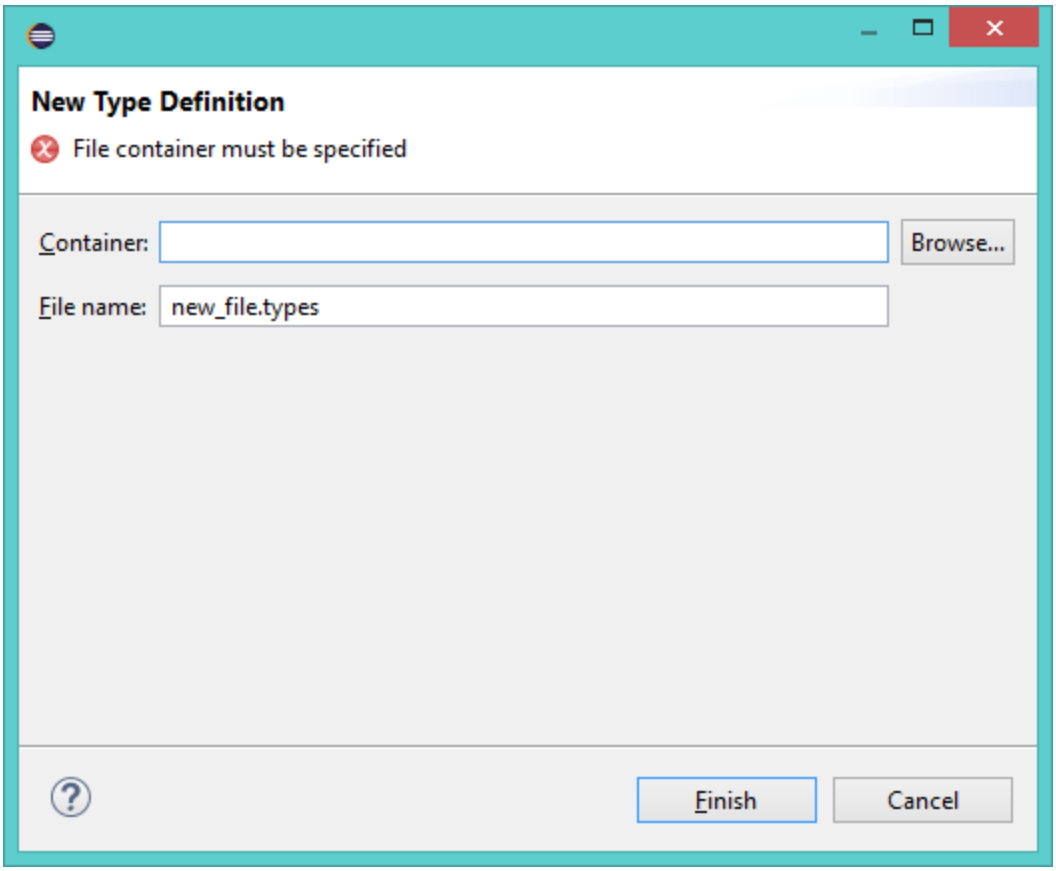
3.1.1.1 Dining Library

The Dining Library has two types:

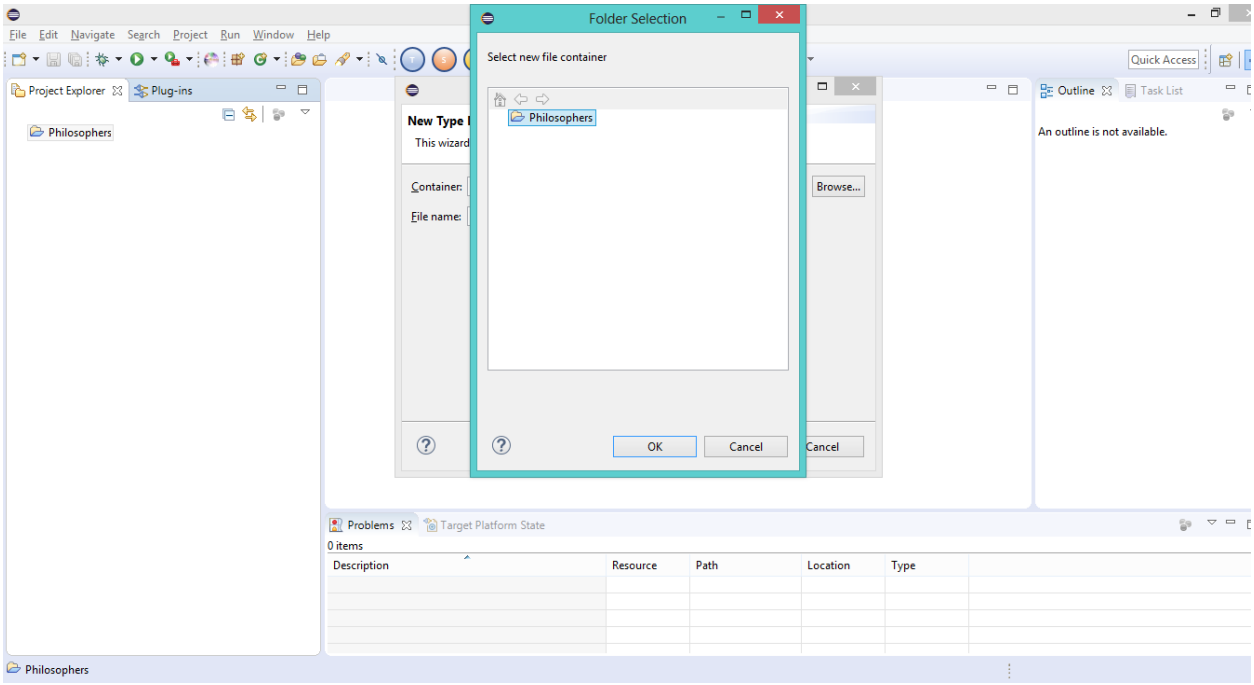
- Philosopher_id: Simple Type with int32 base type, 0-8 Range, Units of Philosopher, precision 1
- Chopstick_id: Simple Type with int32 base type, 0-7 Range, Units of Chopstick

3.1.1.1.1 Process

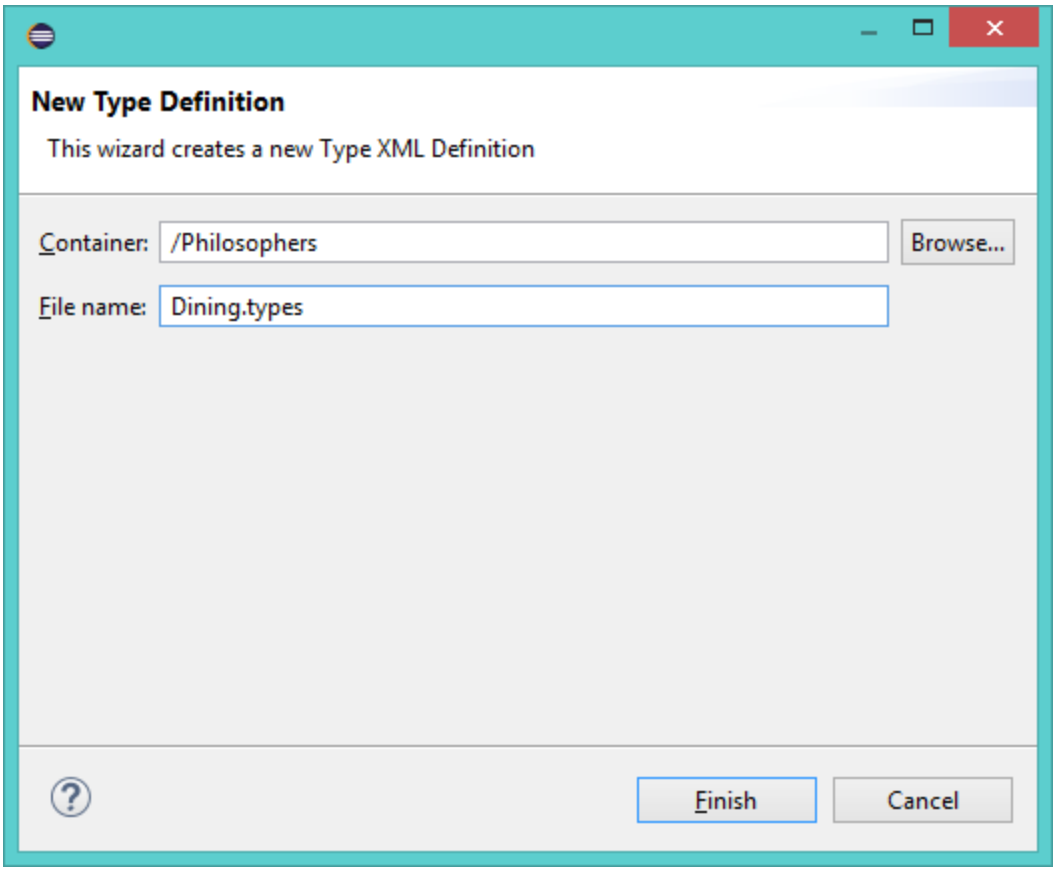
Select Type Editor from Tool bar which opens the below wizard.



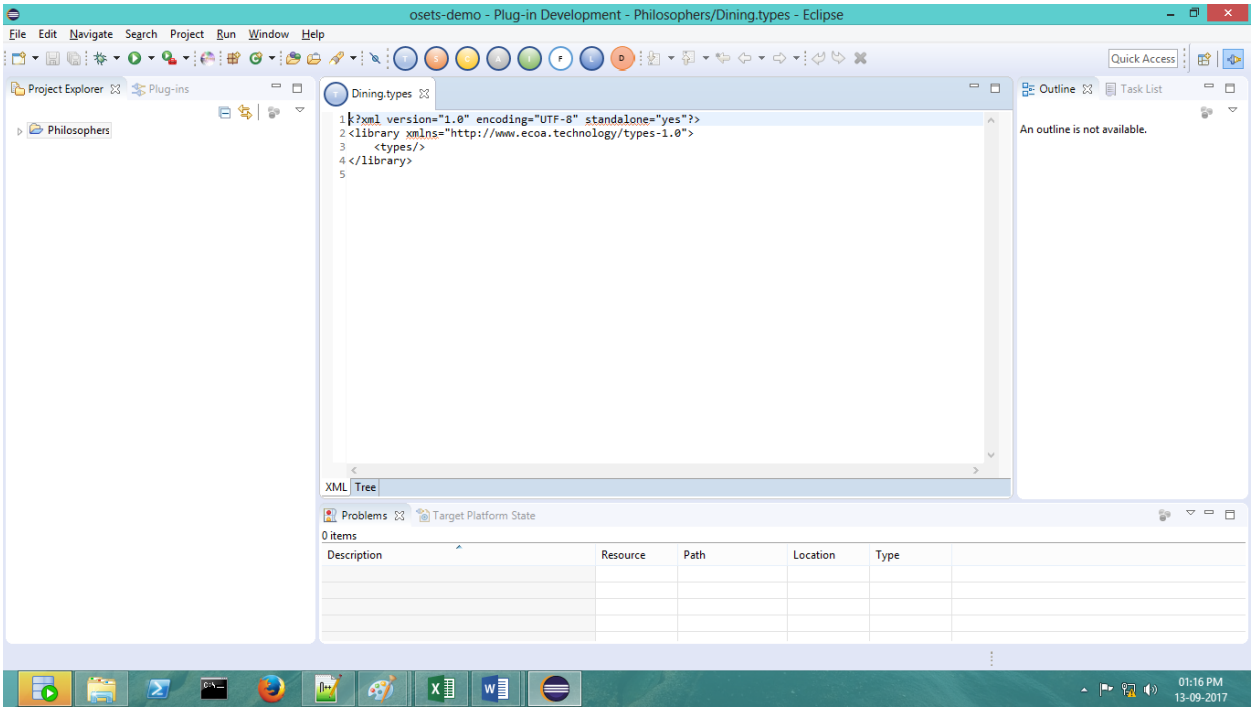
Select the container name as the top-level Project folder Philosophers



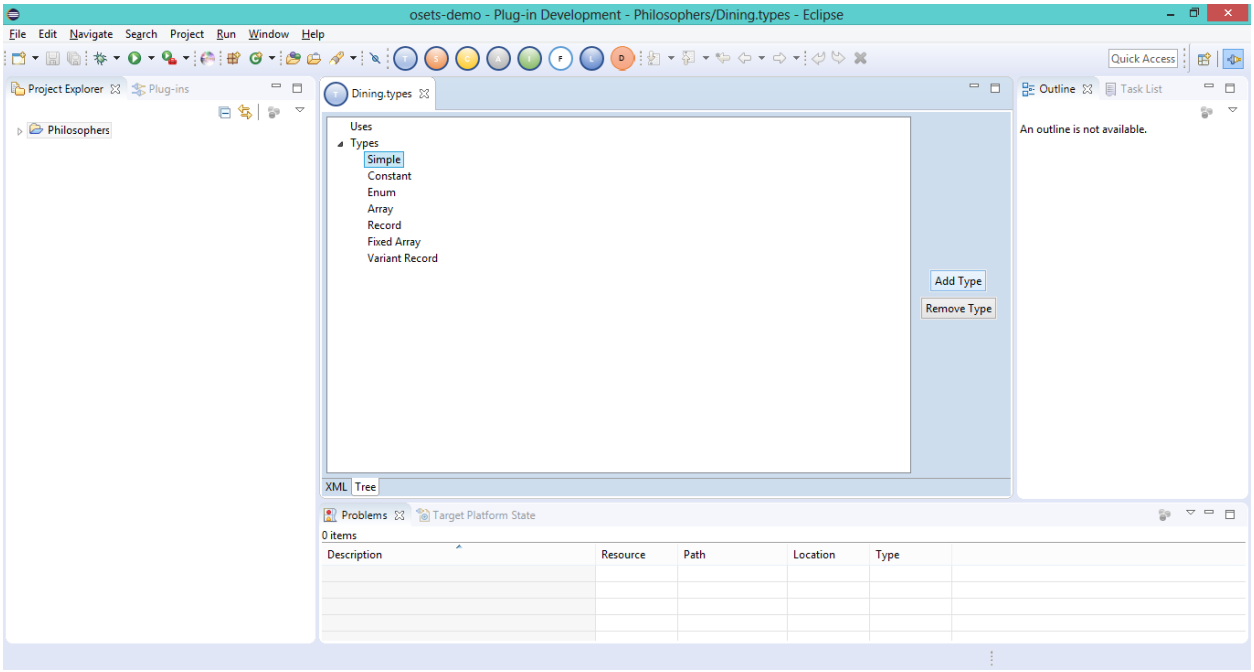
Change the file name to Dining.types



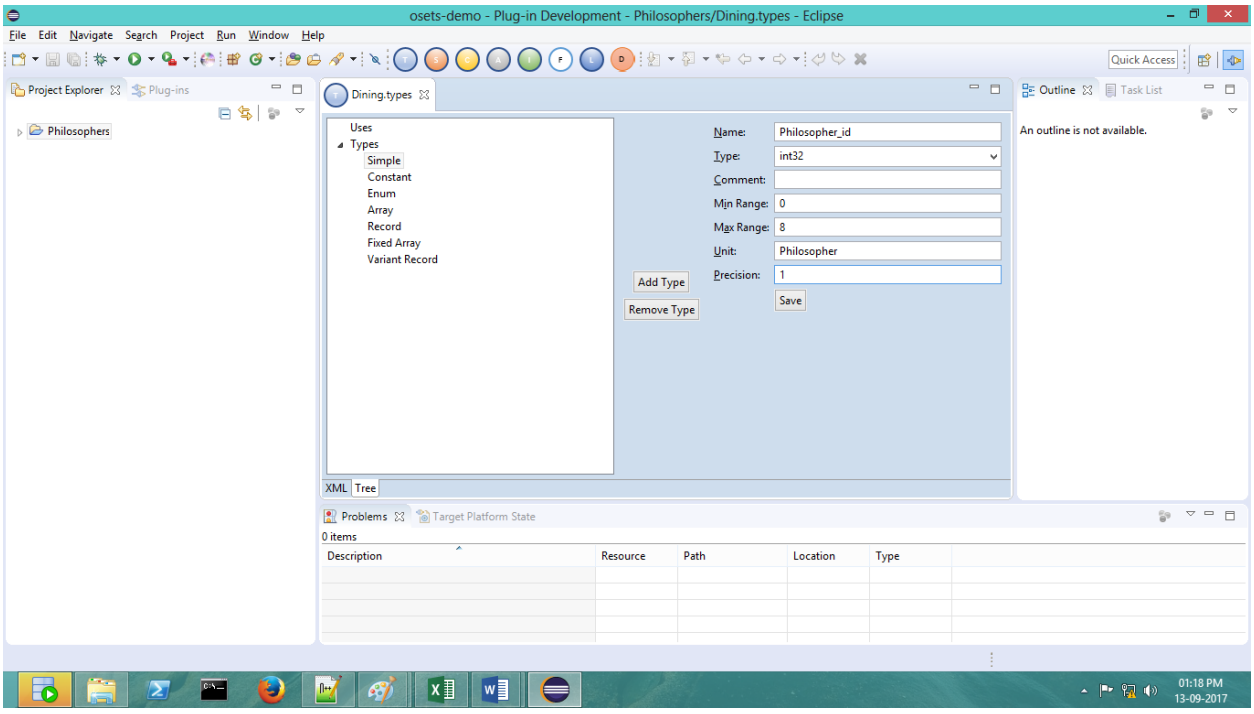
The initial XML Definition is shown



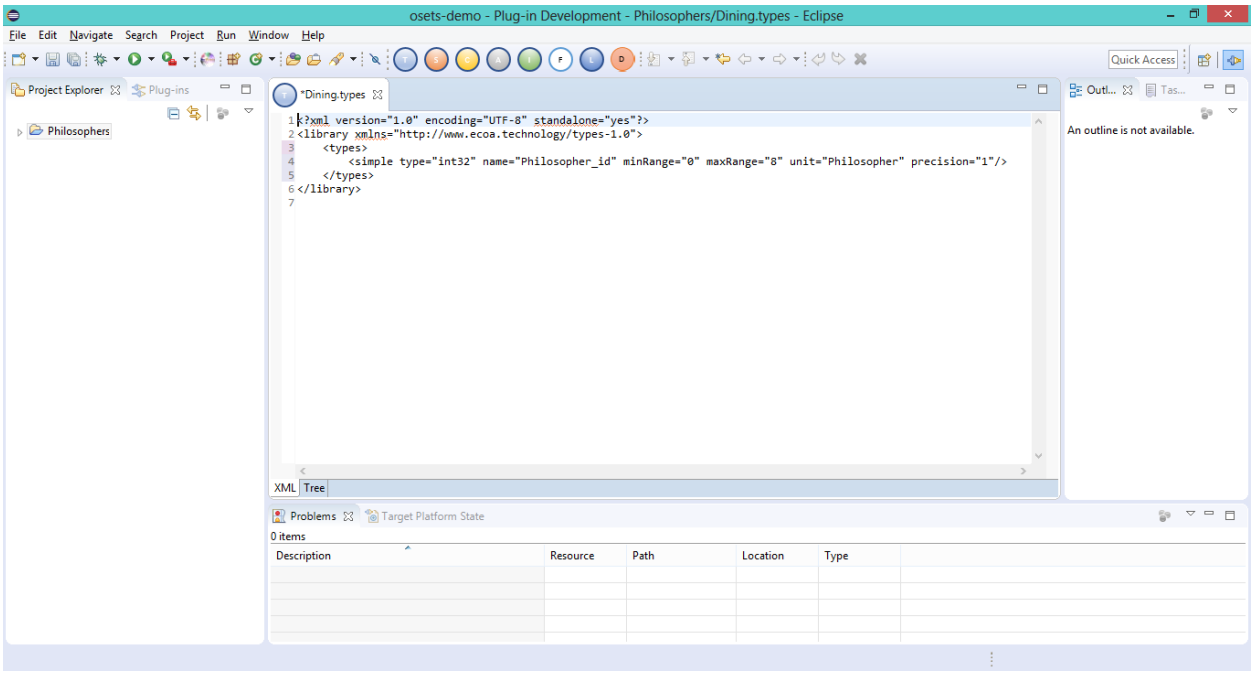
Select the tree tab at the bottom to move to Tree Editor to add the new types. Select Simple and press Add Type



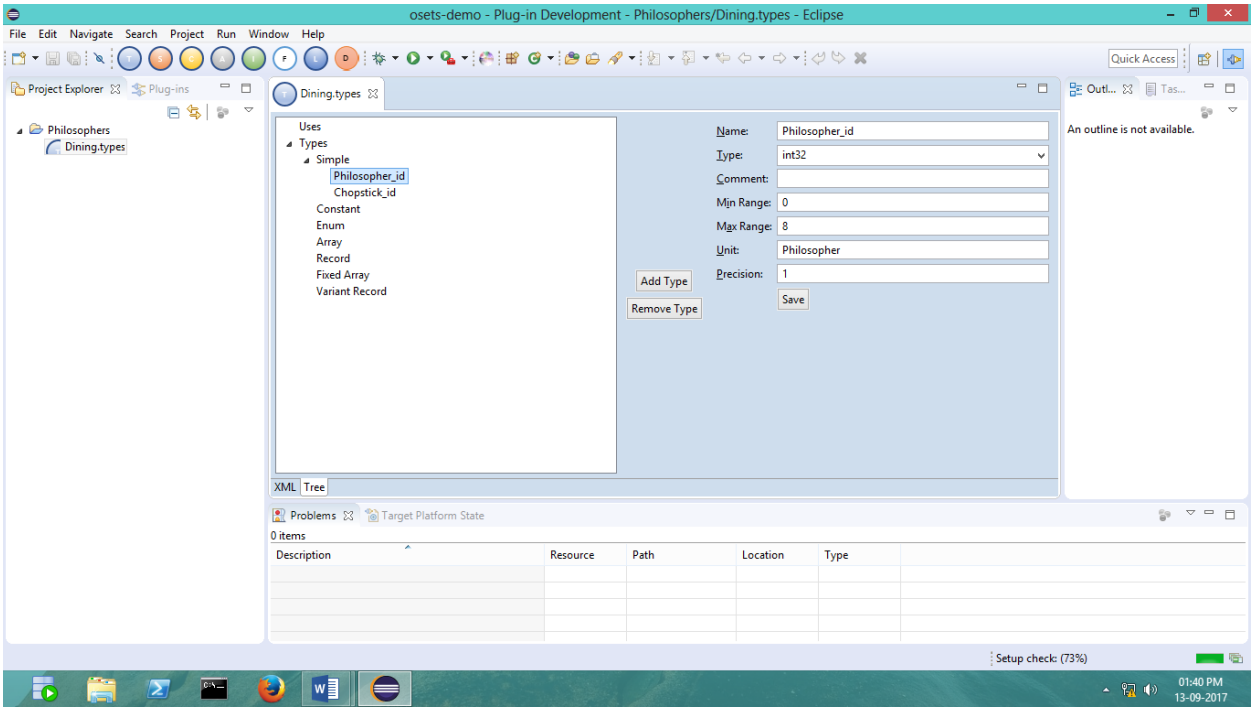
Opens the simple types Editor with the fields. Enter the details and press Save

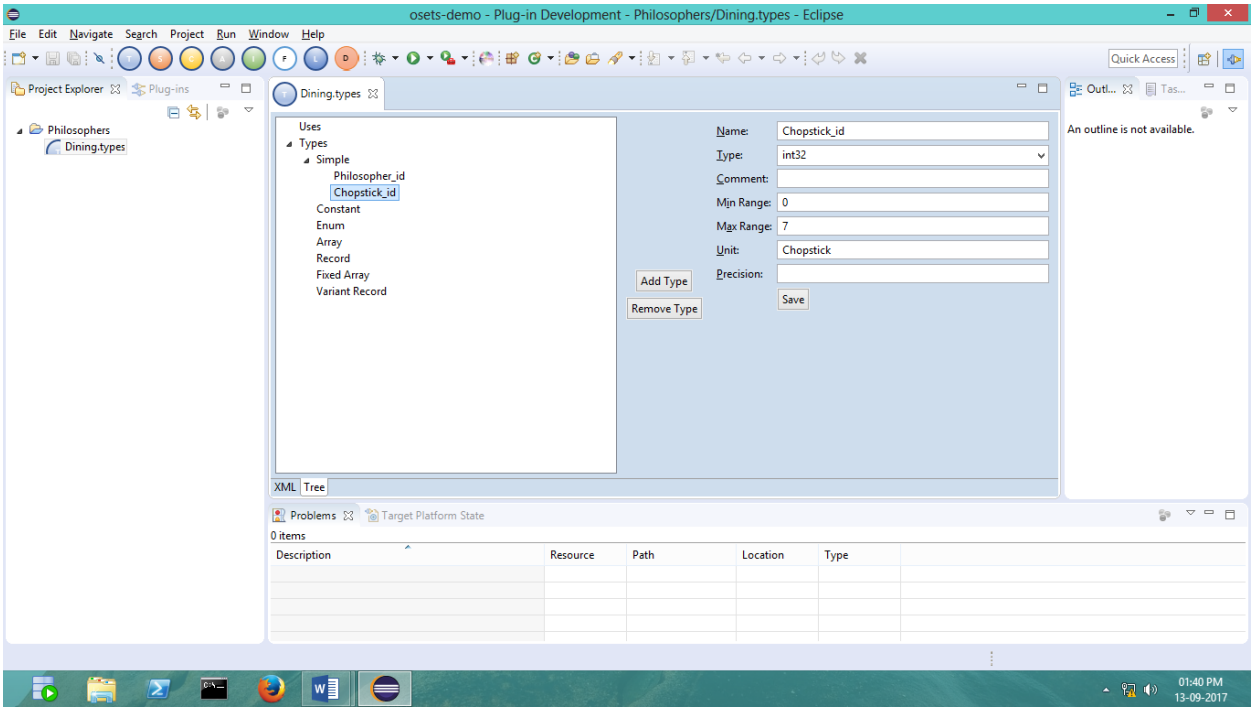


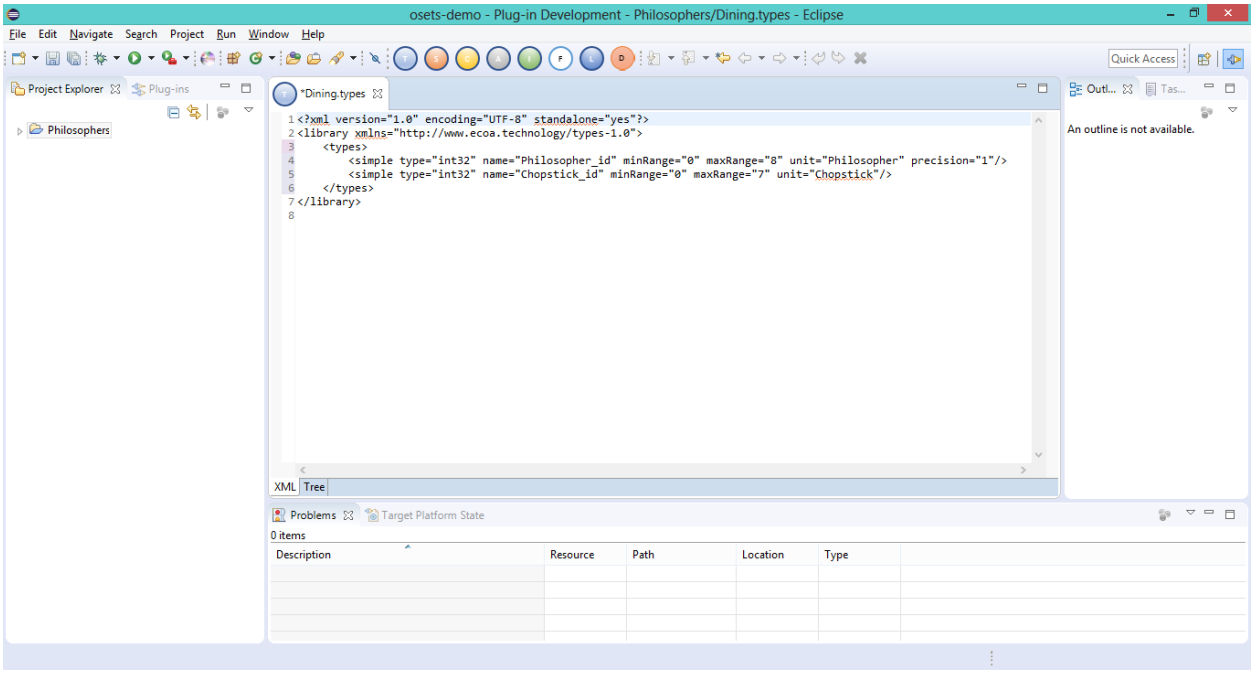
Verify the generated XML



Repeat the steps for Chopstick_id. The resultant Editor and XML Views are as below:







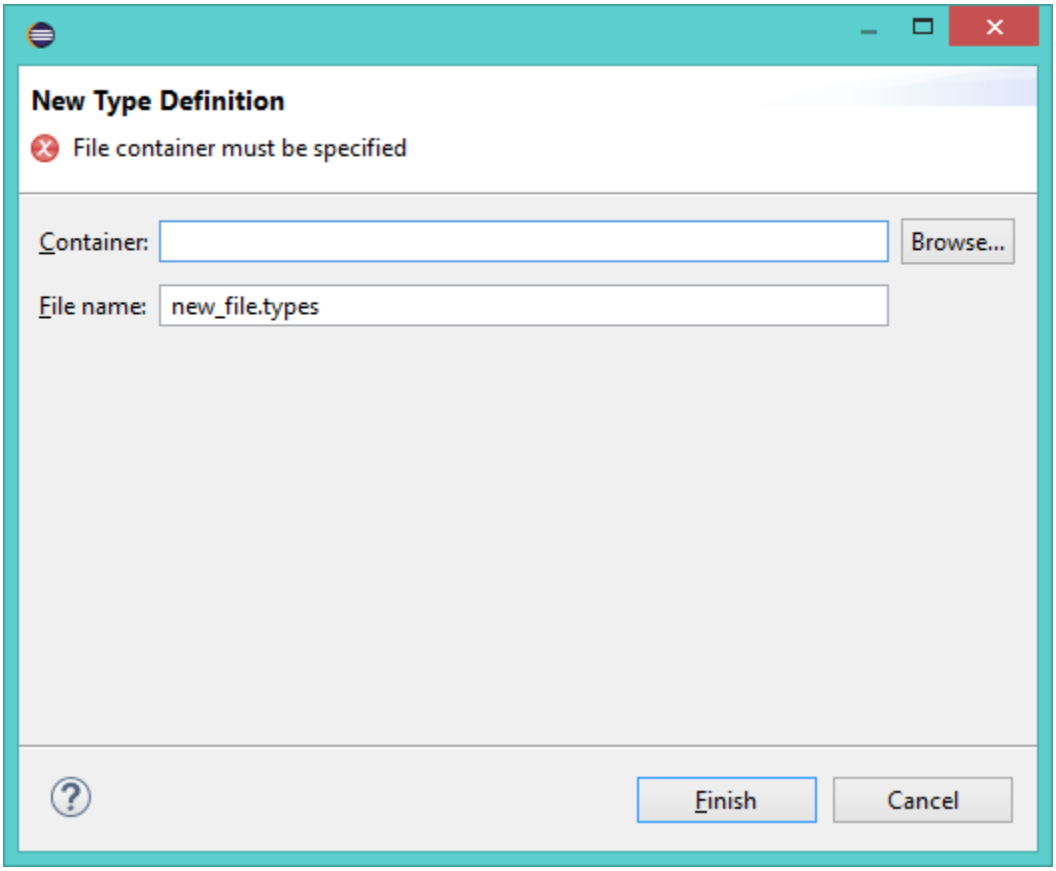
3.1.1.2 Philosopher Library

The Philosopher Library has one type

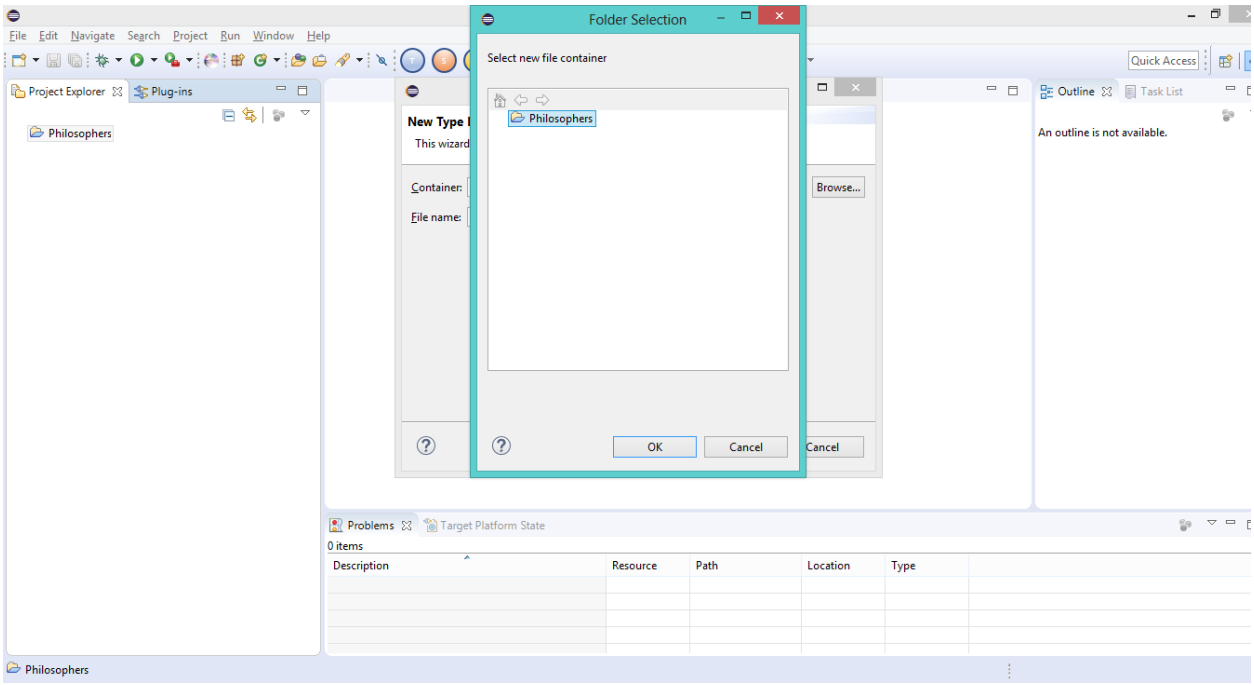
- State: Enum Type with unit8 base type, possible enum values (UNDEFINED, GETTINGSTICKS, EATING, SURRENDERING, THINKING)

3.1.1.2.1 Process

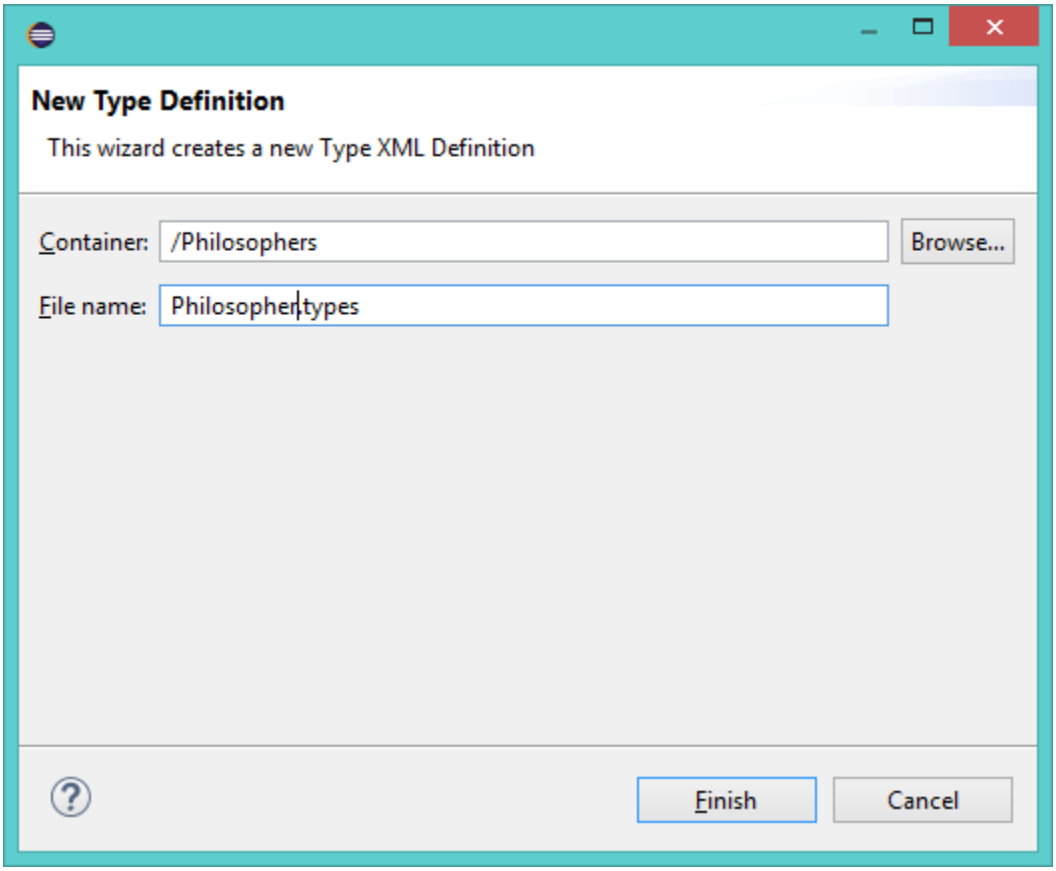
Select Type Editor from Tool bar which opens the below wizard.



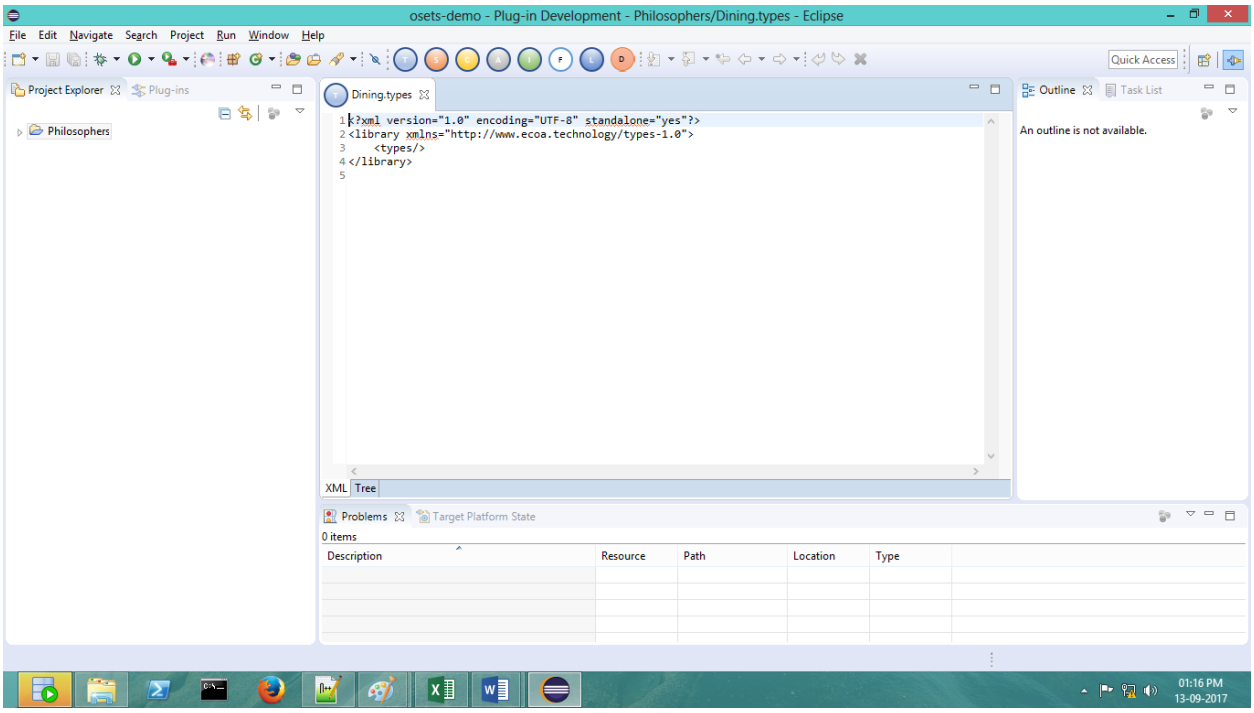
Select the container name as the top-level Project folder Philosophers



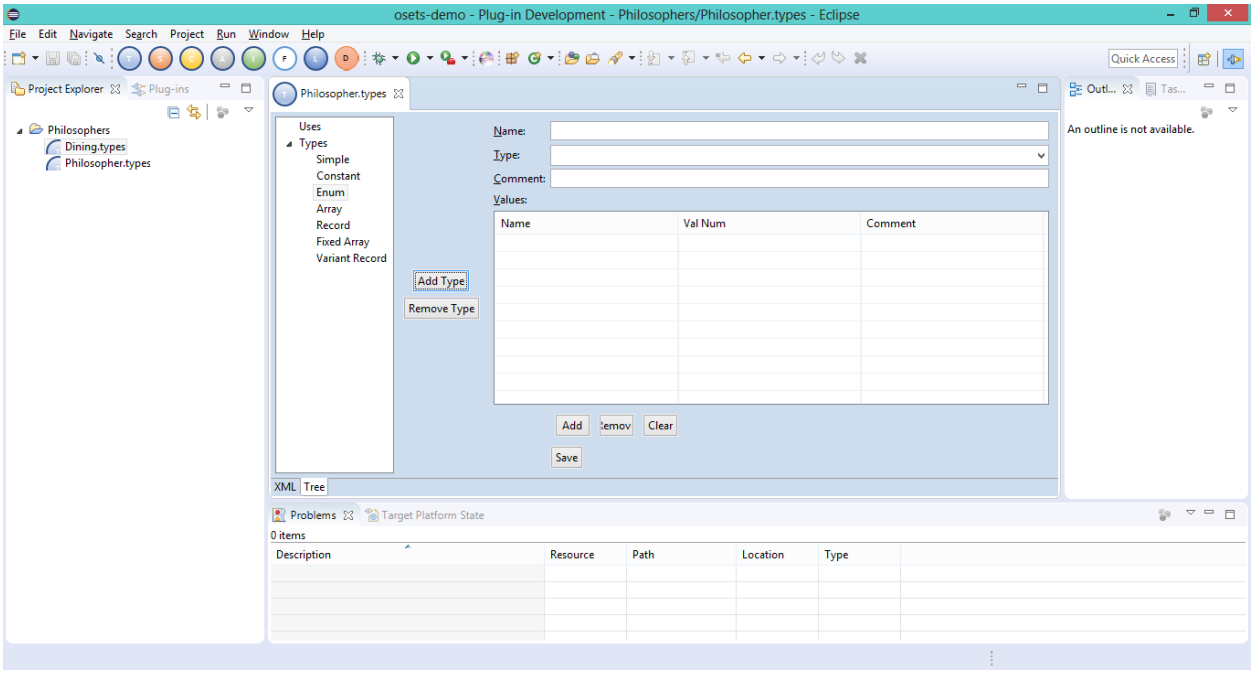
Change the file name to Philosopher.types



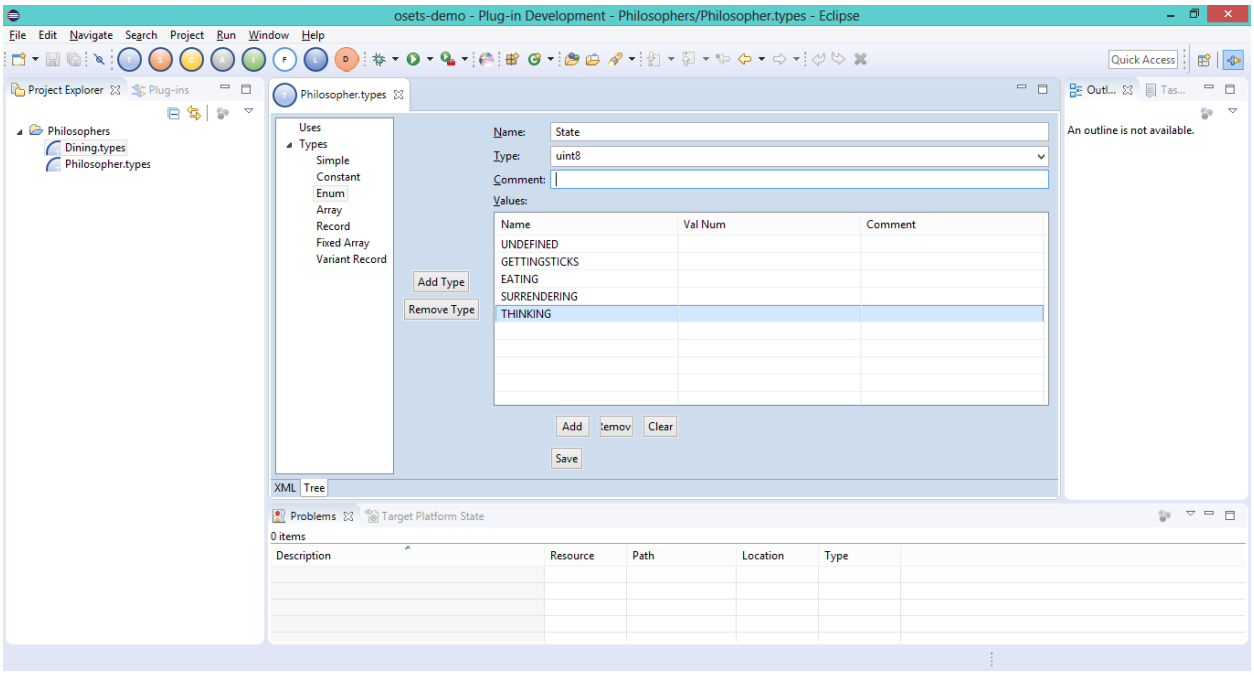
The initial XML Definition is shown



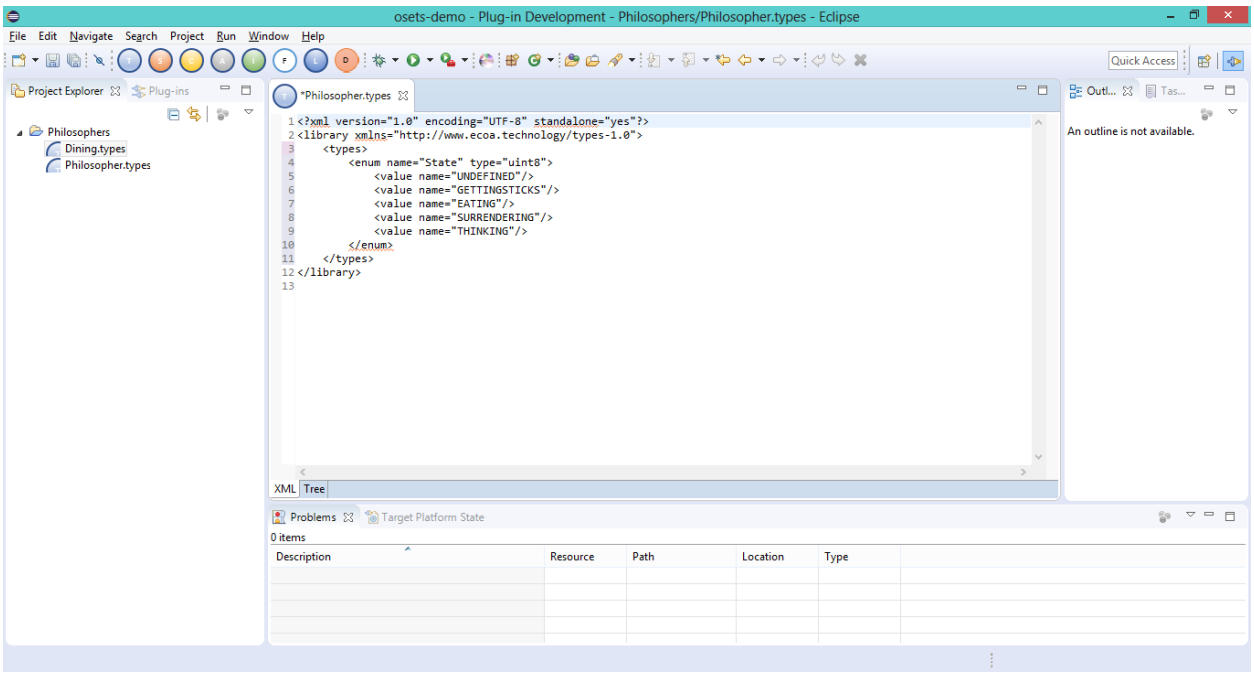
Select the tree tab at the bottom to move to Tree Editor to add the new types. Select Enum and press Add Type



Tis opens the Enum Types Editor. Enter the details and Click Save (Use, Add, Remove and Clear buttons to modify the list of possible values).



Verify the generated XML.

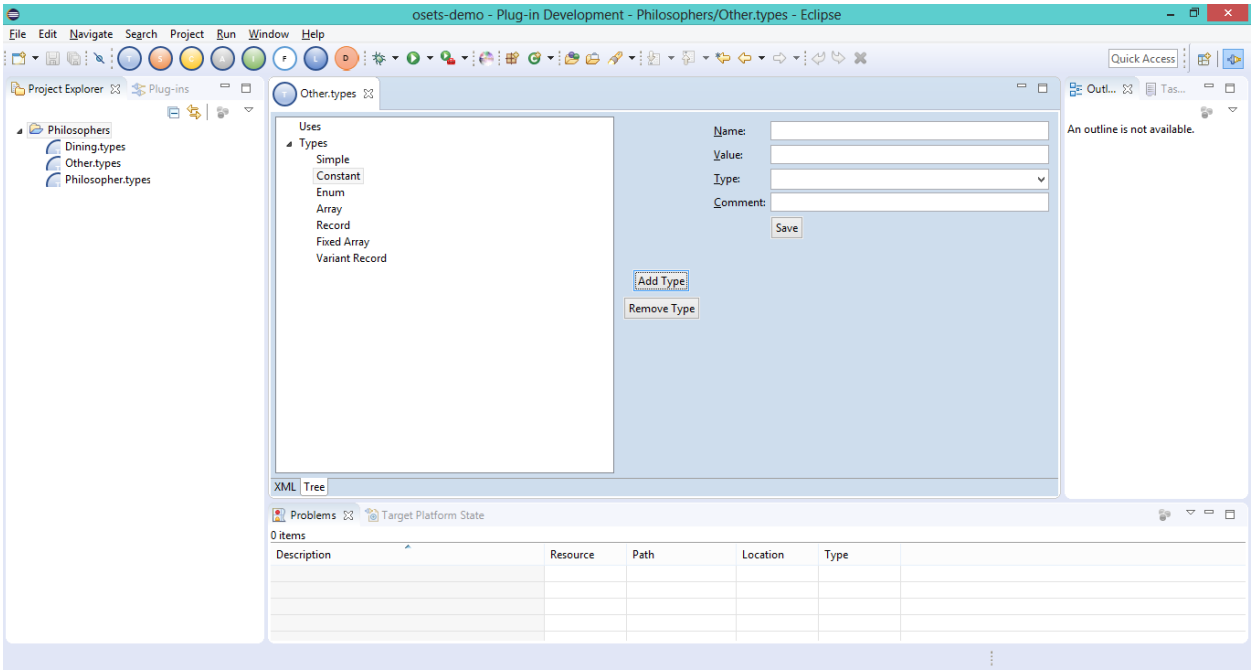


3.1.2 Other Definitions

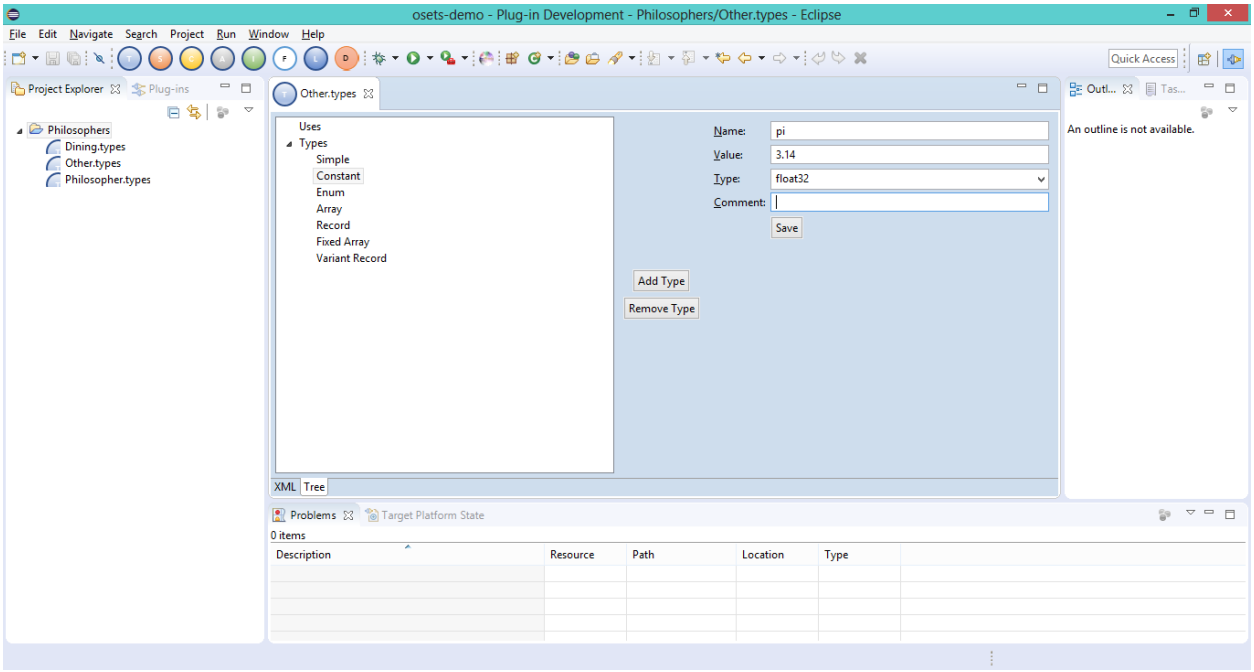
Apart from the types defined on the Dining Philosophers problem, we have the below additional types:

3.1.2.1 Constant Type

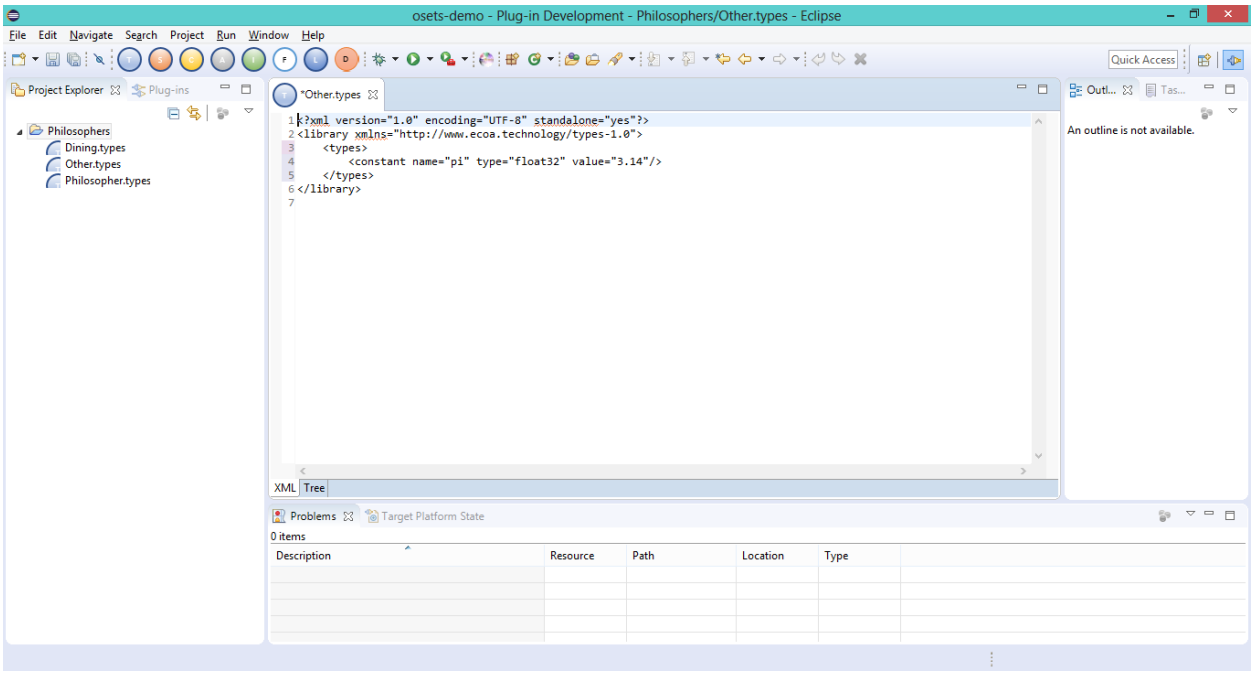
After Creating a Type Library using the steps outlined in the above two sections, Select Constant on the Type Tree and Select Add Type



This will open the Constant editor. Enter the details and click Save.

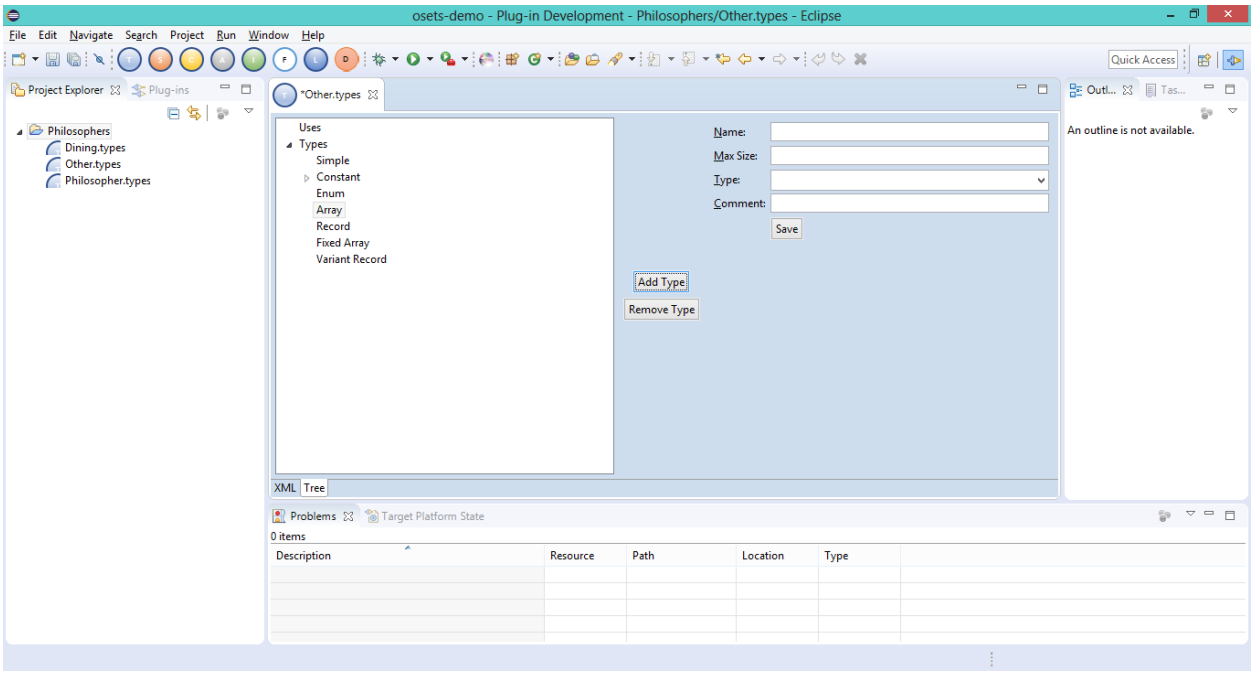


Verify the generated XML.

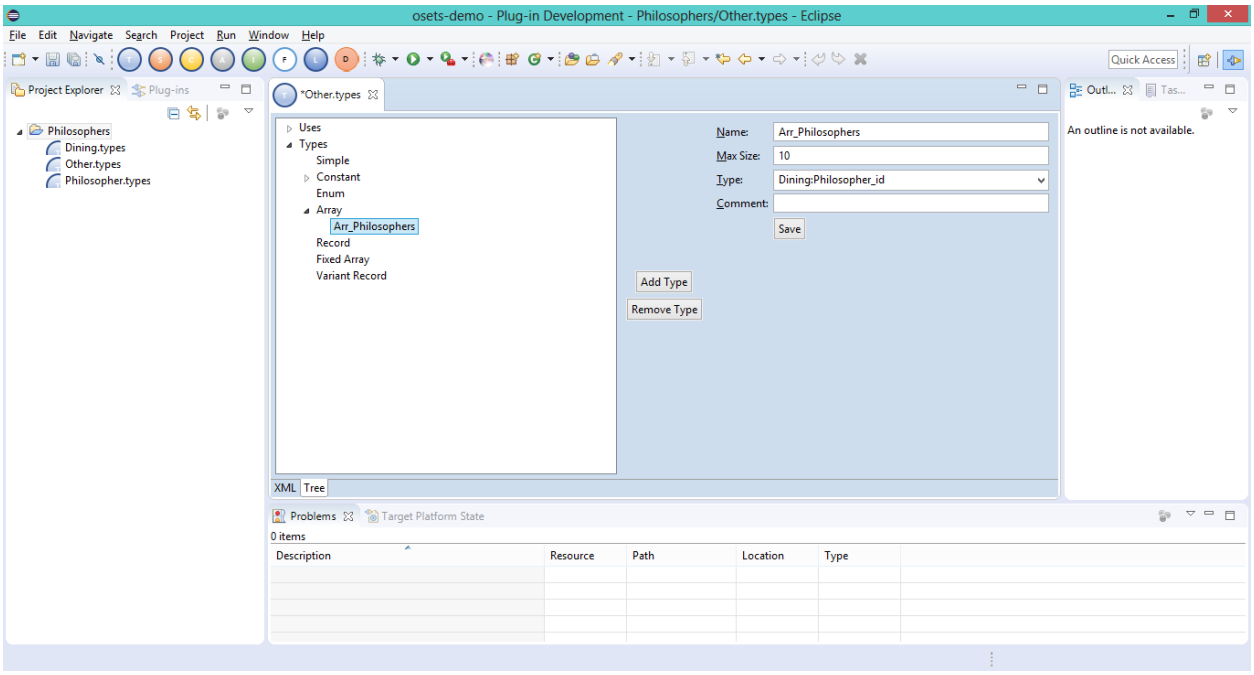


3.1.2.2 *Array Type*

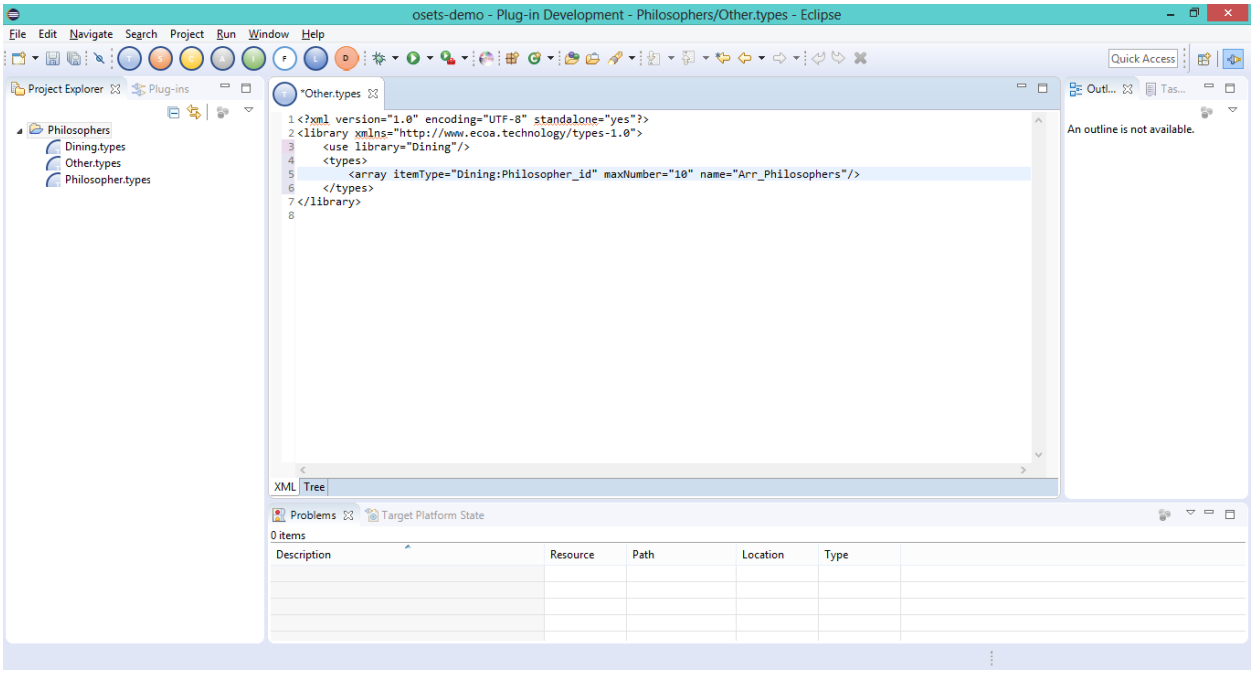
After Creating a Type Library using the steps outlined in the above two sections, Select Array on the Type Tree and Select Add Type



This will open the Array Type editor. Enter the details and click Save.

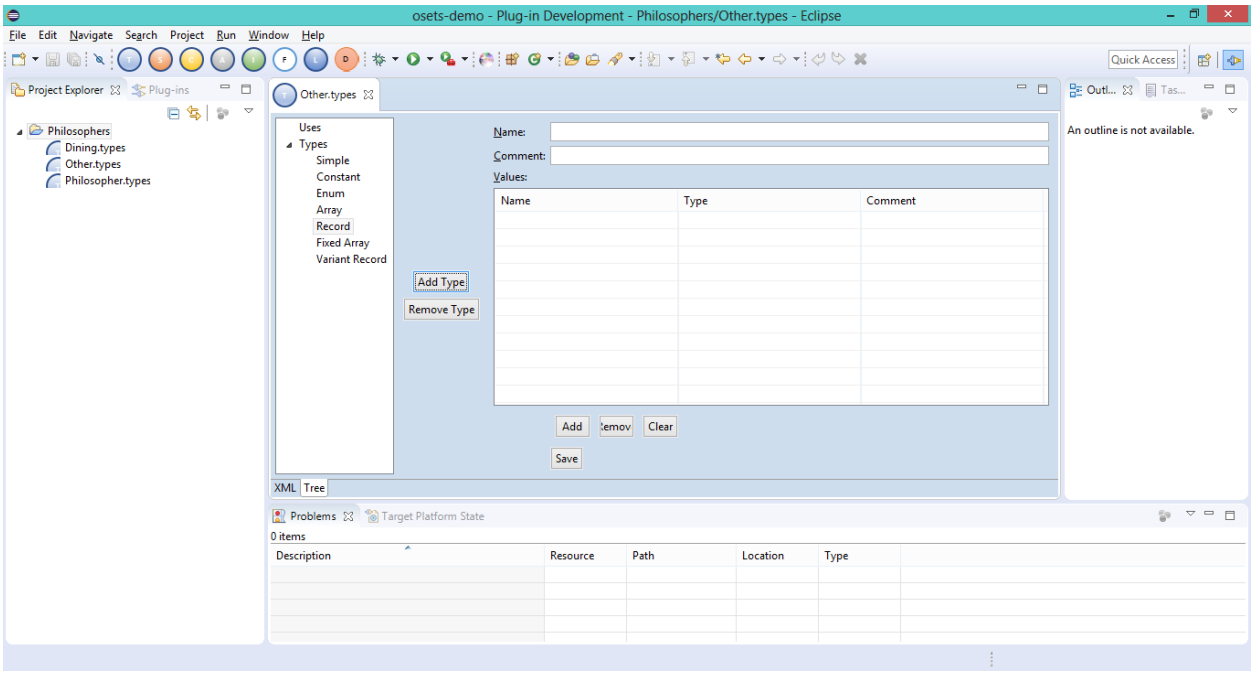


Verify the generated XML.

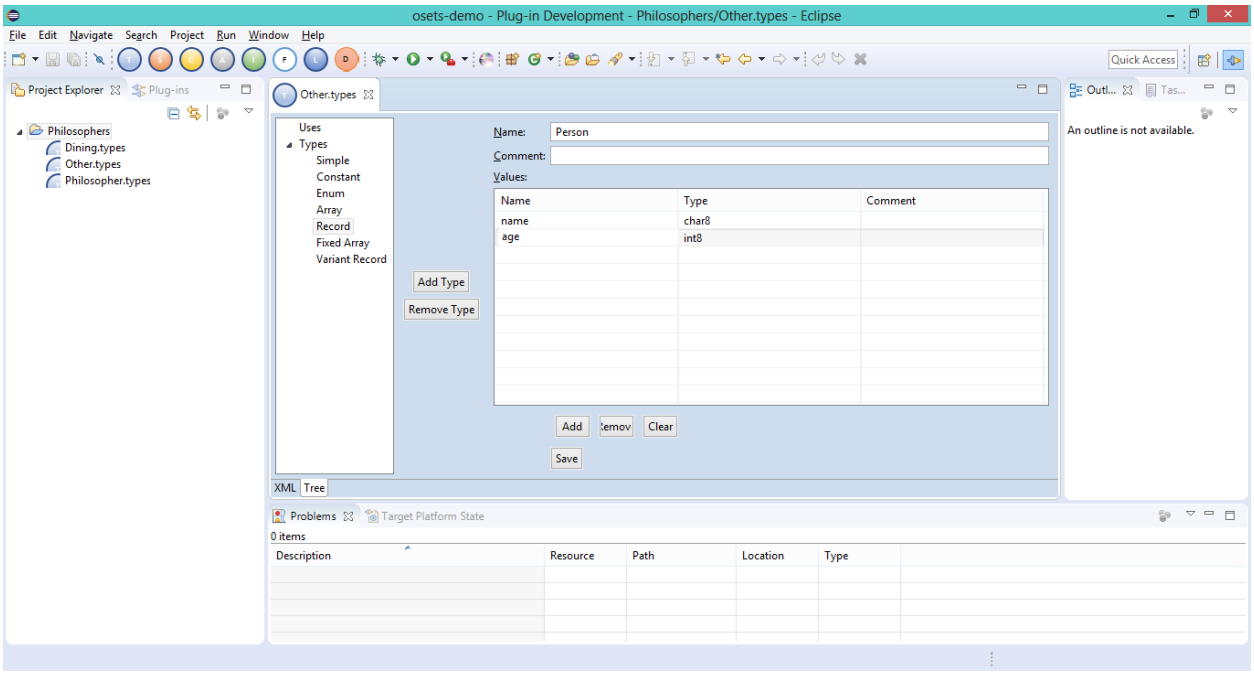


3.1.2.3 Record Type

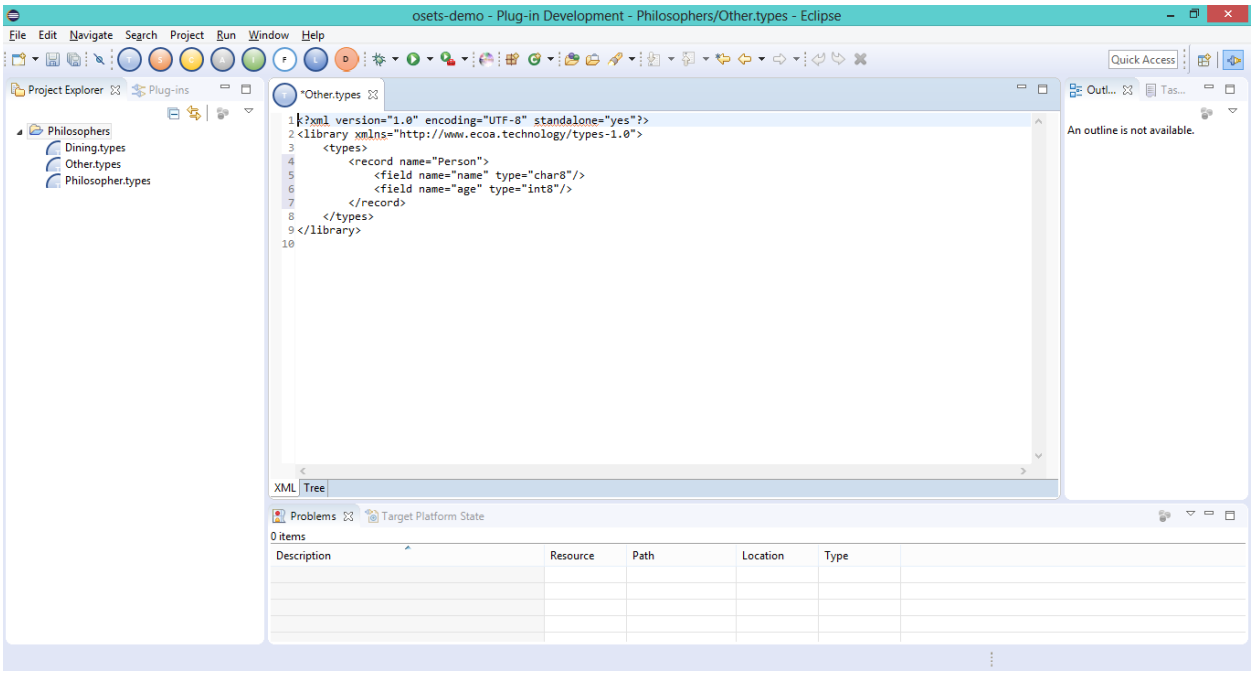
After Creating a Type Library using the steps outlined in the above two sections, Select Record on the Type Tree and Select Add Type



This will open the Record Type Editor. Enter the details and click Save (Use the Add, Remove, Clear buttons to modify the record fields).

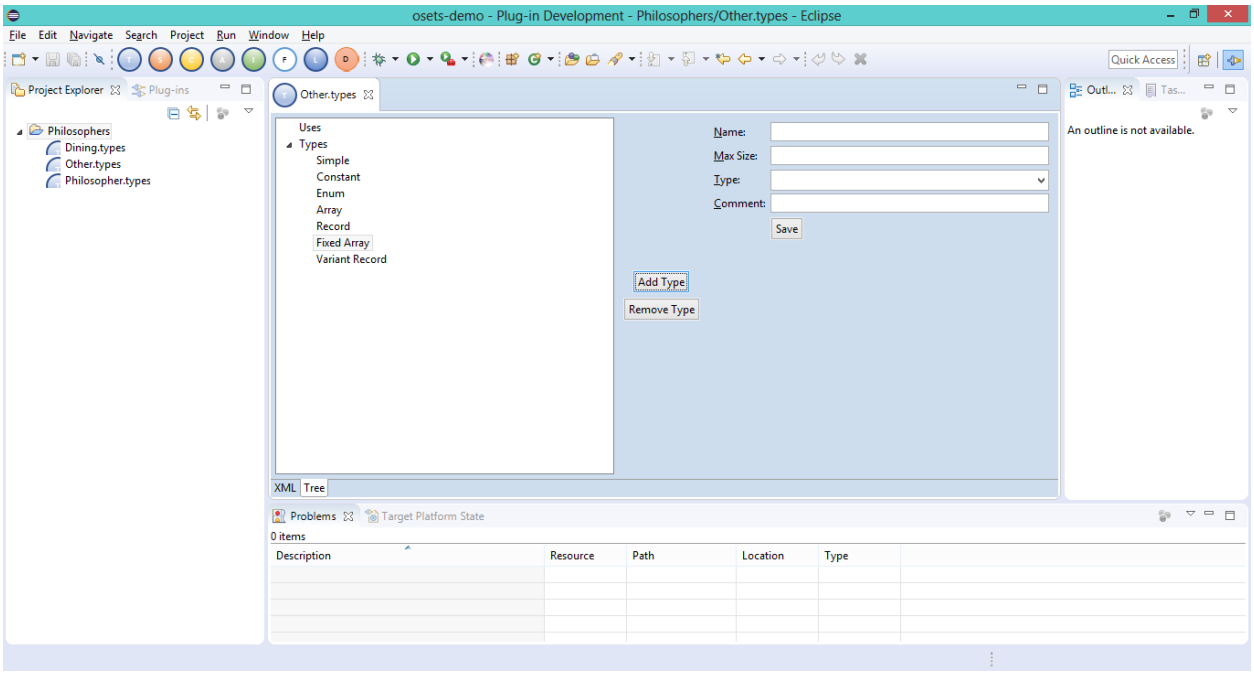


Verify the generated XML.

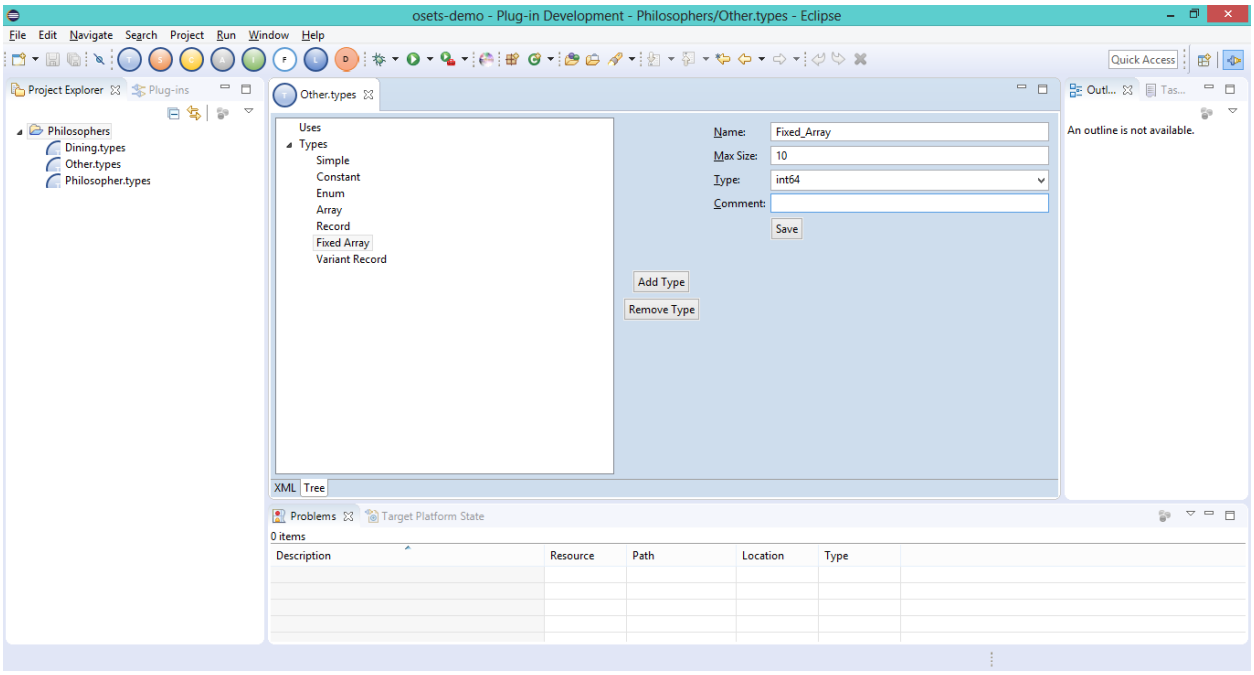


3.1.2.4 *Fixed Array Type*

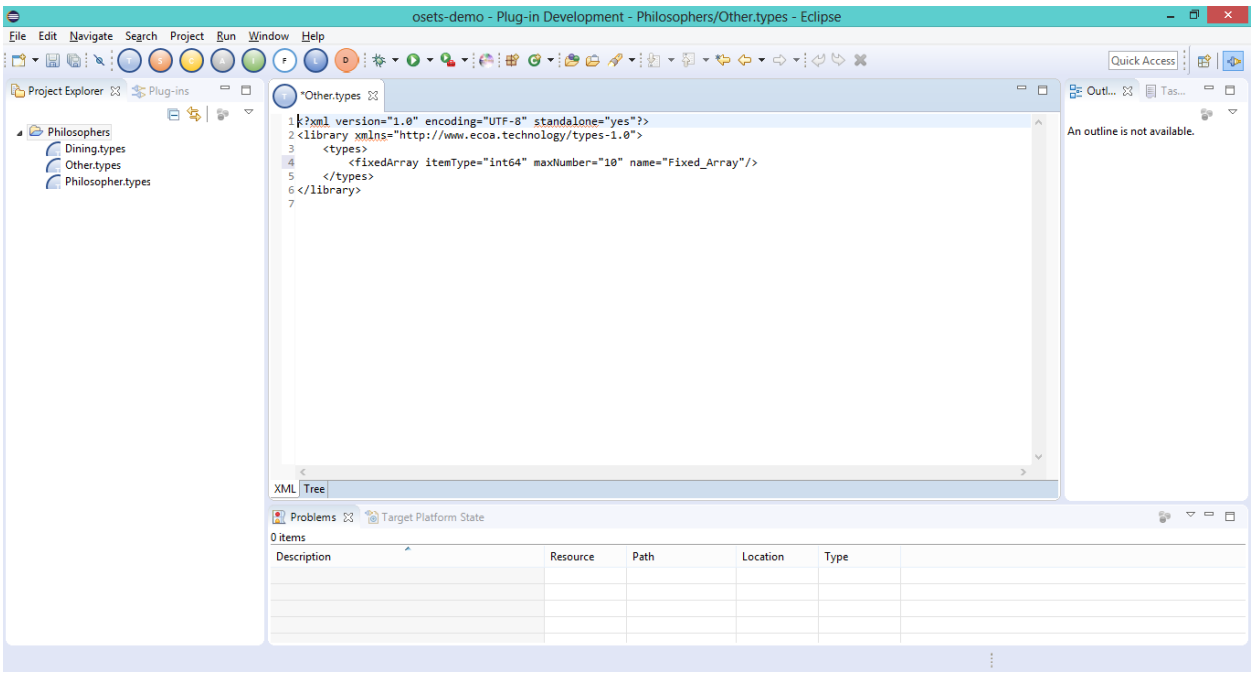
After Creating a Type Library using the steps outlined in the above two sections, Select Fixed Array on the Type Tree and Select Add Type



This will open the Fixed Array Type Editor. Enter the details and click Save.

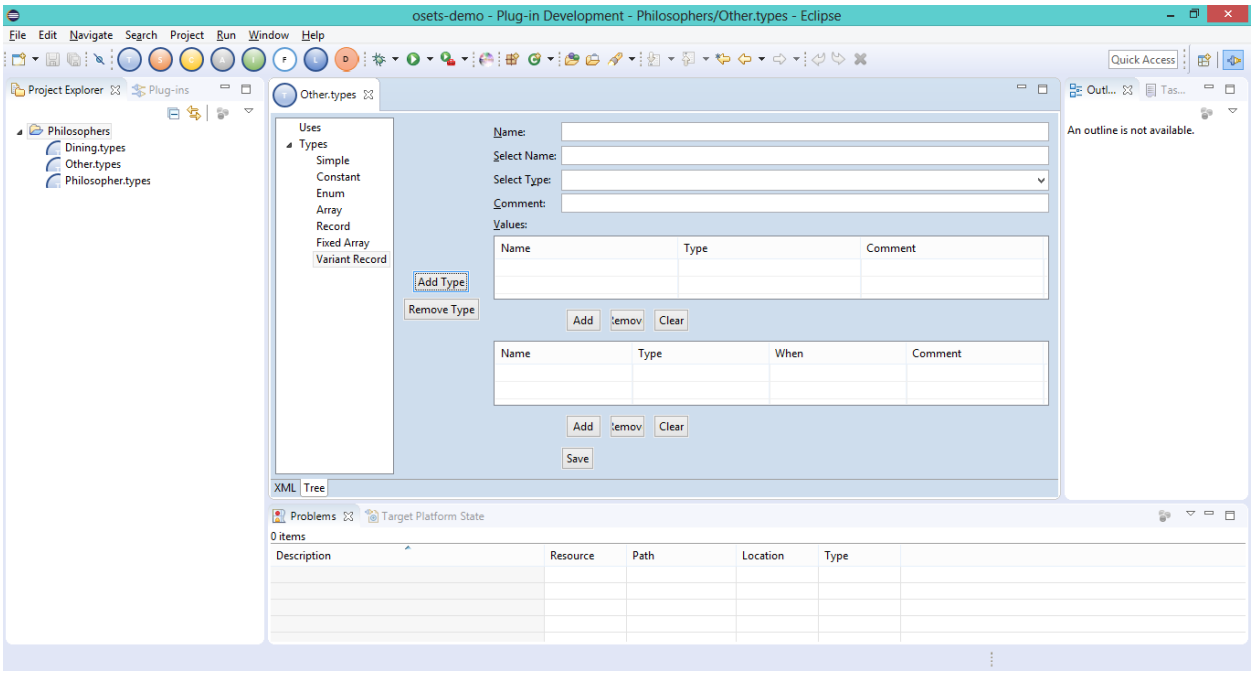


Verify the generated XML.

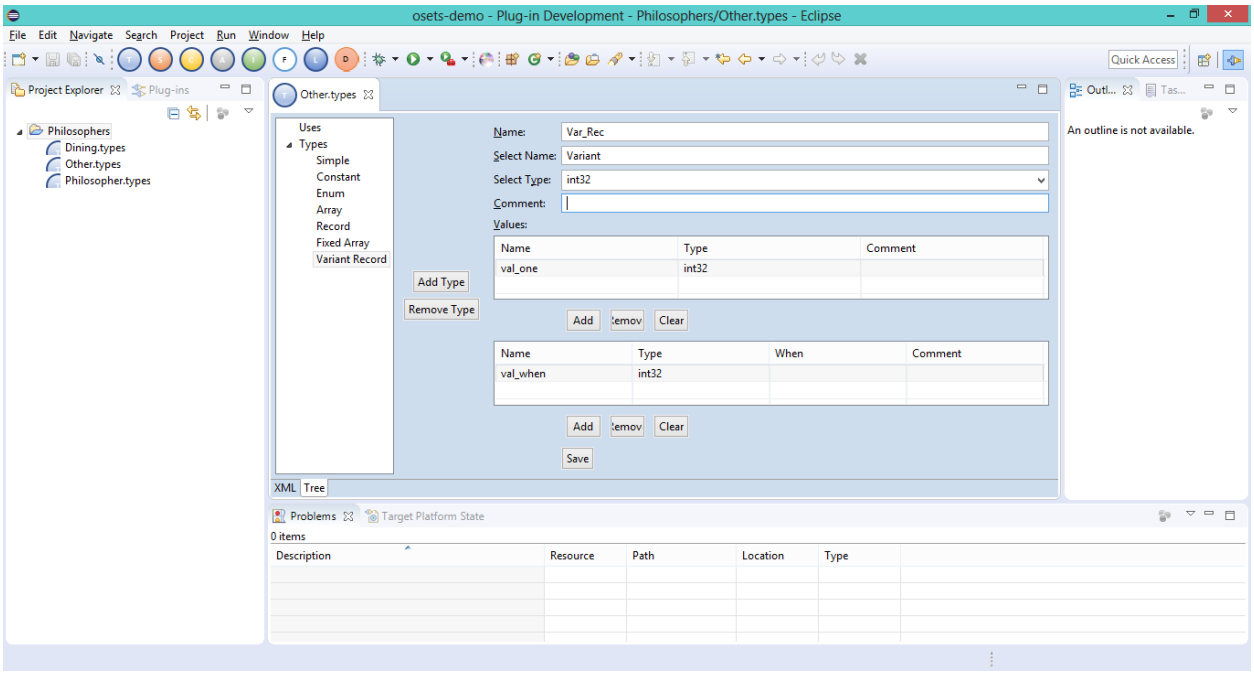


3.1.2.5 Variant Record Type

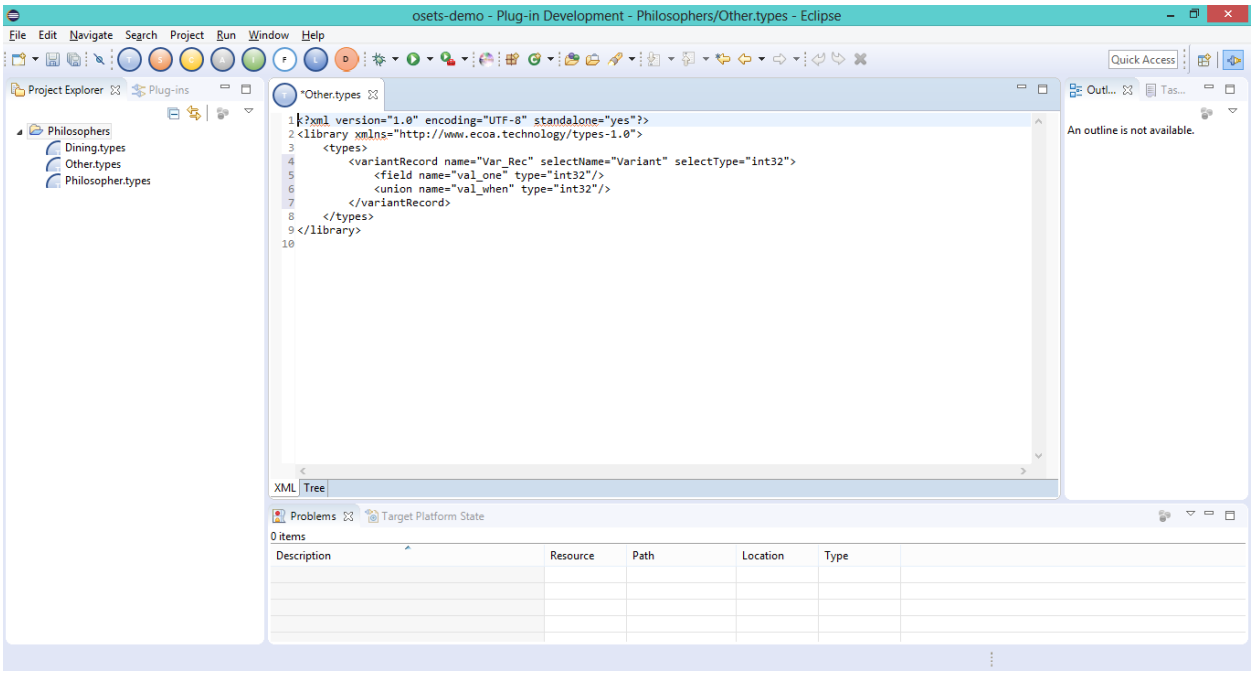
After Creating a Type Library using the steps outlined in the above two sections, Select Variant Record on the Type Tree and Select Add Type



This will open the Variant Record type editor. Enter the details and click Save (Use individual Add, Remove, and Clear buttons to modify the sections of Variant Record).



Verify the generated XML.



This summarizes the creation of the Types.

3.2 Services Editor

ECOA has 3 types of Service Operations. One or Many of them combine to form a Service Definition. Each service definition is created in its own file. The Three Service Operations are:

1) Data
2) Event
3) Request Response

3.2.1 Scenario

The Dining Philosophers problem covers Request-Response Operations. The Service Definitions are constructed as below:

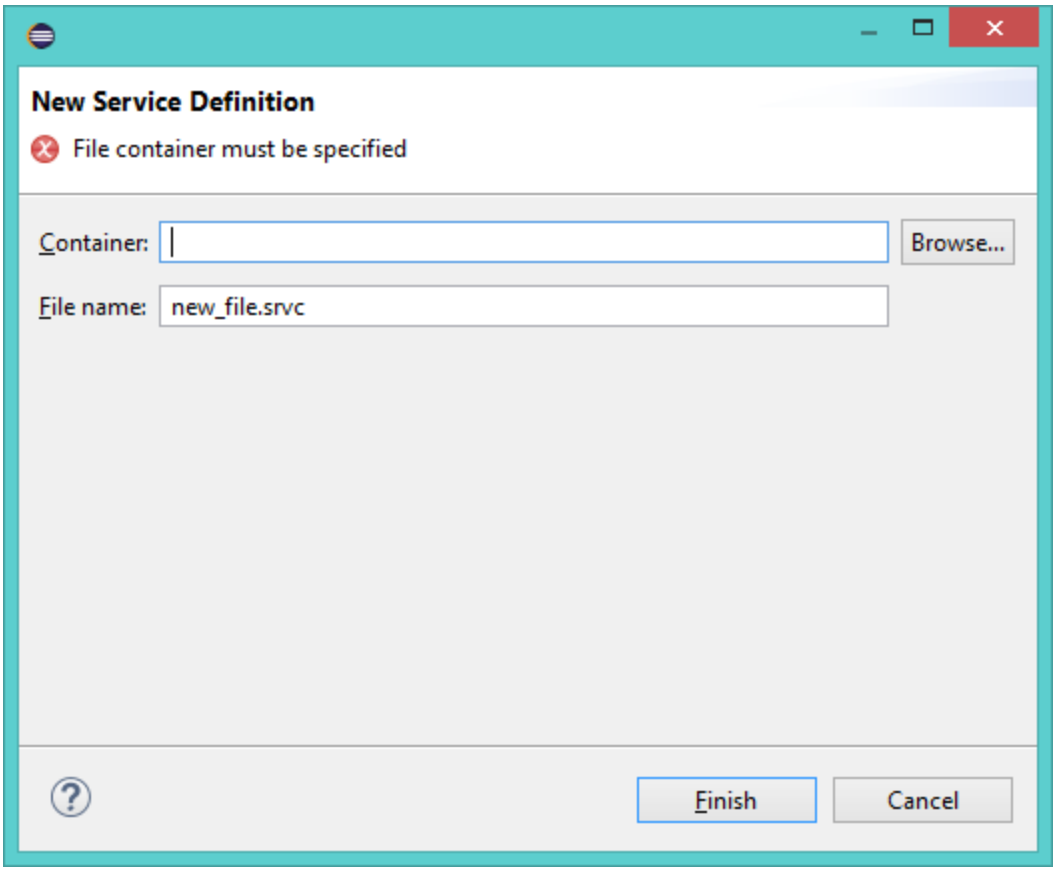
3.2.1.1 svc_Chopsticks

The Service svc_Chopsticks has two request-response service operations:

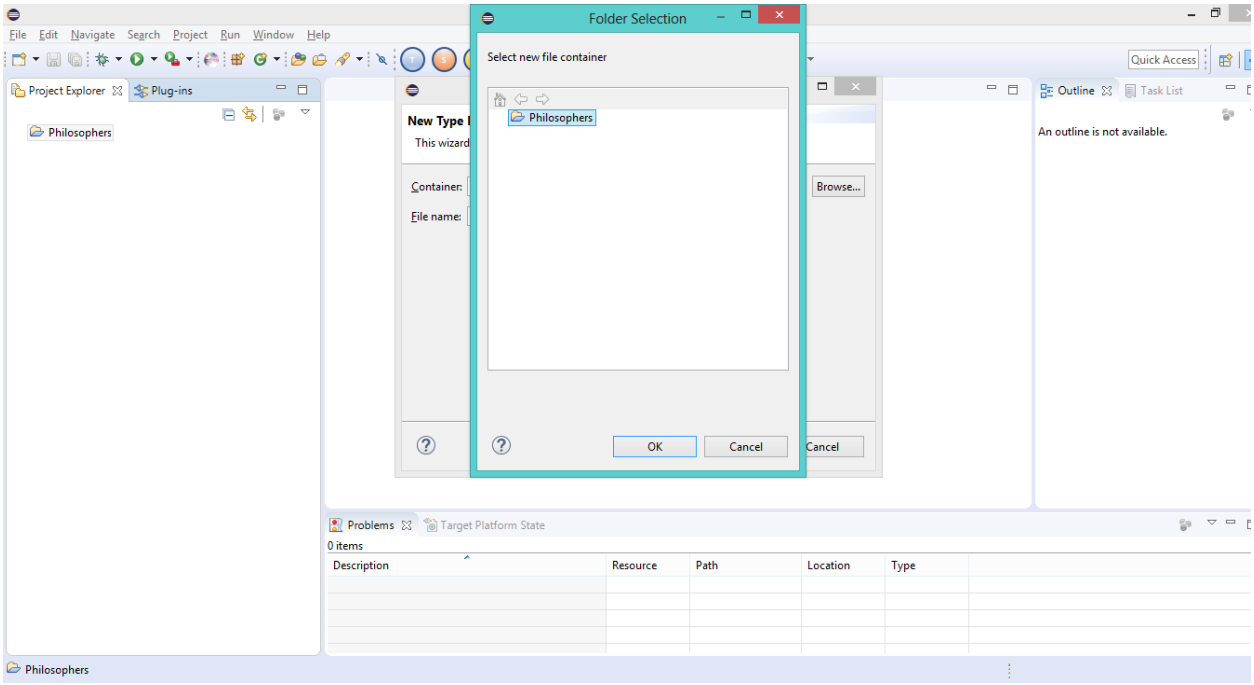
- take: with which - Dining:Chopstick_id and who - Dining:Philosopher_id as two input parameters, and taken – boolean8 as an output parameter
- surrender: with which - Dining:Chopstick_id and who - Dining:Philosopher_id as two input parameters

3.2.1.1.1 Process

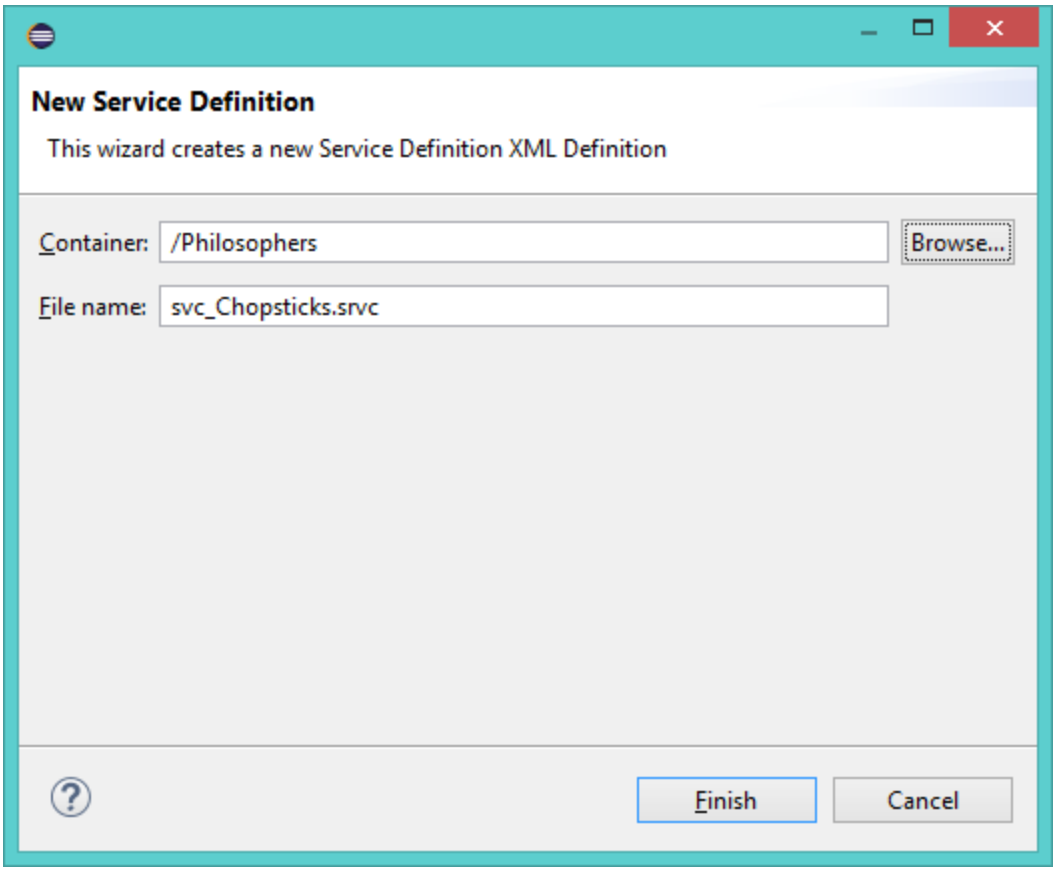
Select Service Editor from the Toolbar which opens the below Wizard:



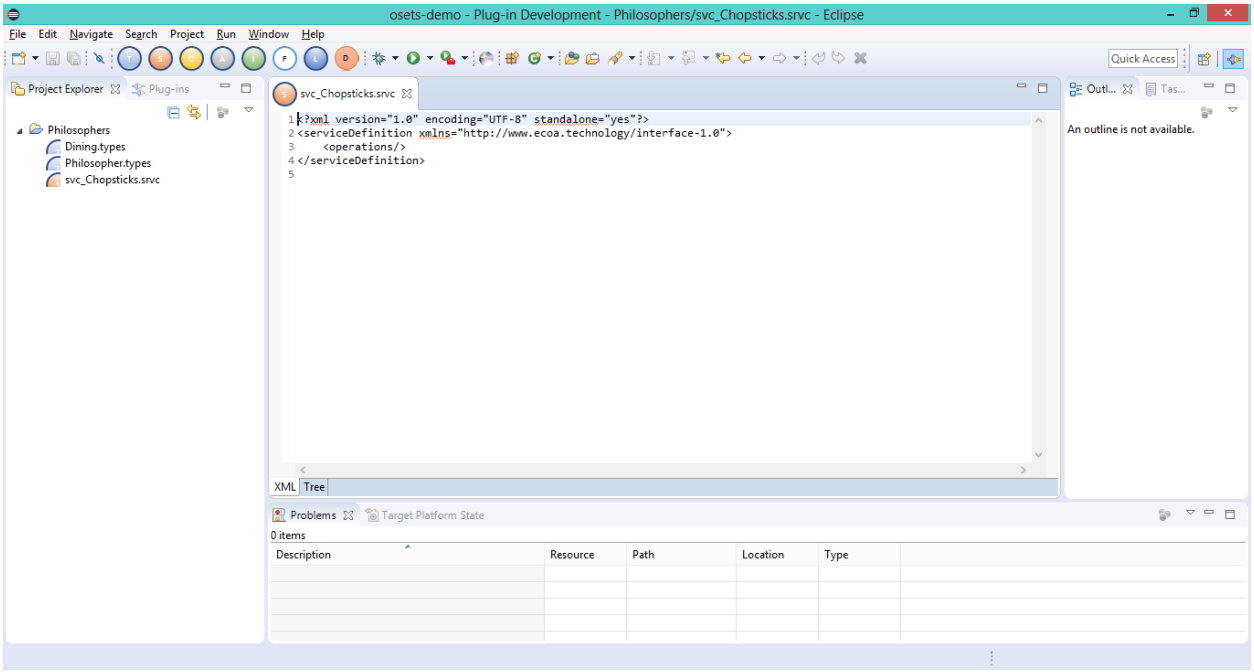
Select the container name as the top-level Project folder Philosophers



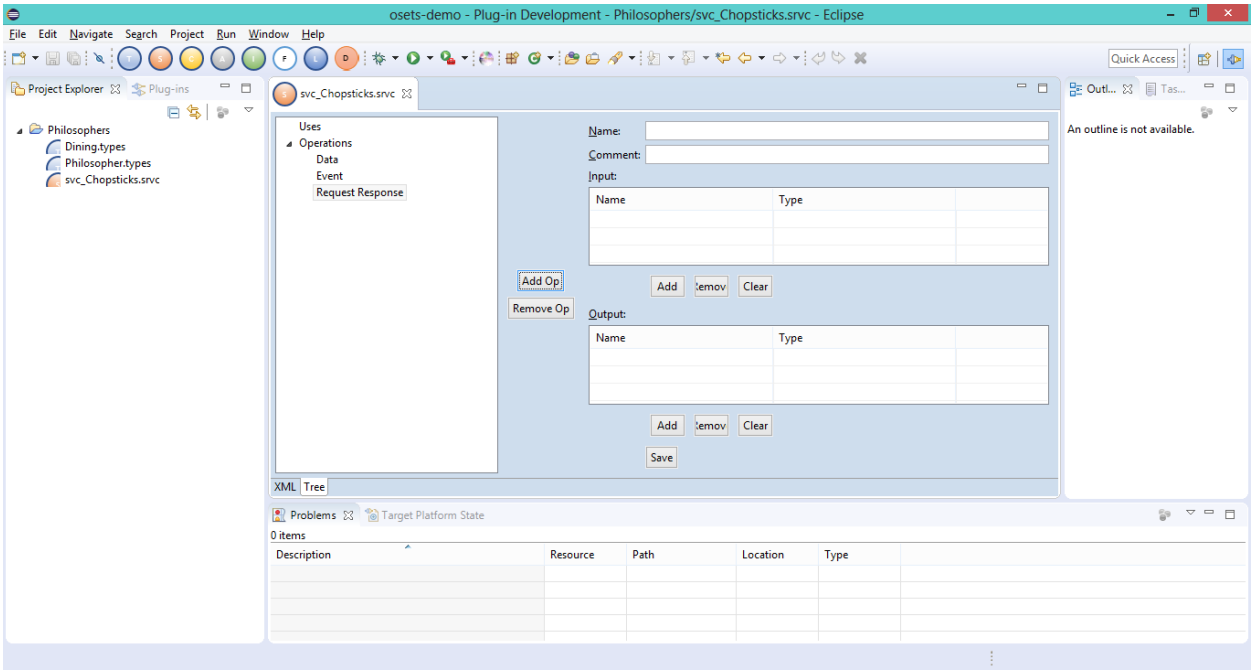
Change the file name to svc_Chopsticks.srvc and click on Finish.



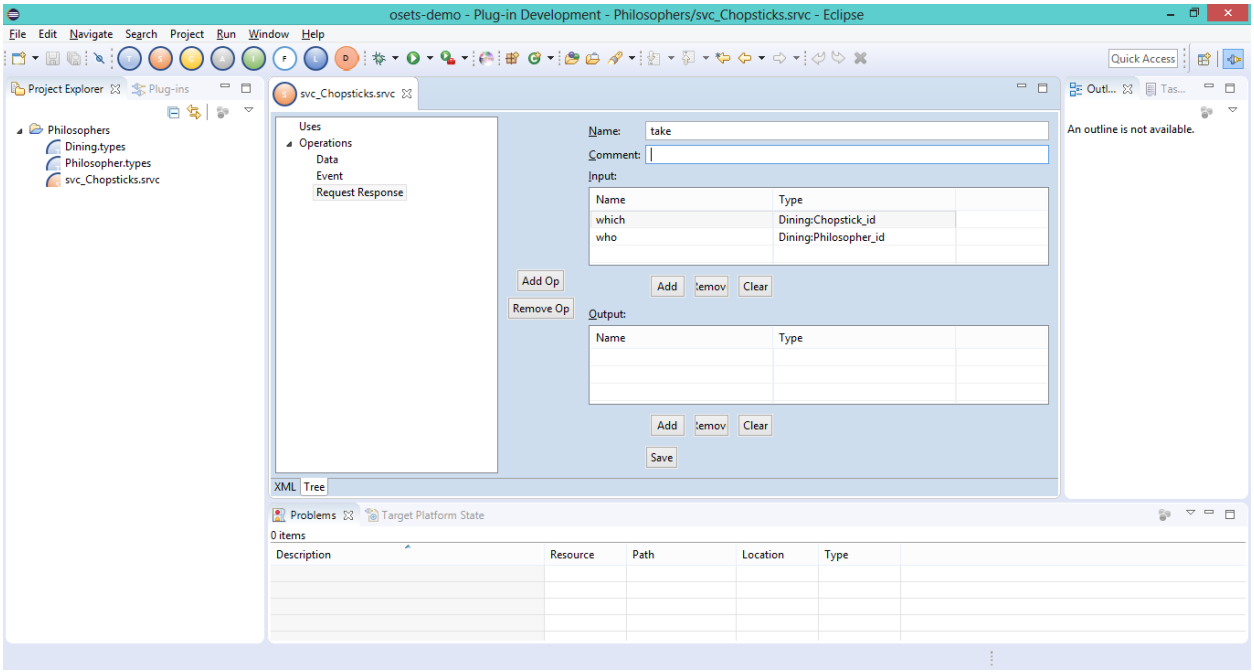
The initial XML definition is shown



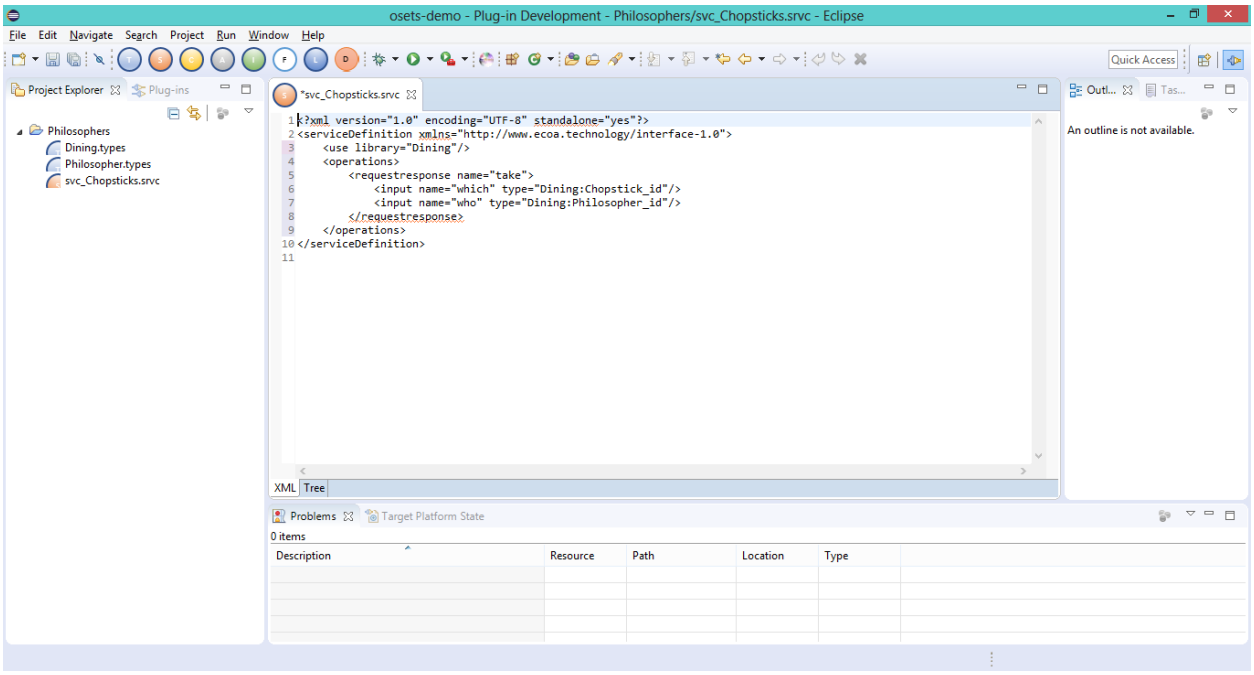
Select the tree tab at the bottom to move to Tree Editor to add the new Operations. Select Request Response and press Add Op



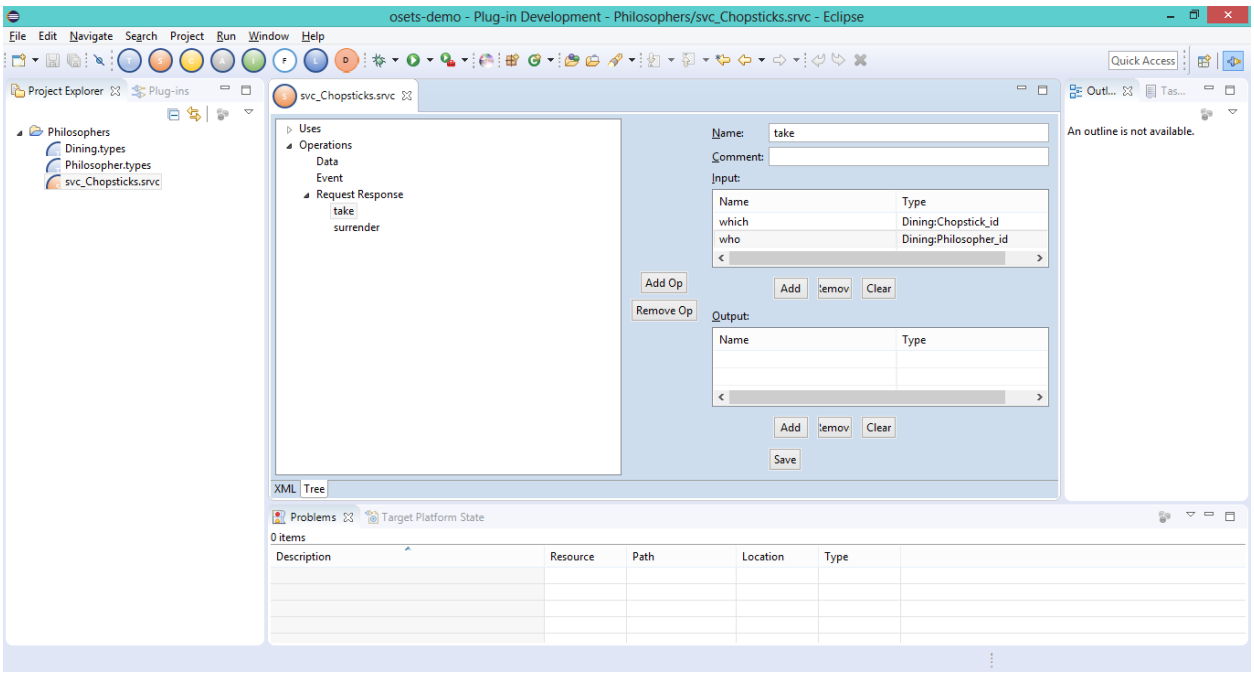
This opens the Request Response Operation Editor. Enter the details and Click Save (Use the individual Add, Remove and Clear buttons to modify the parameters)

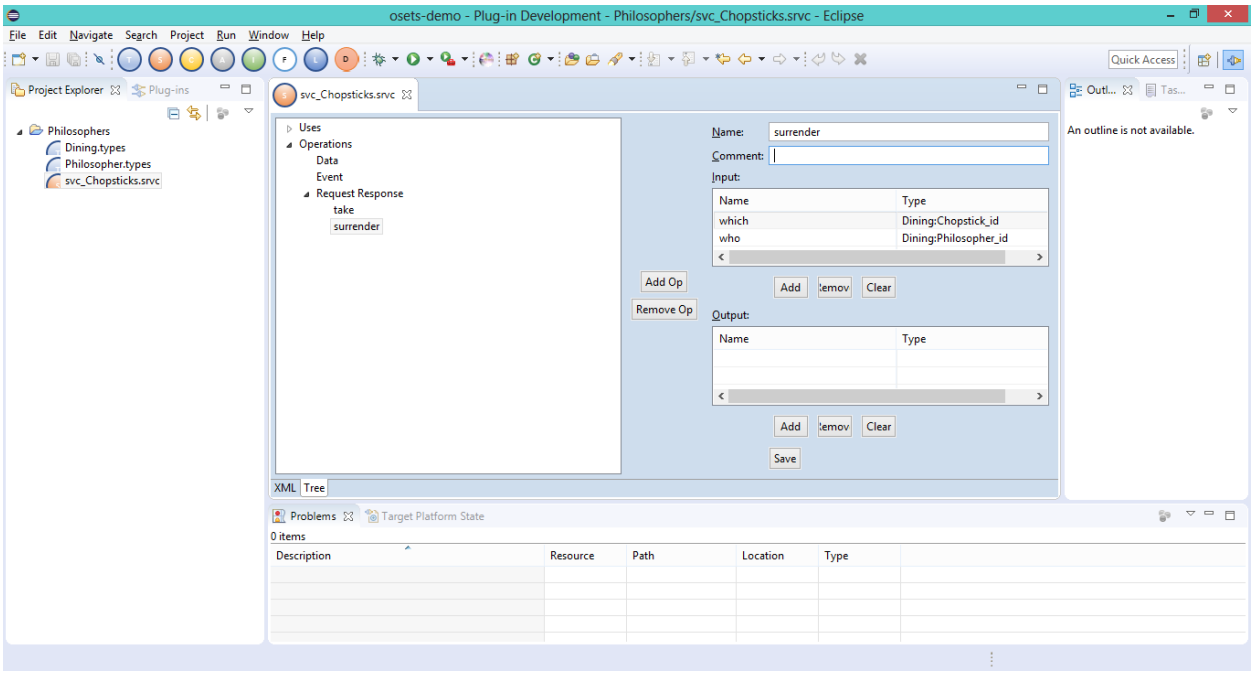


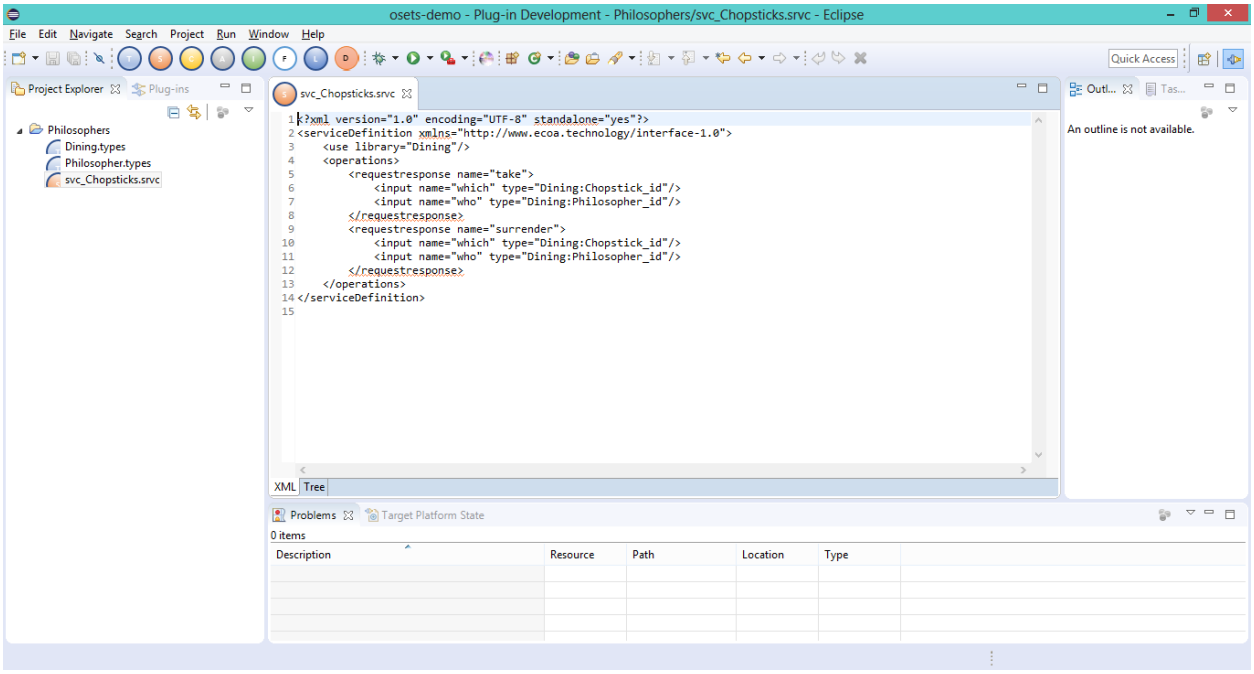
Verify the Generated XML.



Repeat the steps for surrender. The resultant Editor and XML Views are as below:





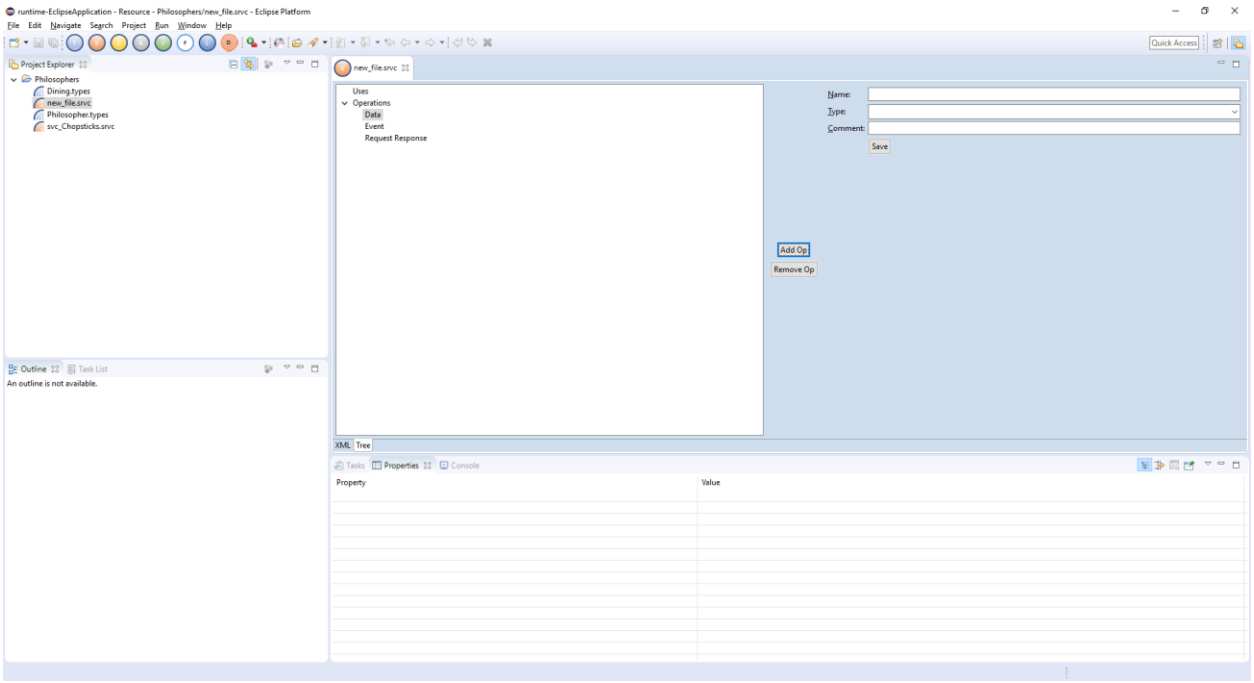


3.2.2 Other Definitions

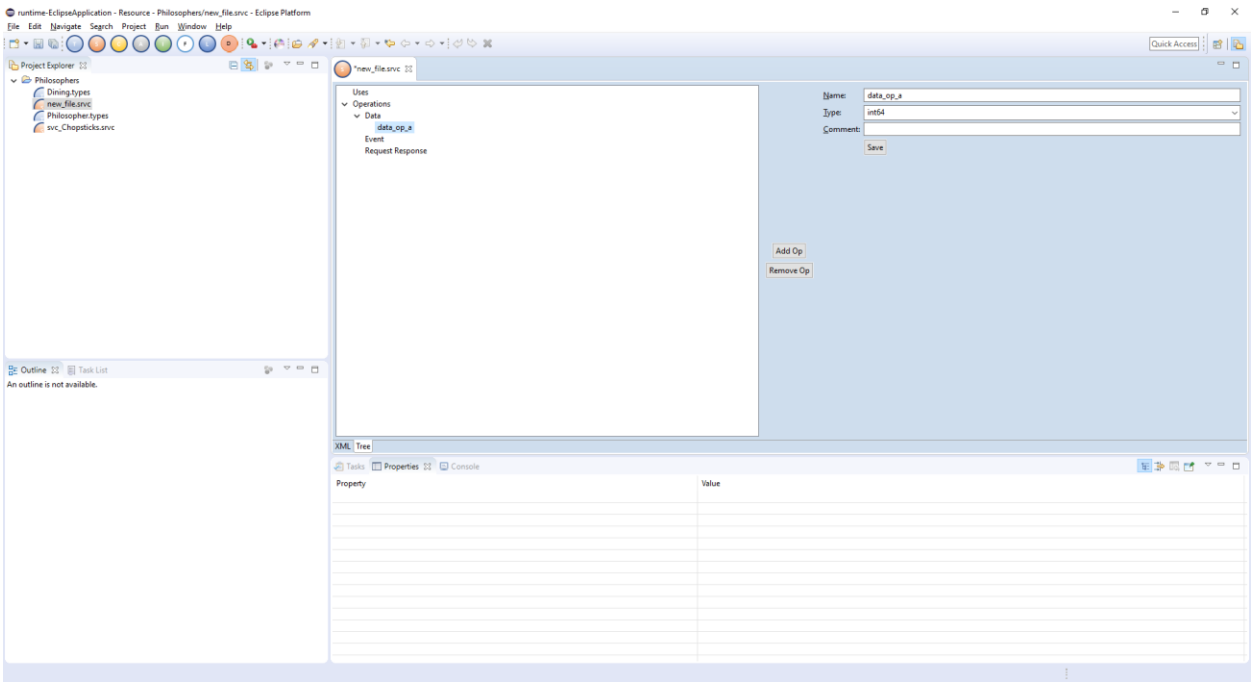
Apart from the Service Operations defined on the Dining Philosophers problem, we have the below additional operations:

3.2.2.1 Data Operation

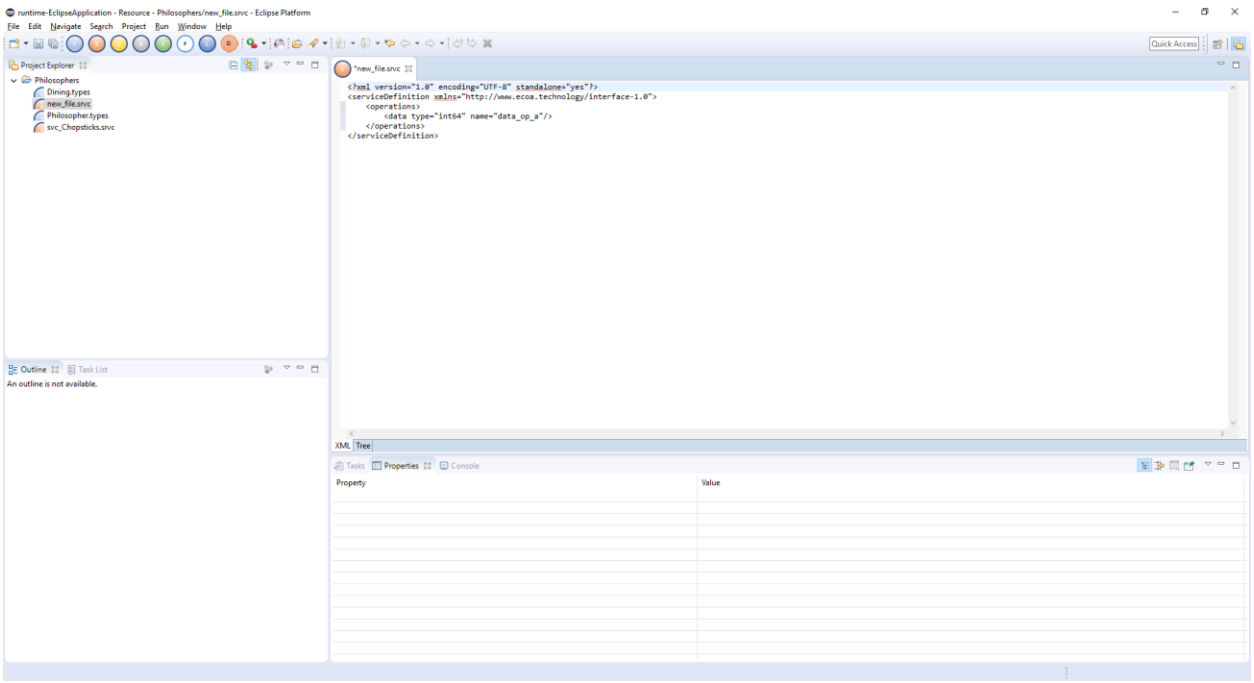
After Creating a Service Definition using the steps outlined in the above section, Select Data on the Operations Tree and Select Add Op



This will open the Data Operation editor. Enter the details and click Save.

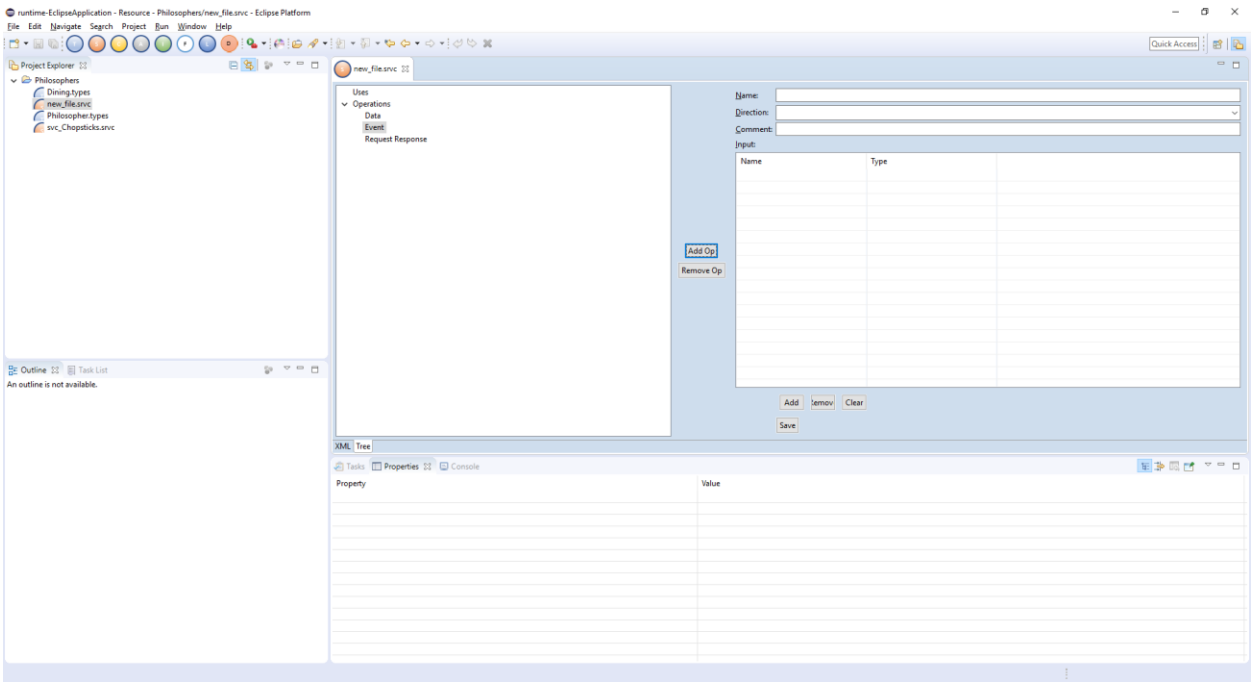


Verify the generated XML.

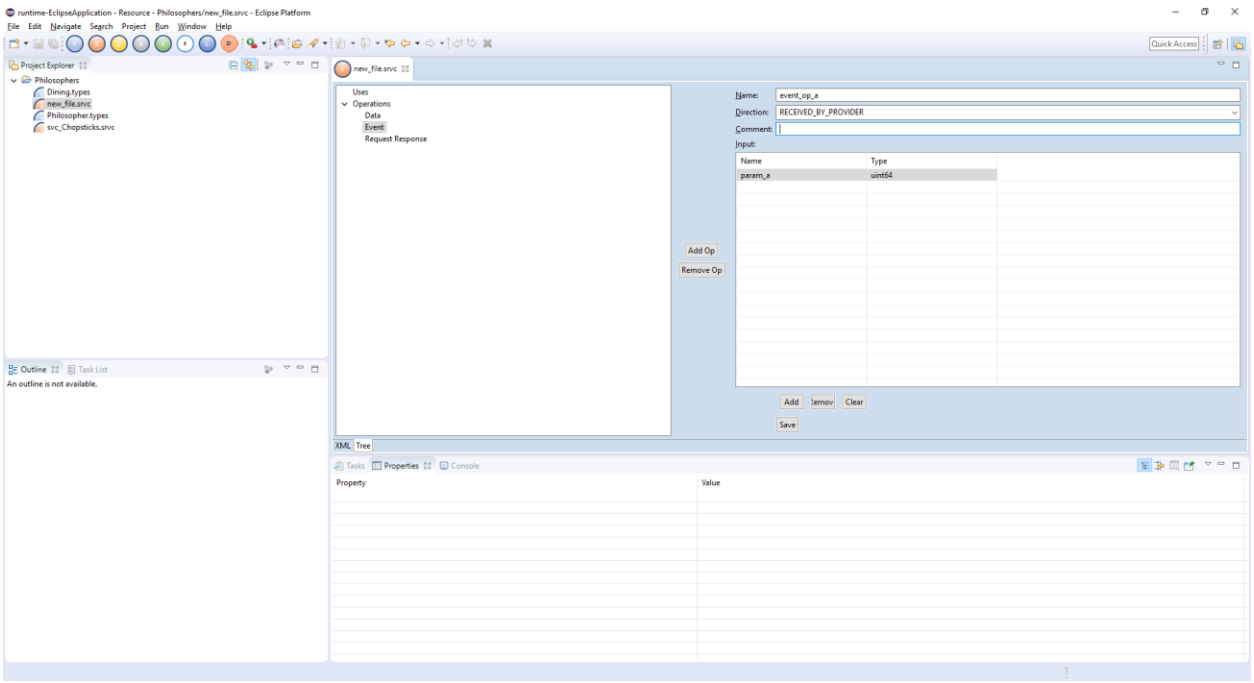


3.2.2.2 Event Operation

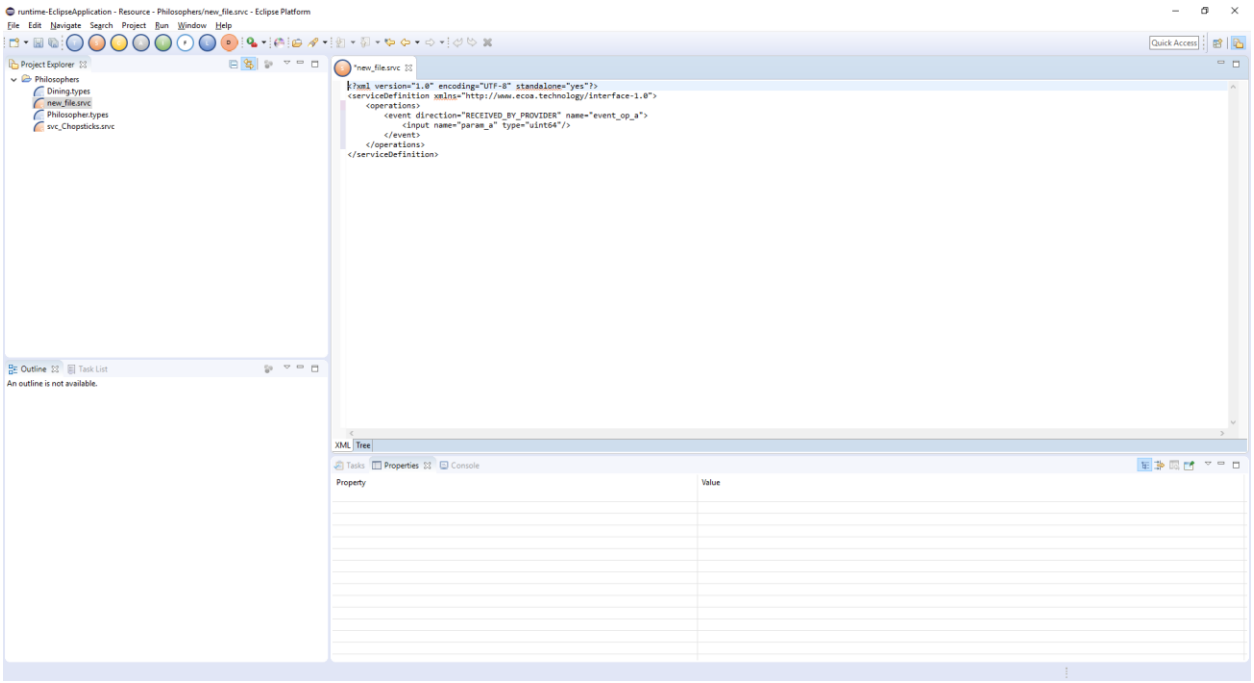
After Creating a Service Definition using the steps outlined in the above section, Select Event on the Operations Tree and Select Add Op



This will open the Event Operation Editor. Enter details and click on Save (Use Add, remove, and Clear buttons to modify the Input parameters).



Verify the generated XML.



3.3 Component Definition Editor

ECOA has two types of Interface definitions. One or Many of these two form a Component Definition. Each Component Definition has its own file. The two interface definitions are:

1) Provided Service

2) Required Service

There is additional provision to add Properties as well.

3.3.1 Scenario

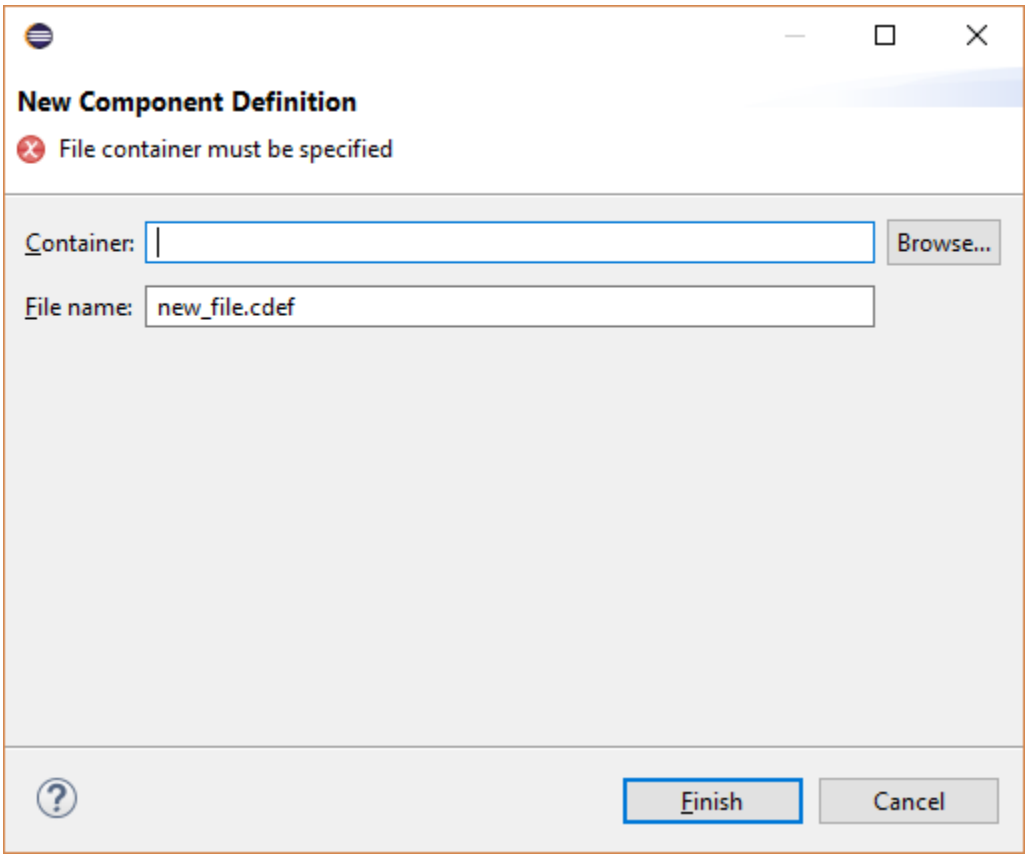
The Dining Philosophers problem covers both Provided and Required Service. The Component Definition is constructed as below:

3.3.1.1 Table

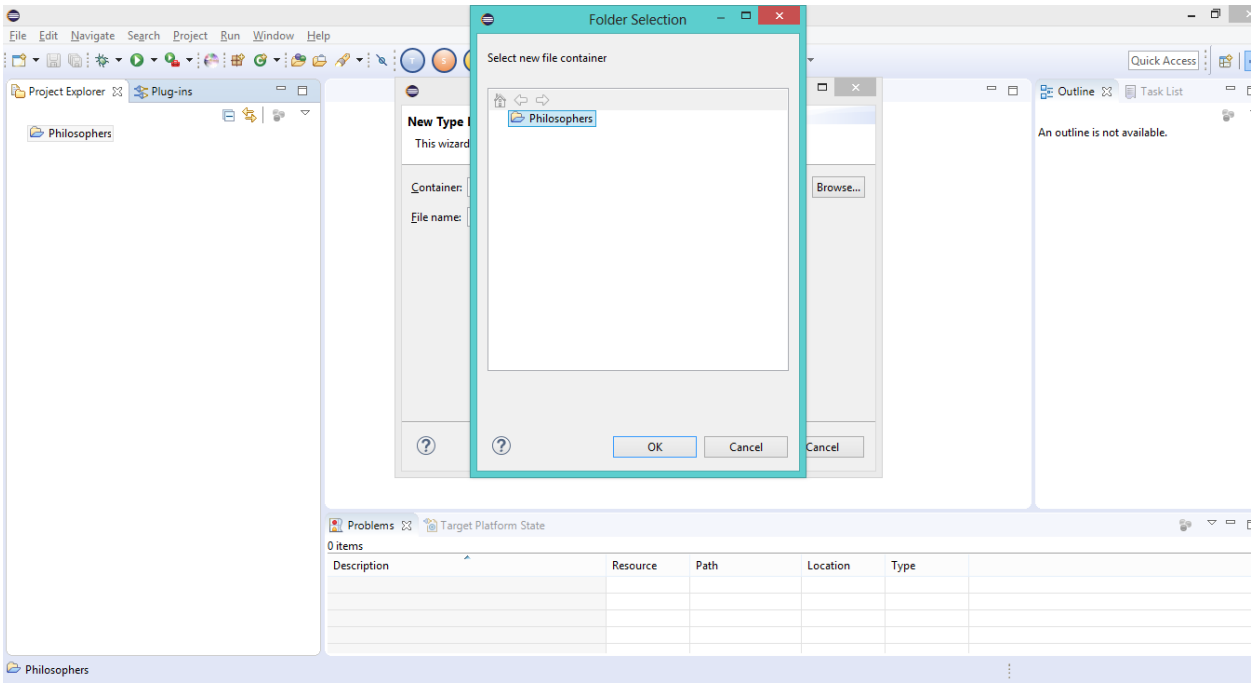
The Table Component Definition provides a Service Chopsticks of type svc_Chopsticks.

3.3.1.1.1 Process

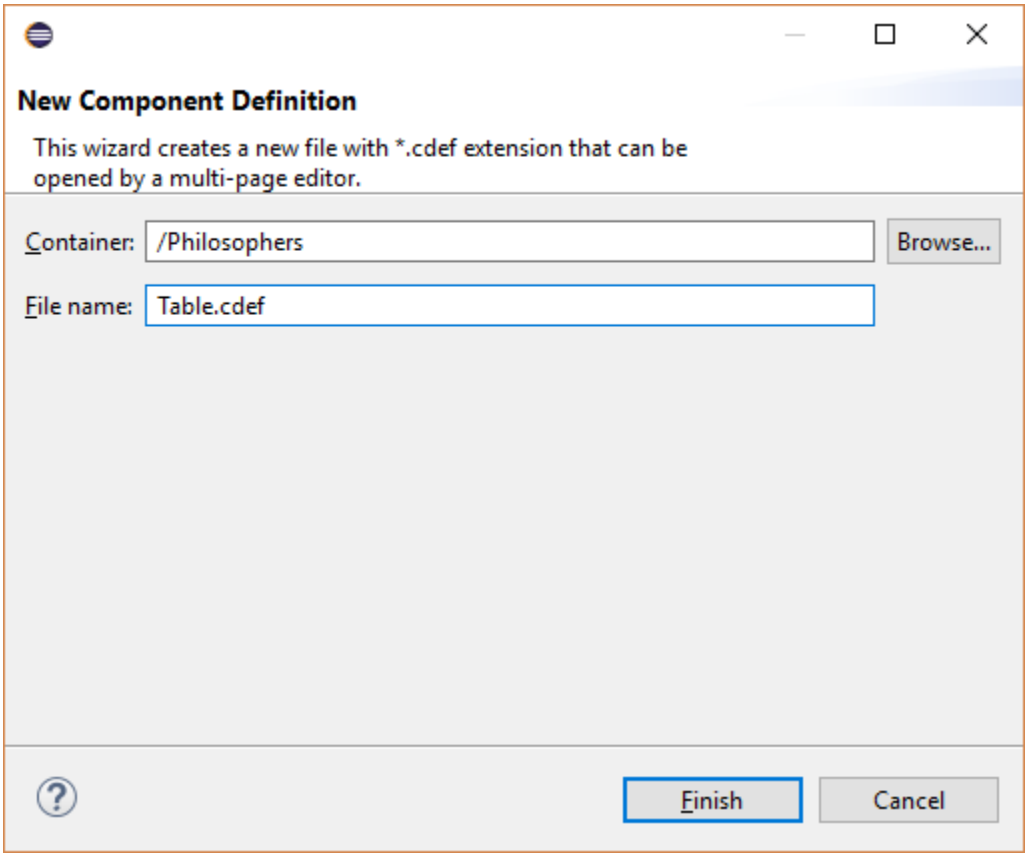
Select Component Definition Editor from the Toolbar which opens the below Wizard:



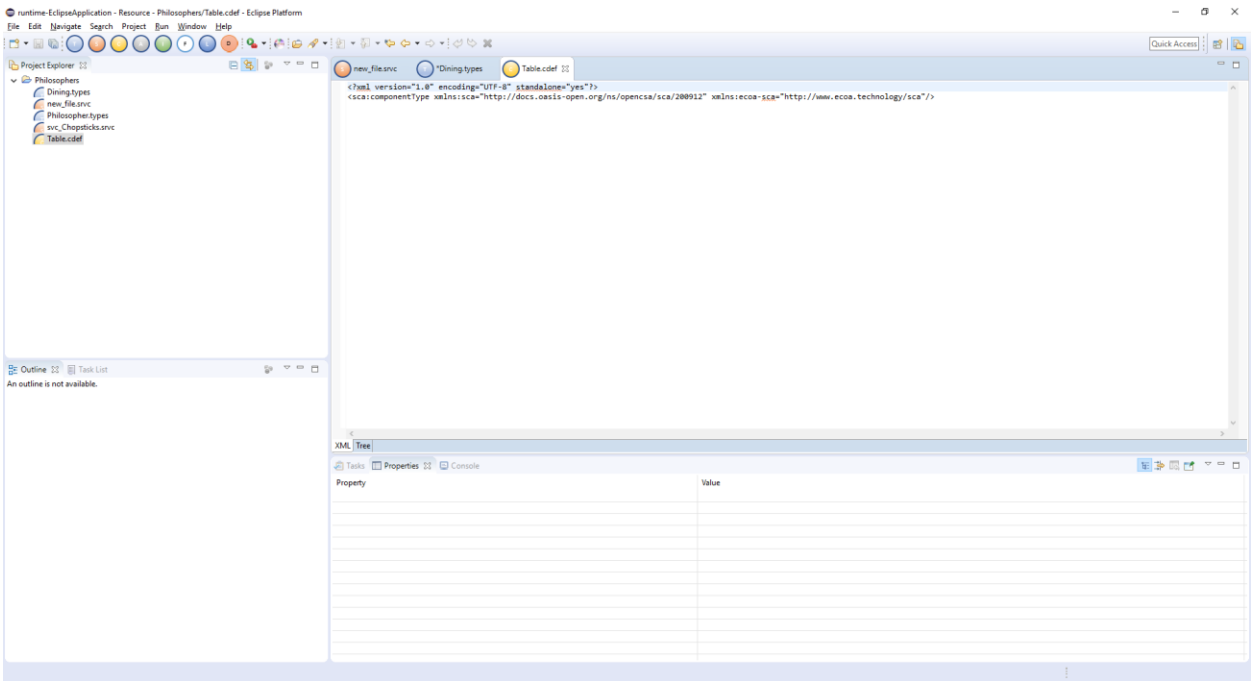
Select the container name as the top-level Project folder Philosophers



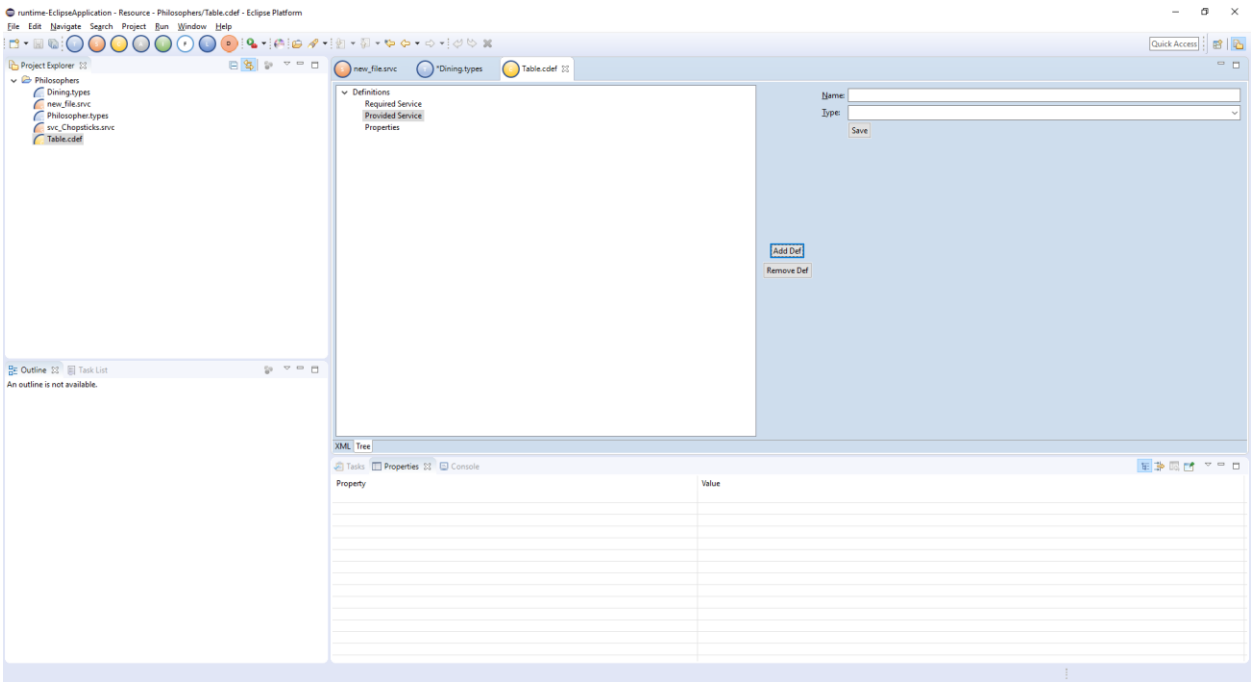
Select the file name as Table.cdef and Click on Finish.



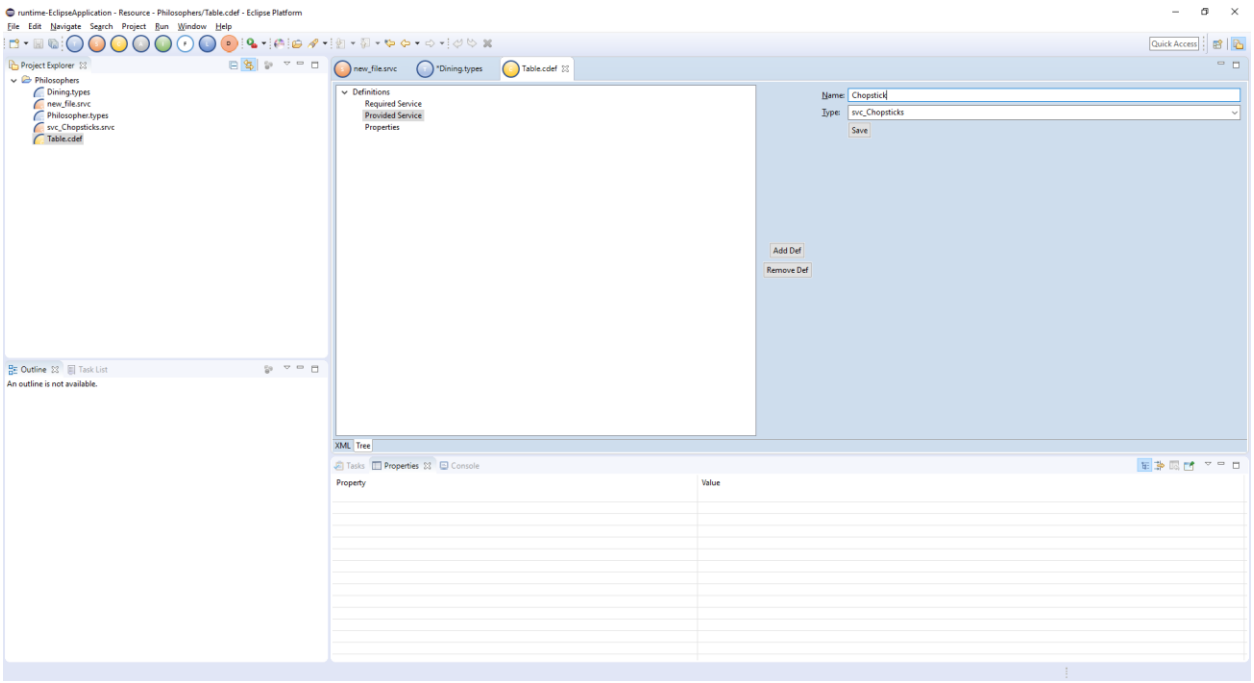
The initial XML Definition is shown.



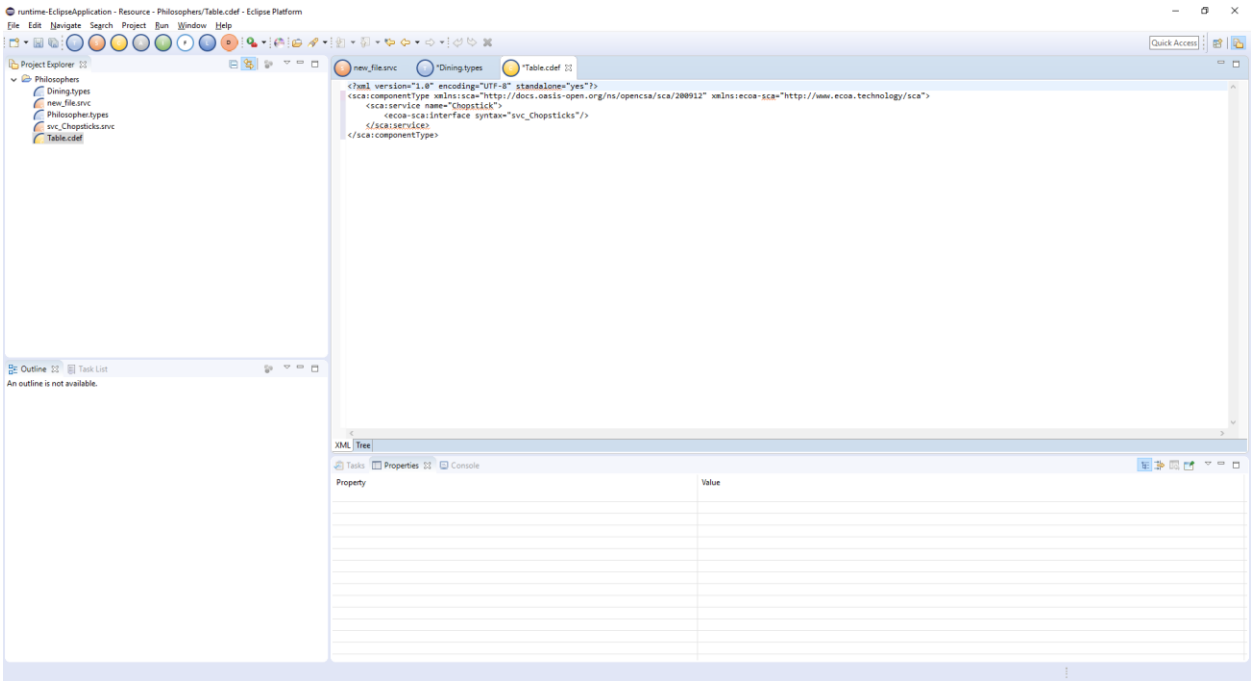
Select the tree tab at the bottom to move to Tree Editor to add the new Definitions. Select Provided Services and press Add Def



This will open the Definition editor. Enter the details and click Save (Use the Service dropdown to select from a list of defined services).



Verify the generated XML.

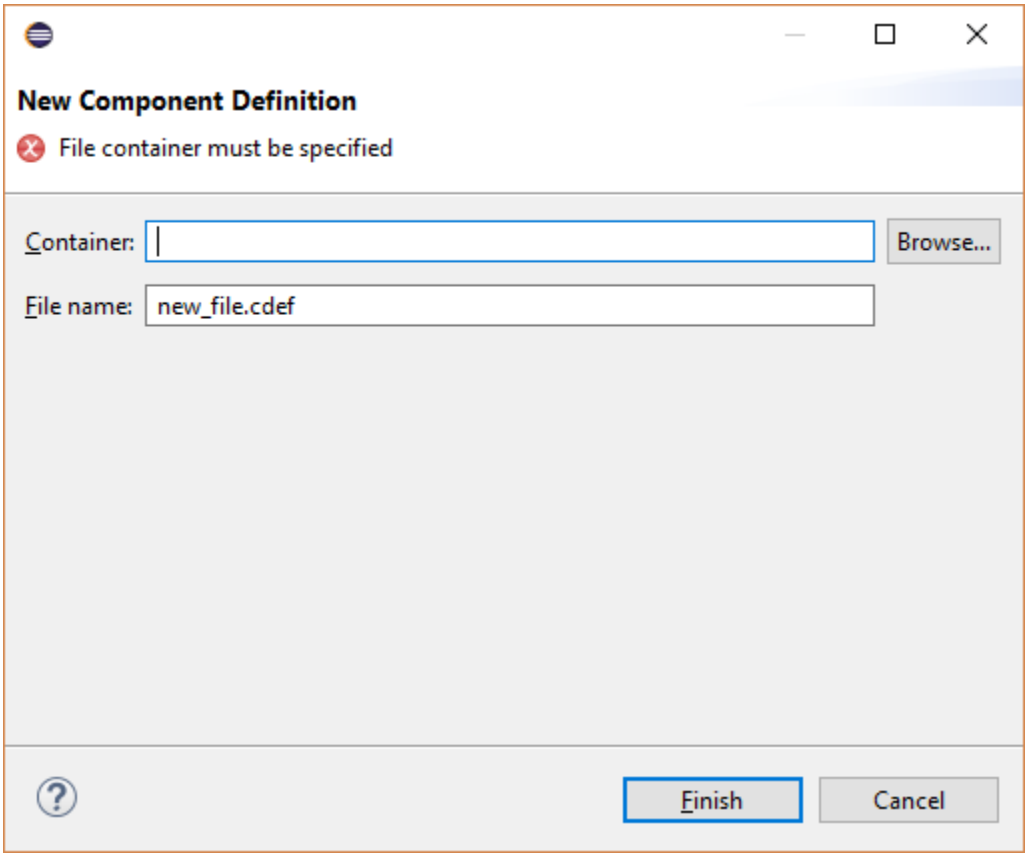


3.3.1.2 Philosopher

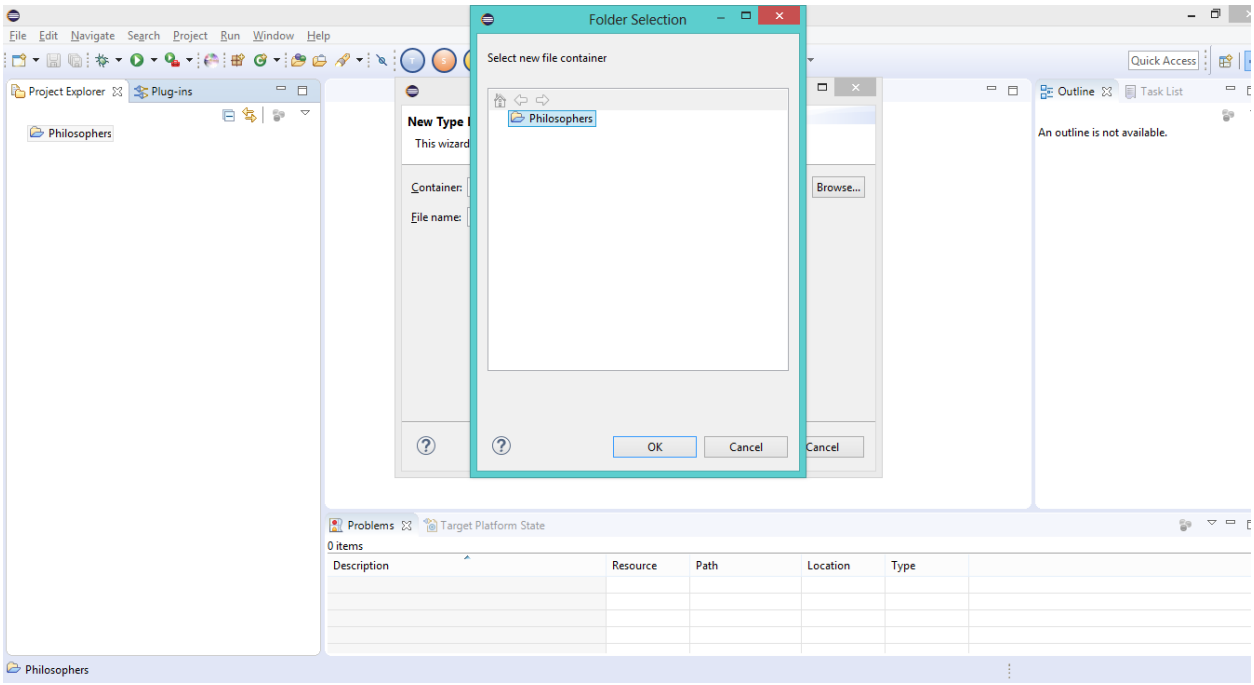
The Philosopher Component Definition requires a Service Chopsticks of type svc_Chopsticks. It also has a Property ID of Type Dining:Philosopher_id.

3.3.1.2.1 Process

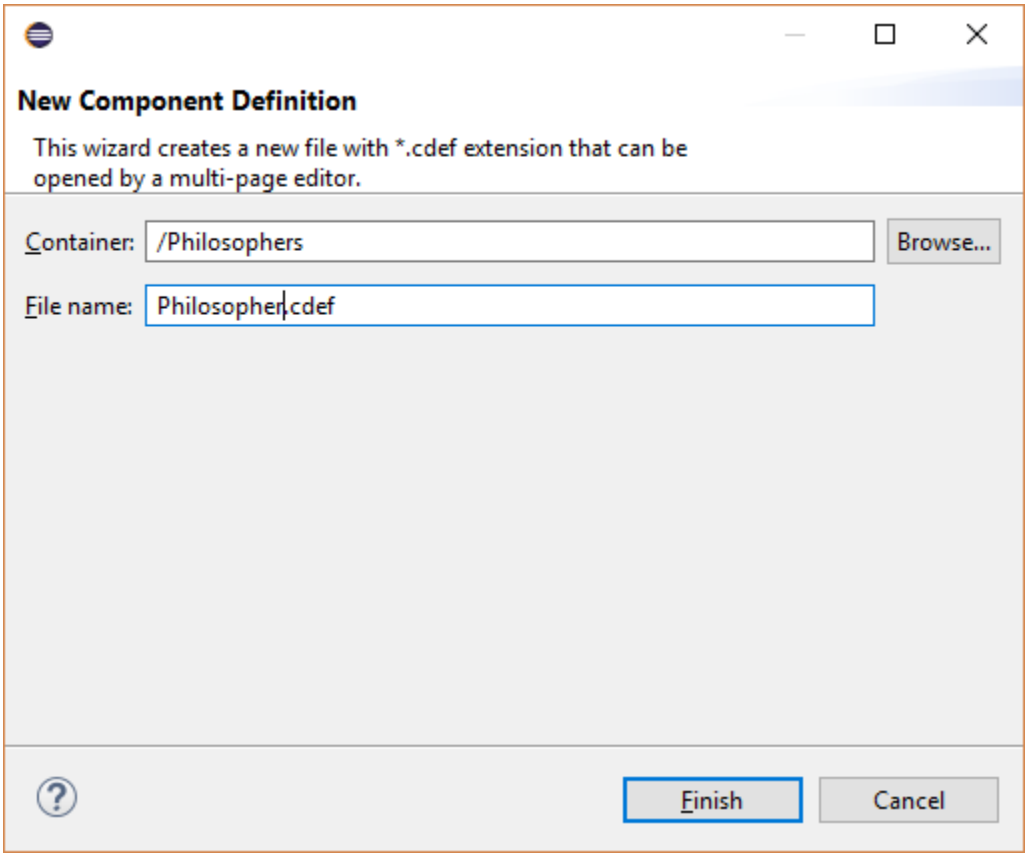
Select Component Definition Editor from the Toolbar which opens the below Wizard:



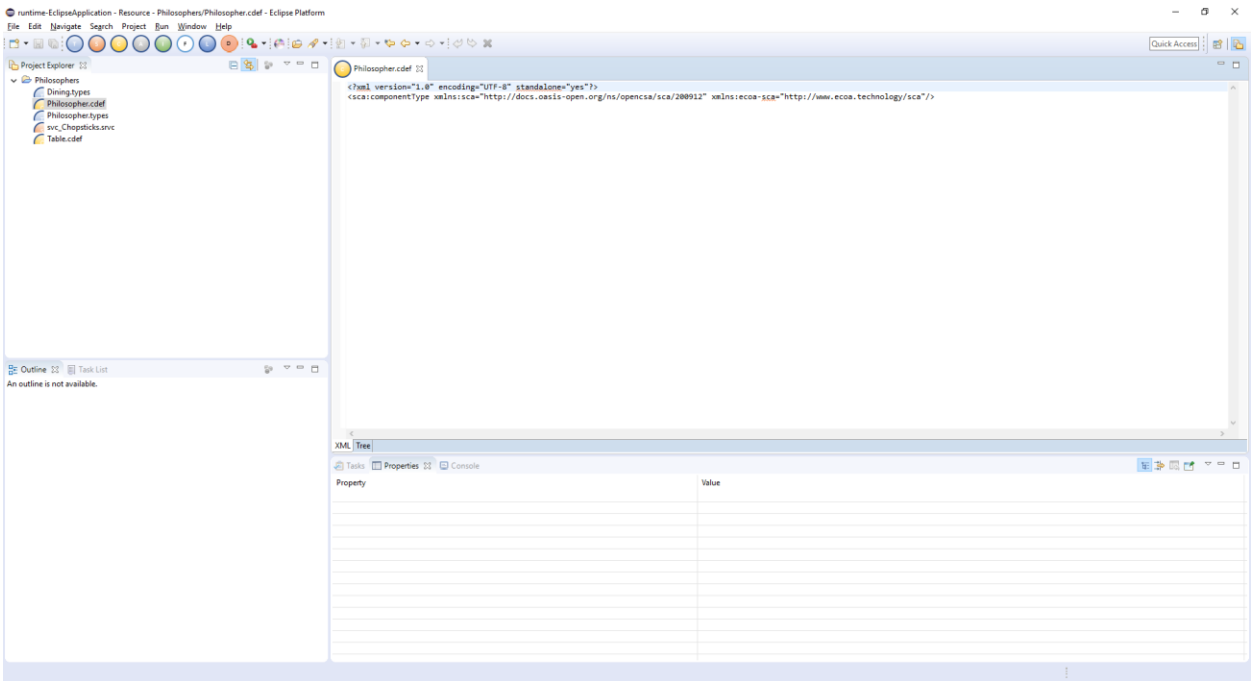
Select the container name as the top-level Project folder Philosophers



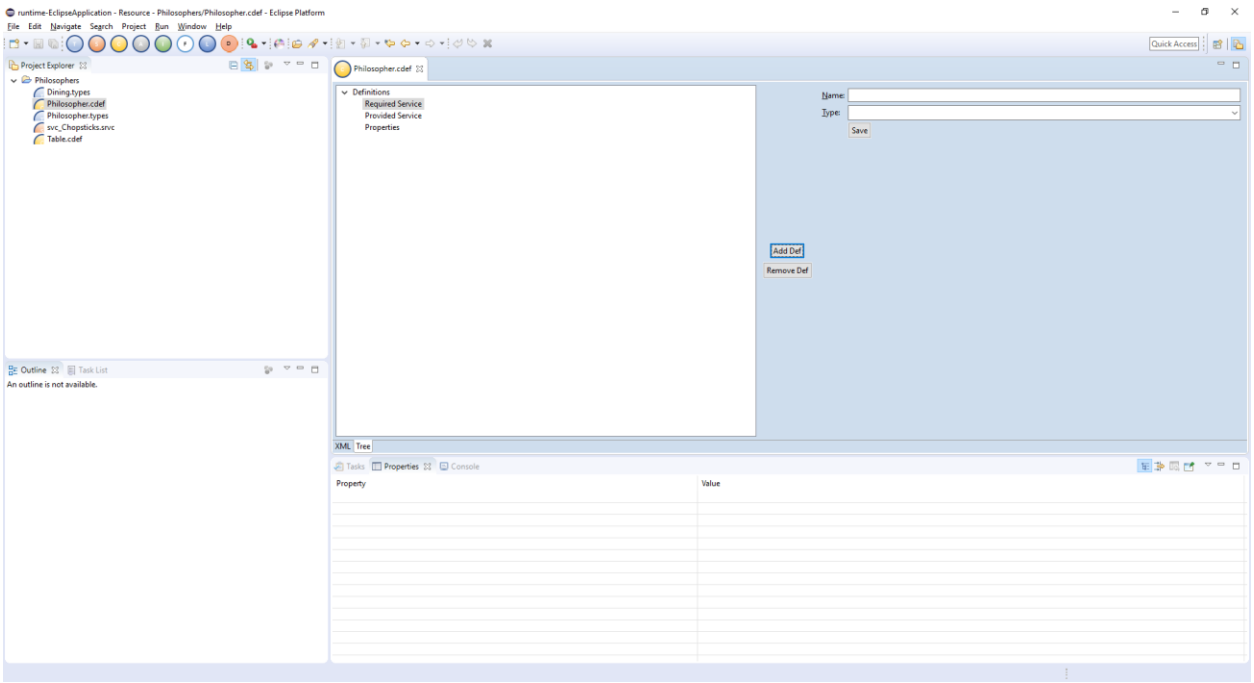
Select the file name as Philosopher.cdef and Click on Finish.



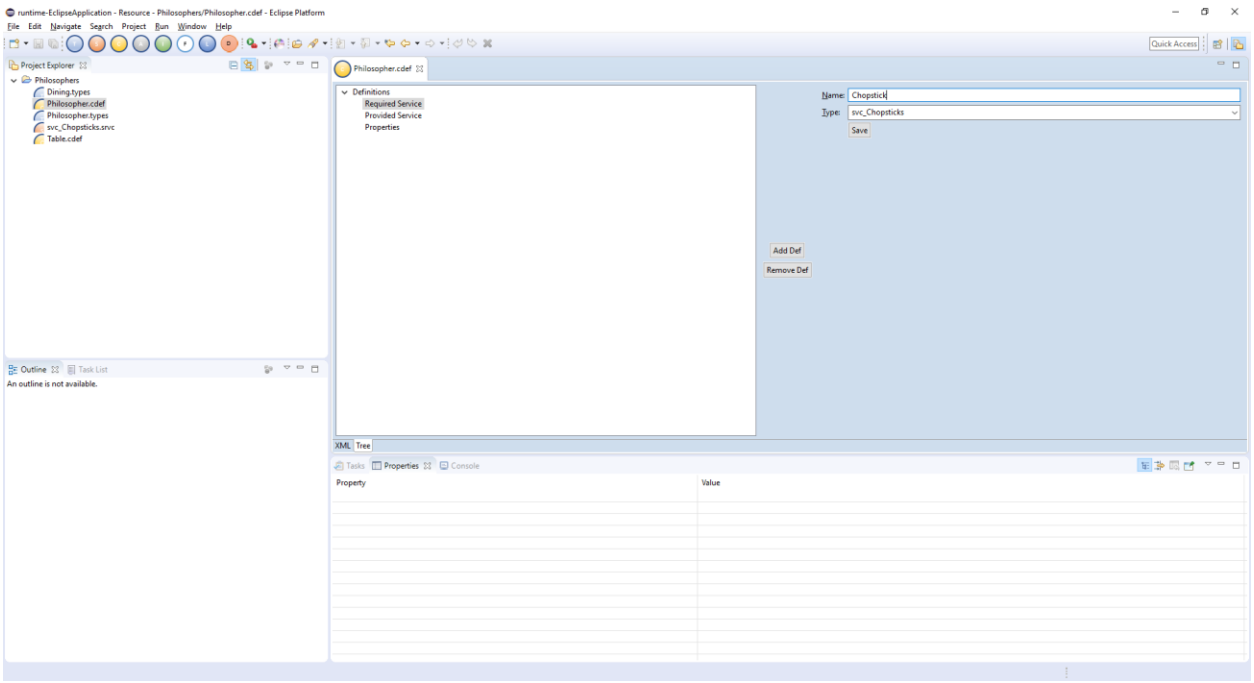
The initial XML Definition is shown.



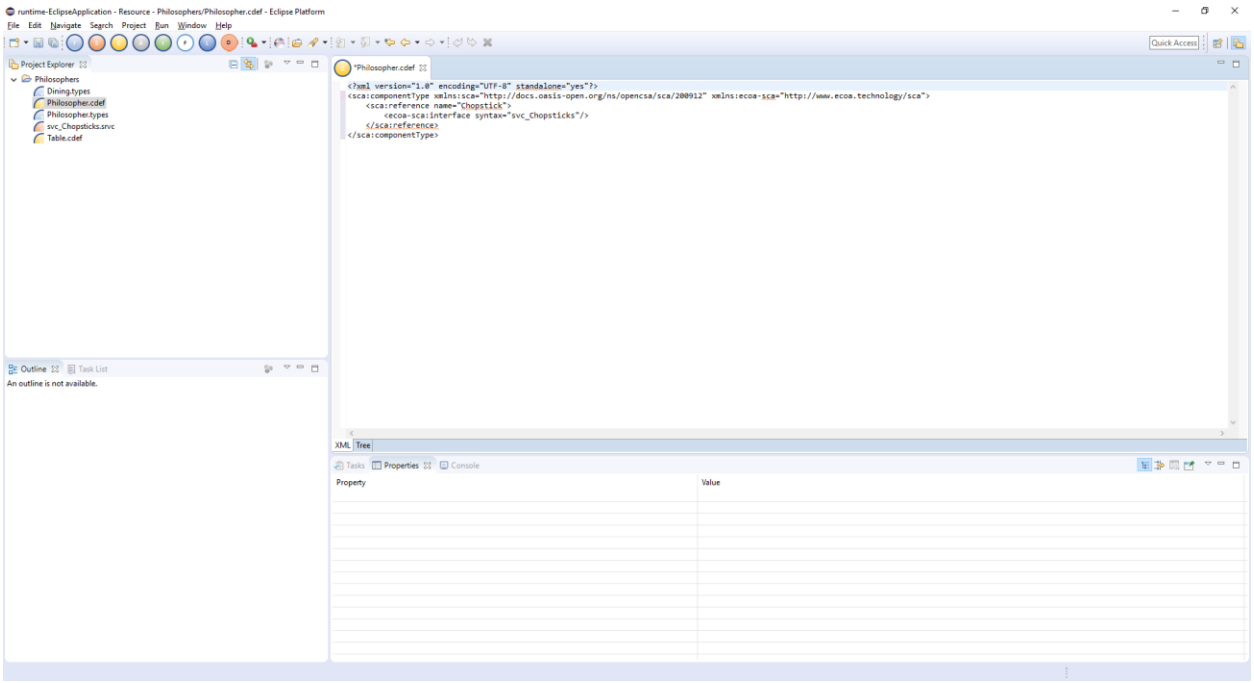
Select the tree tab at the bottom to move to Tree Editor to add the new Definitions. Select Required Services and press Add Def



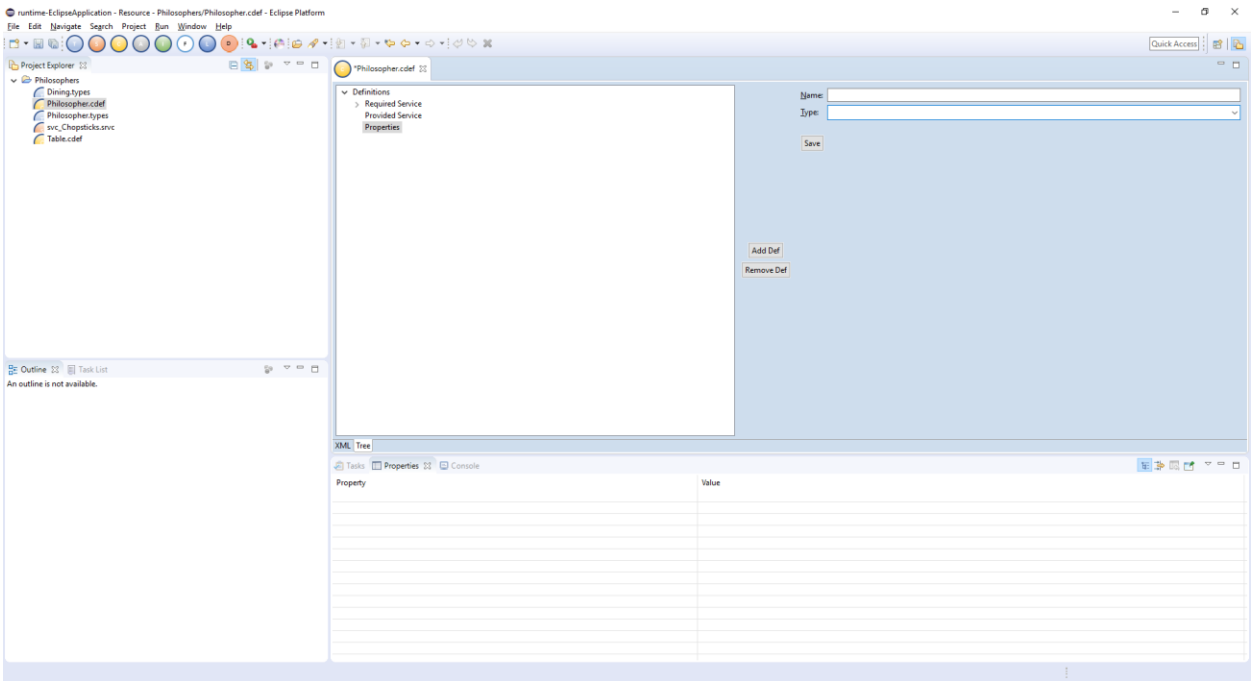
This will open the Definition editor. Enter the details and click Save (Use the Service dropdown to select from a list of defined services).



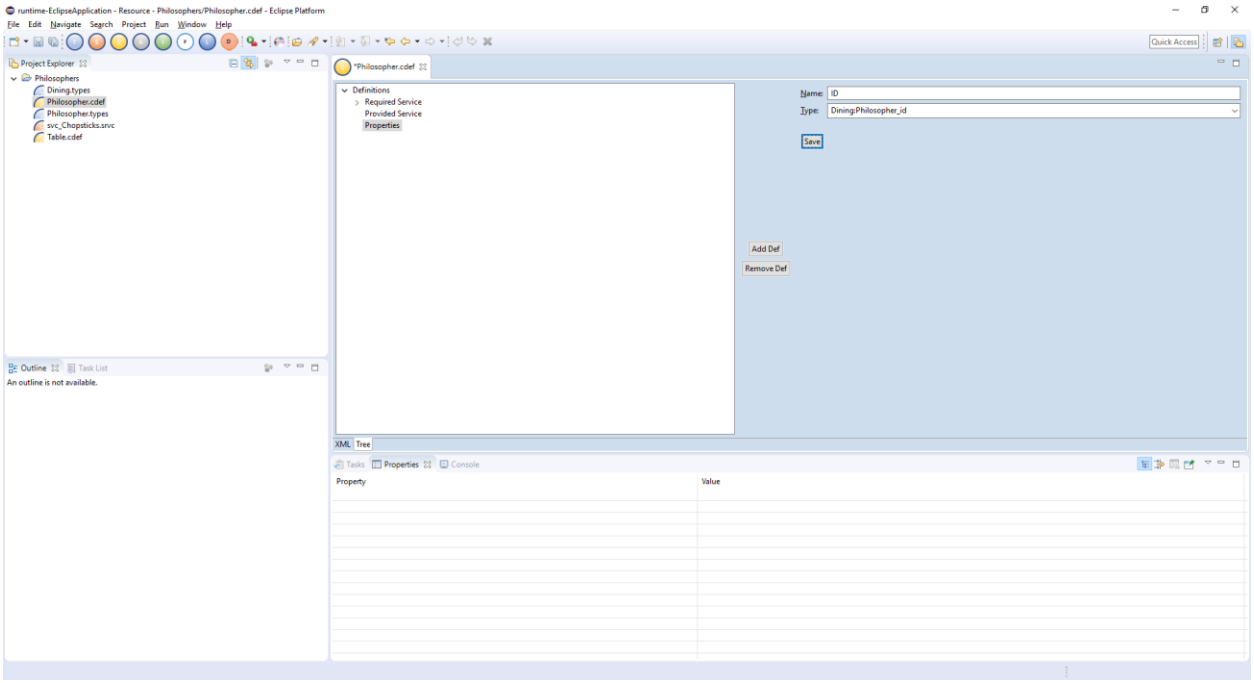
Verify the generated XML.



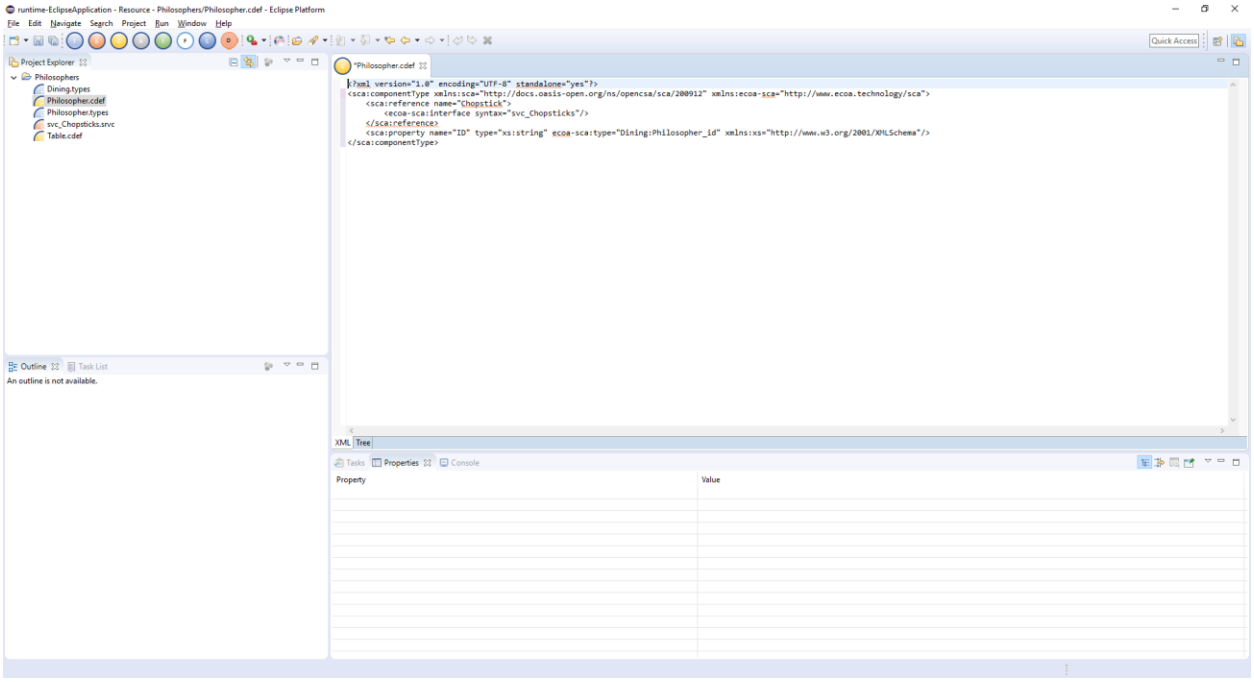
Select the tree tab at the bottom to move to Tree Editor to add the new Definitions. Select Properties and press Add Def



This will open the Definition editor. Enter the details and click Save (Use the Service dropdown to select from a list of defined services).



Verify the generated XML.


runtime-EclipseApplication - Resource - Philosophers/Philosopher.cdef - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Quick Access
Project Explorer
Philosophers
Dining.types
Philosopher.cdef
Philosopher.types
svc_Chopsticks.srvc
Table.cdef
Outline
Task List
An outline is not available.
*Philosopher.cdef
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<sca:componentType xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912" xmlns:ecoa-sca="http://www.ecoa.technology/sca">
<sca:reference name="Chopstick">
<ecoa-sca:interface syntax="svc_Chopsticks"/>
</sca:reference>
<sca:property name="ID" type="xs:string" ecoa-sca:type="Dining:Philosopher_id" xmlns:xs="http://www.w3.org/2001/XMLSchema"/>
</sca:componentType>
XML Tree
Tasks Properties Console
Property
Value

4 Graphical Editors

This section explains about the GEF Graphical editors used for the rest of the ECOA Constructs. There is a requirement for Graphical editors for these constructs as they must be shown as interlinked and dependant figures.

4.1 Initial Assembly Editor

ECOA terms Initial Assembly as a construct linking dependent components. All or a subset of defined components with in a project can be pulled into an Initial Assembly editor and can be linked with Wires. The basic components of an Initial Assembly Editor are:

1) Composites
2) Composite Properties
3) Components
4) Component Properties
5) Wires

4.1.1 Scenario

The Dining Philosophers Problem, defines Restaurant as an Initial Assembly where Table Component is instantiated as Our_Table and Philosopher is instantiated as Plato, Aristotle, Socrates, Epicurus, and Confucius. The Component property of Philosopher Component is pulled in and each of the Philosopher is assigned an Id from 1 thru 5. The Assembly is constructed as below:

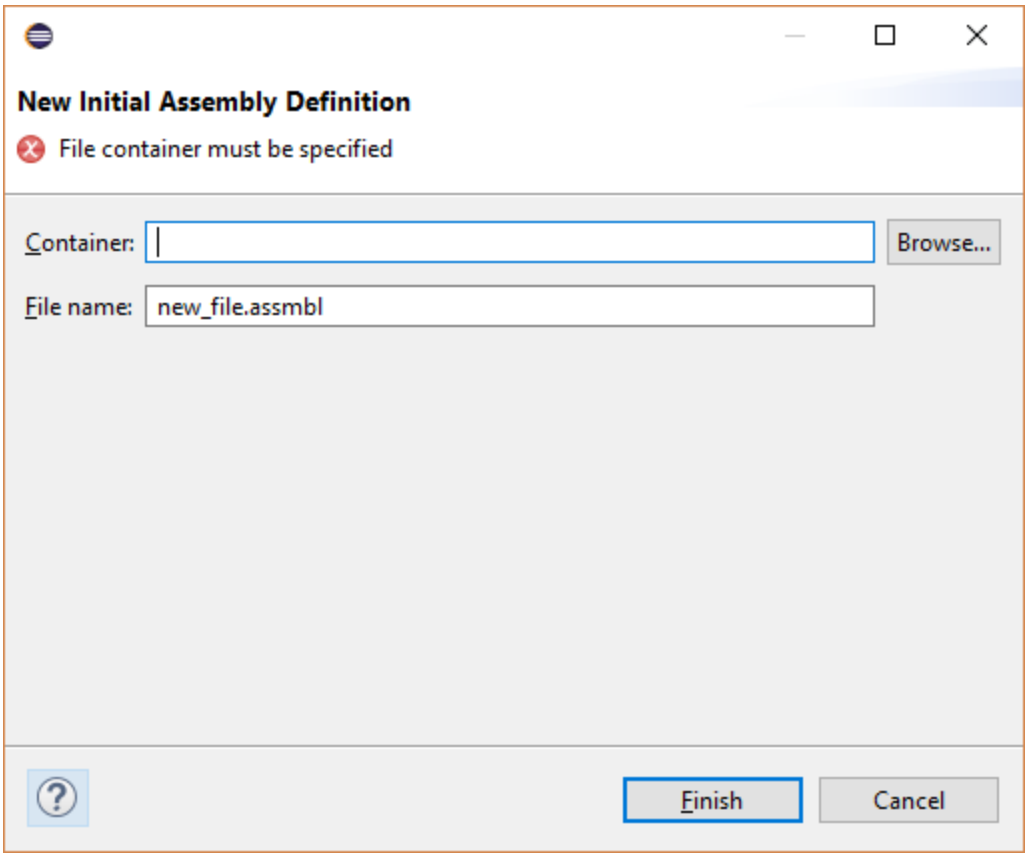
4.1.1.1 Restaurant Assembly

The Assembly consists of:

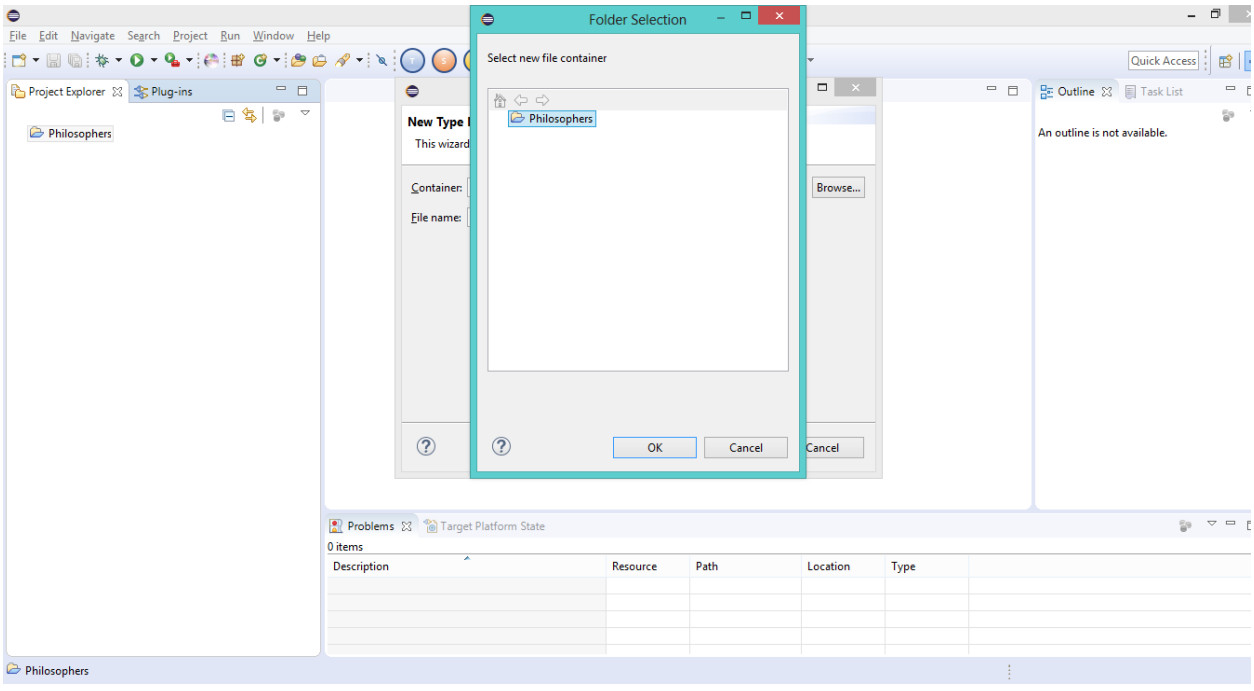
- 5 - Philosophers
- 1 - Table

4.1.1.1.1 Process

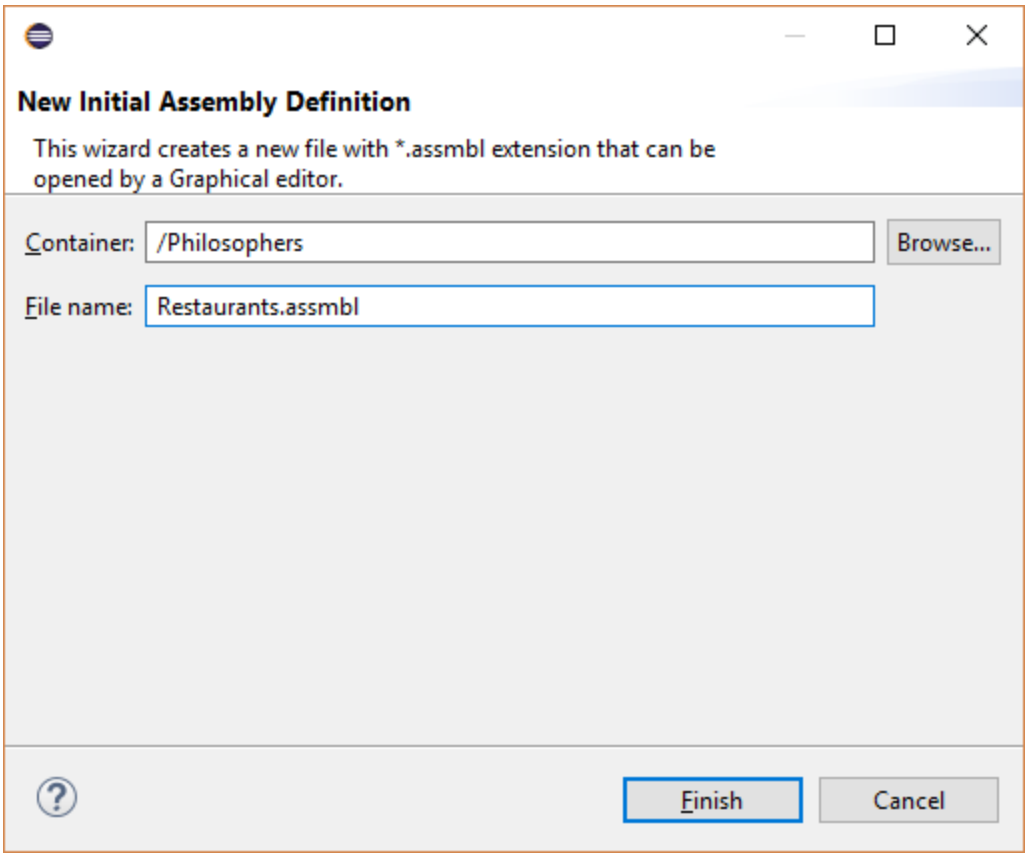
Select Initial Assembly Editor from Tool bar which opens the below wizard.



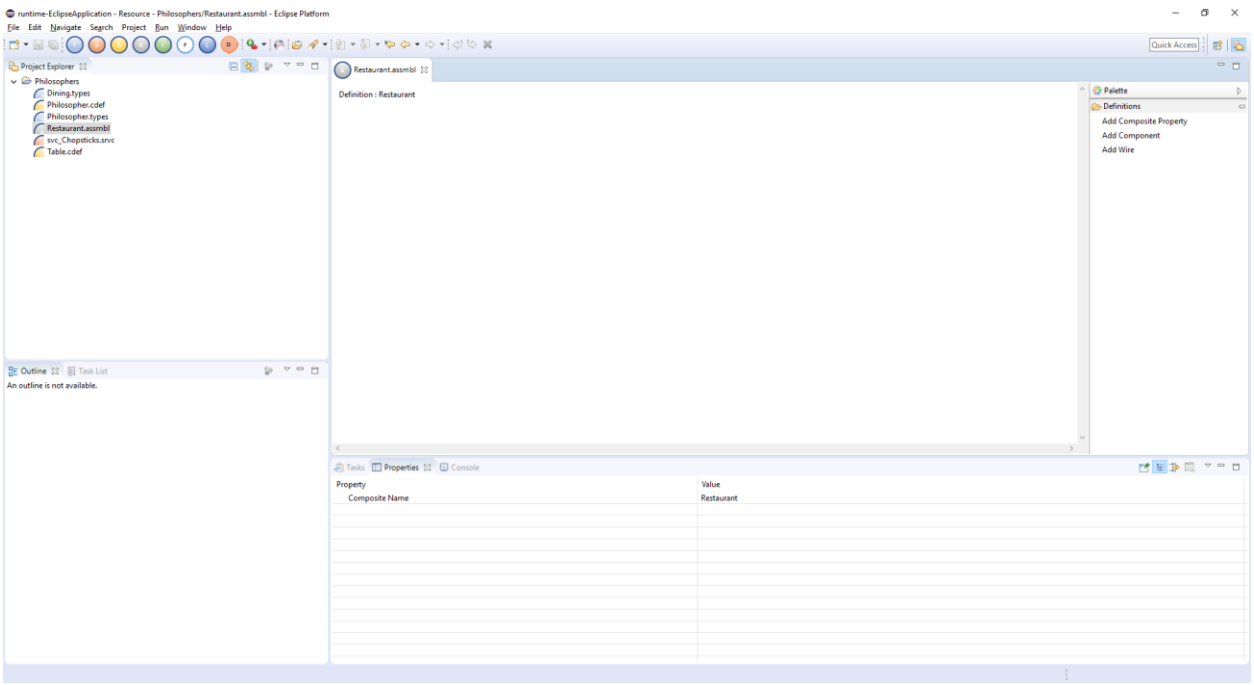
Select the container name as the top-level Project folder Philosophers



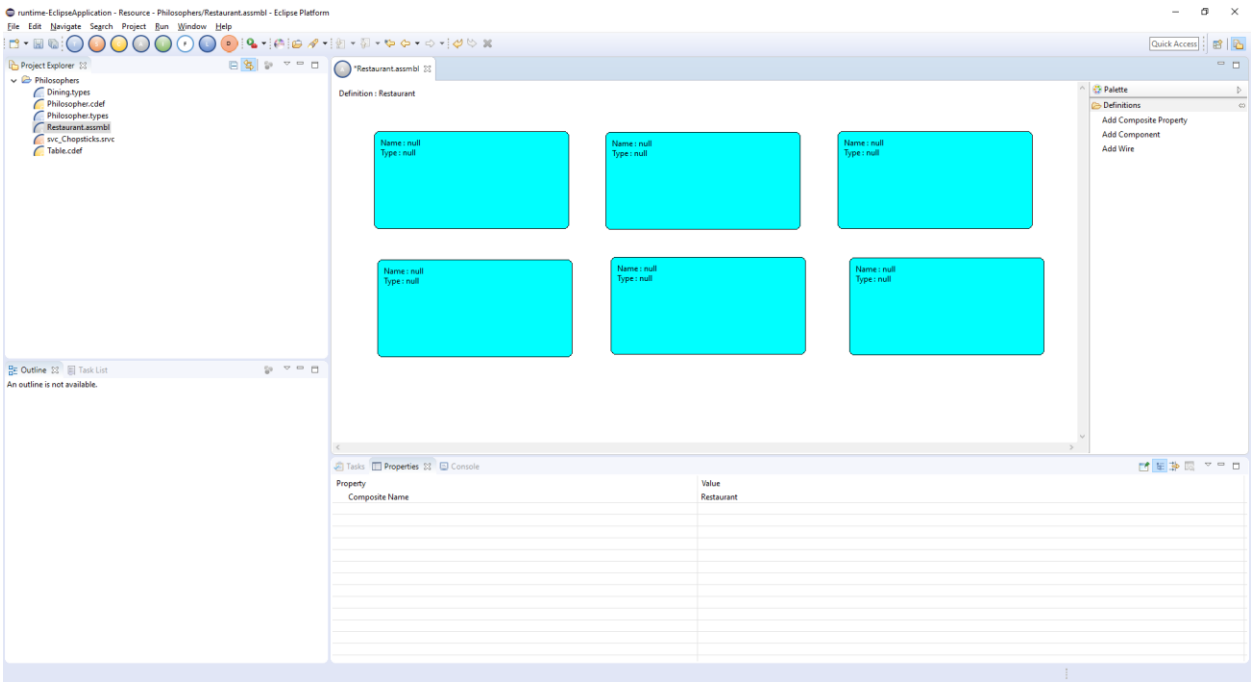
Change the file name to Restaurant.assmbl.



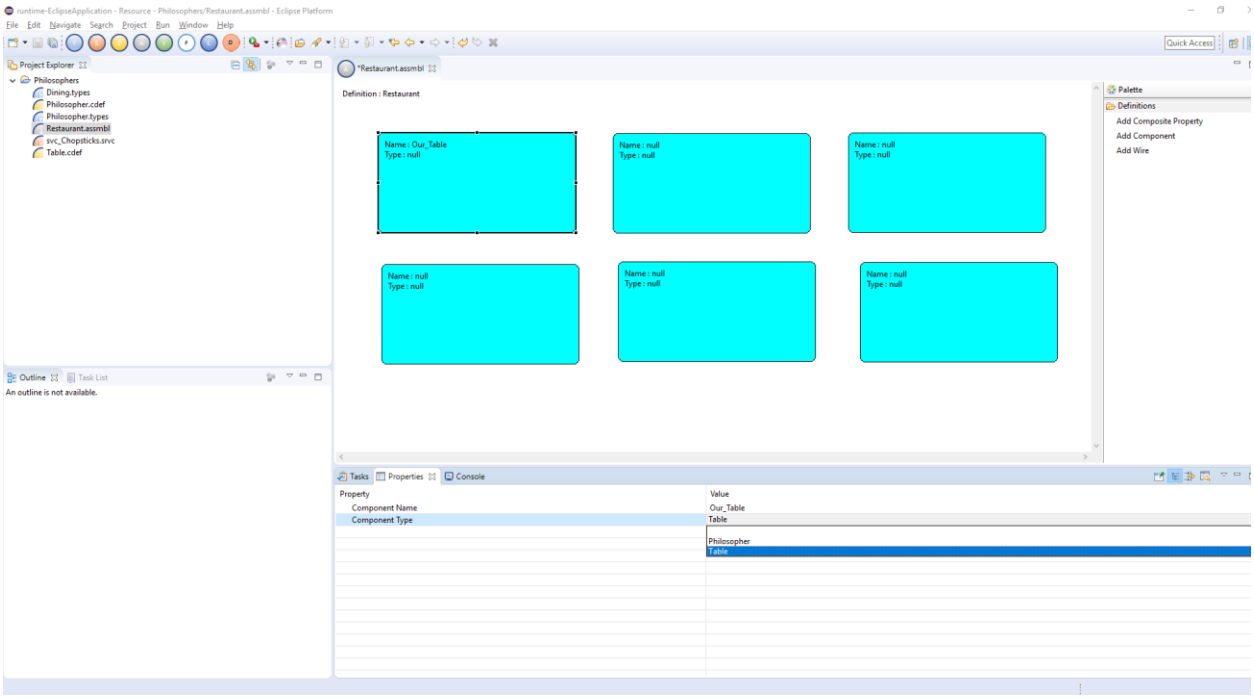
The Initial Screen is as below (Make sure that the Palette is drawn out for us to be able to add design elements):



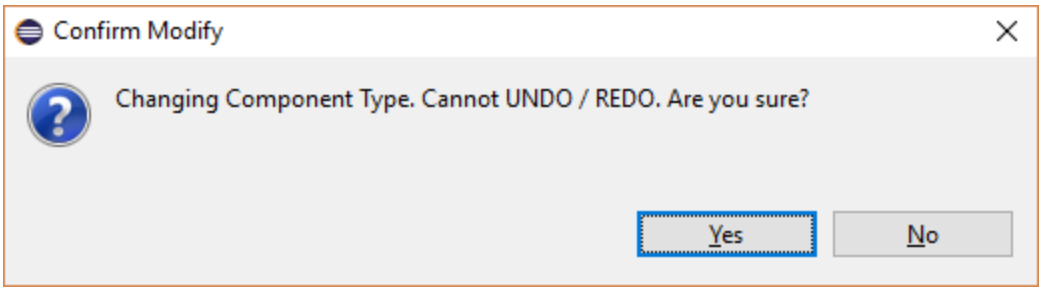
Add 5+1 Components from the palette onto the canvas.



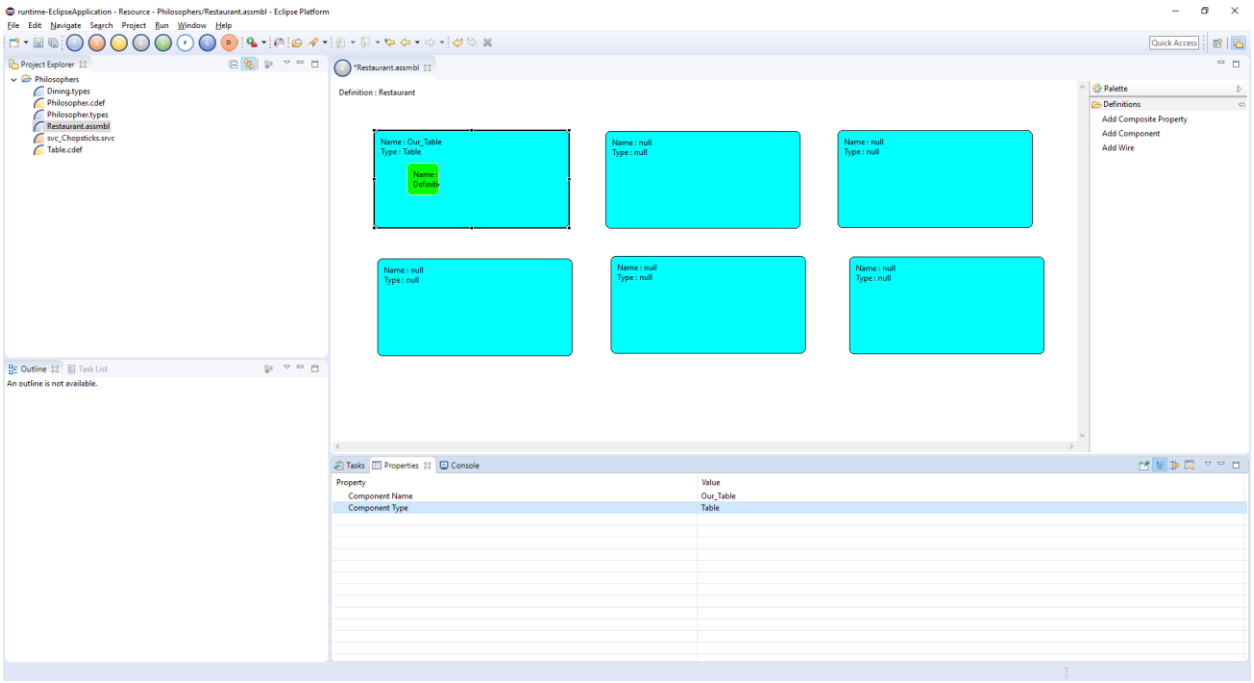
Double click on a Component to open the Properties Editor. Specify Component Name as Our_Table and Component Type as Table (select from drop down).



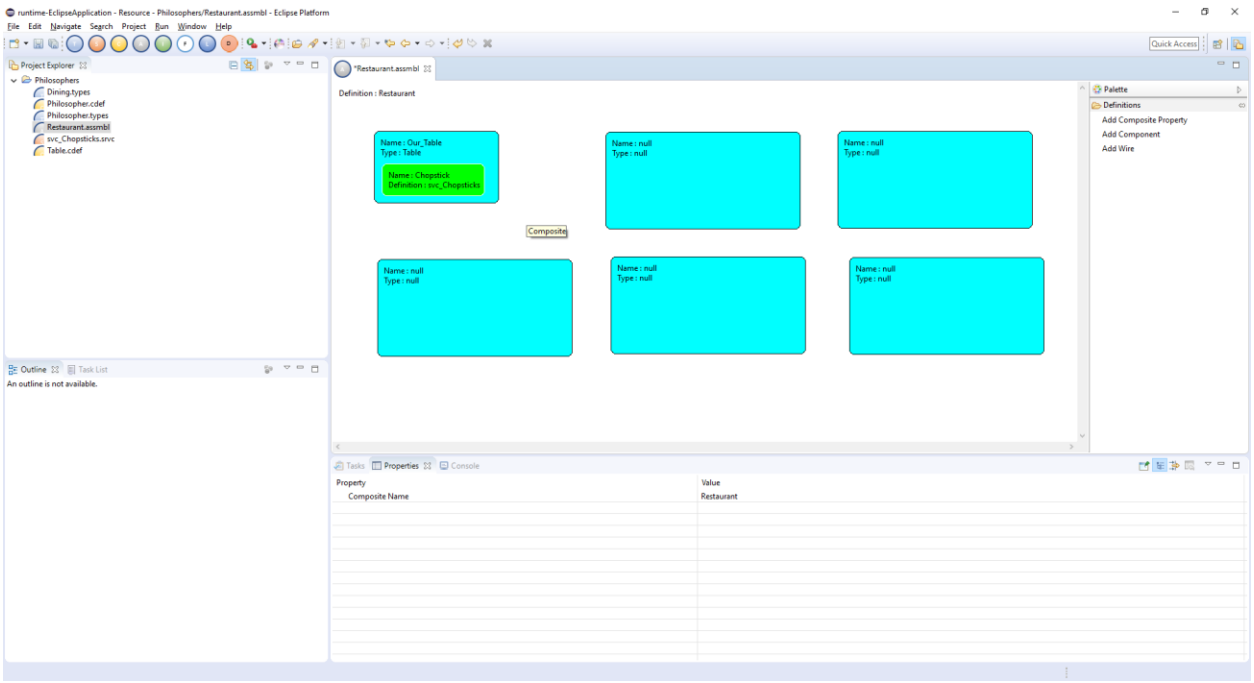
Confirm the Dialog to pull the Component information from Component Definition.



The Provided and Required Services along with the properties are pulled into the Initial Assembly canvas and are displayed using default widths.



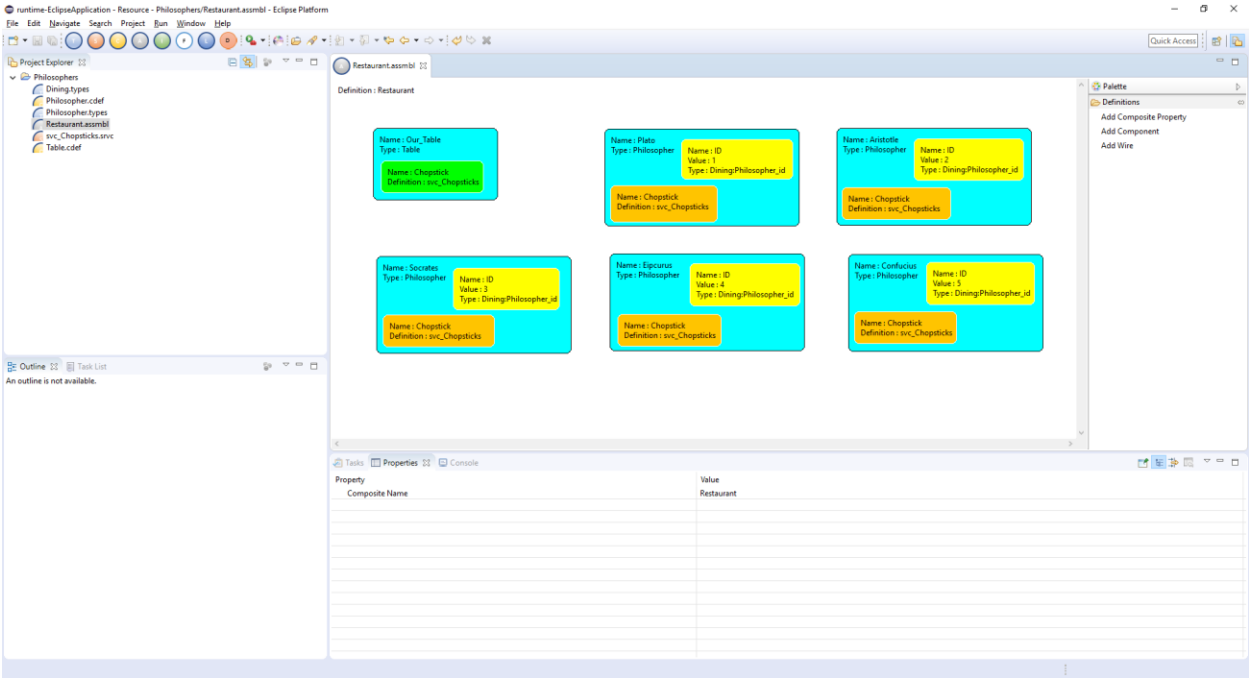
Re-size the component block to make the information visible.



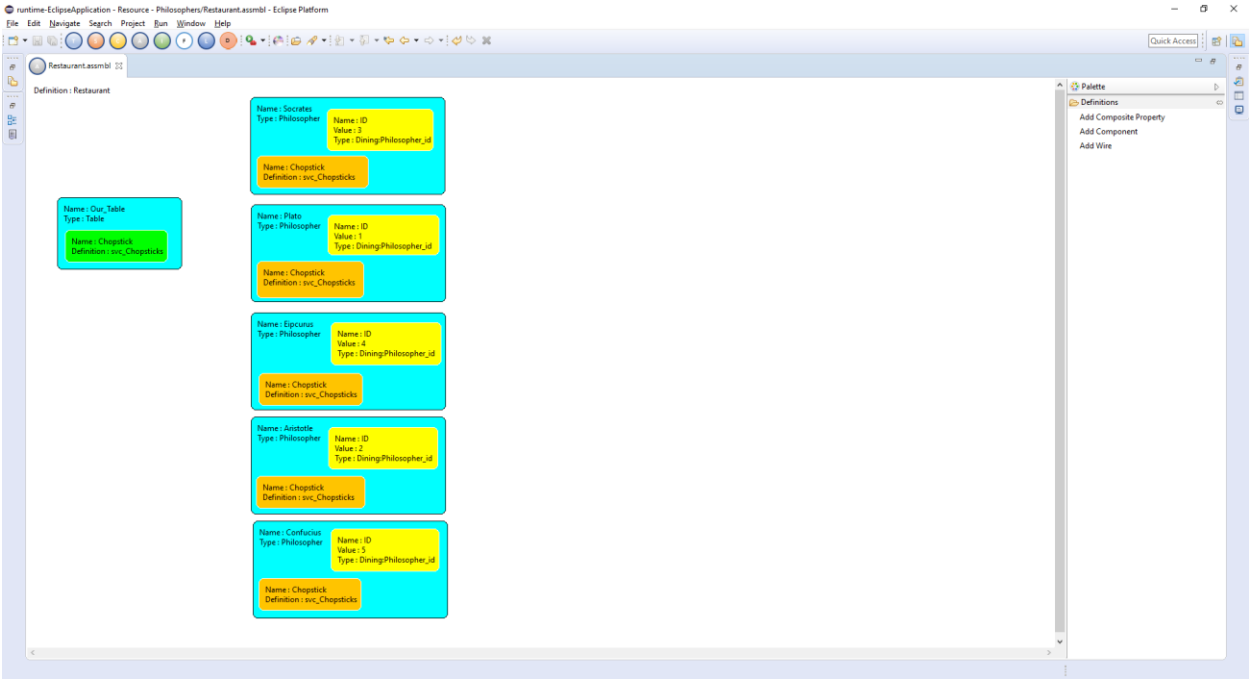
Repeat the process for the remaining component definitions as below:

- Plato – Philosopher
- Aristotle – Philosopher
- Socrates – Philosopher
- Epicurus – Philosopher
- Confucius – Philosopher

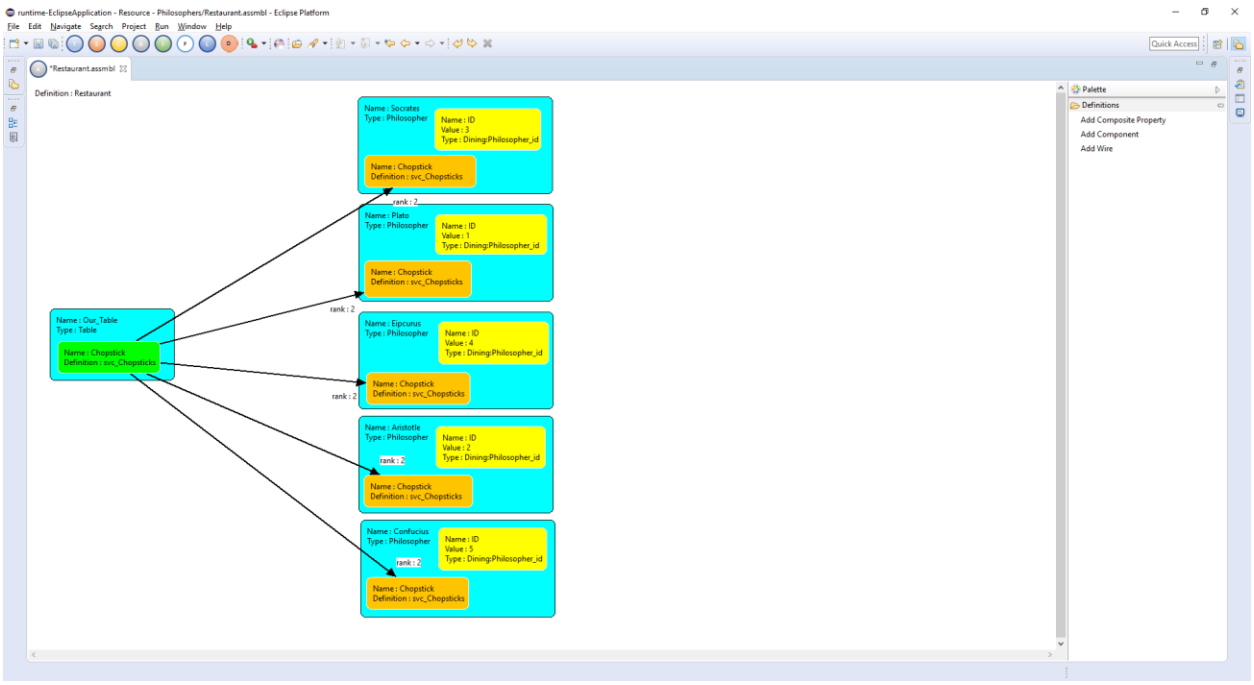
The below is the resultant Canvas.



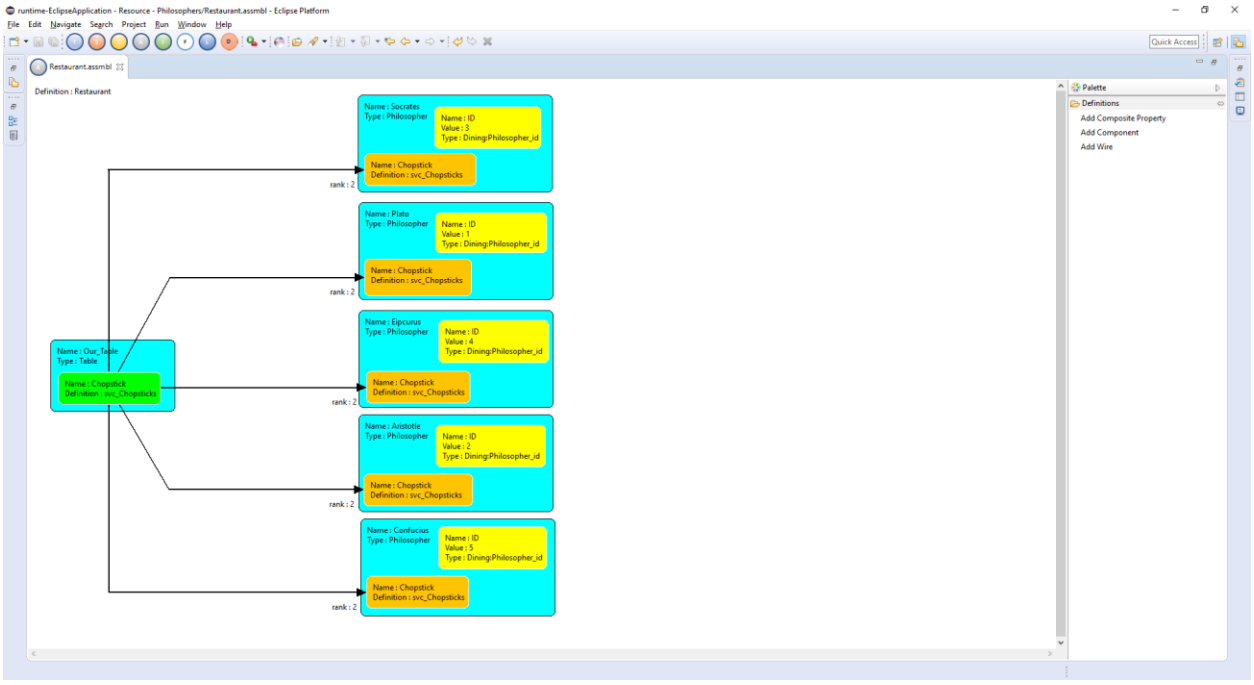
Reset the Canvas to Allow for Wiring:



Select Wire from the drawn-out palette and connect the Table Provided Service with Philosophers Required Services.



The Wires initially are connected as direct straight lines. You can create bend points on the line to make the diagram more presentable. Each bend point is located mid-way of the line and you can hold and drag it until you feel that the diagram is complete.



4.2 Component Implementation Editor

ECOA terms Component Implementation as a construct to define the Implementation of a defined component. The Basic constructs of a Component Implementation Editor are:

1) Imported Service / Service Operation References
2) Triggers
3) Dynamic Triggers
4) Module Types
5) Module Implementation
6) Module Instances
7) Module Properties
8) Module Operations
9) Module Operation Parameters
10) Wires (Classified as Request, Event and Data Links)

4.2.1 Scenario

The Dining Philosophers problem defines Philosopher_impl and Table_impl as Component Implementations.

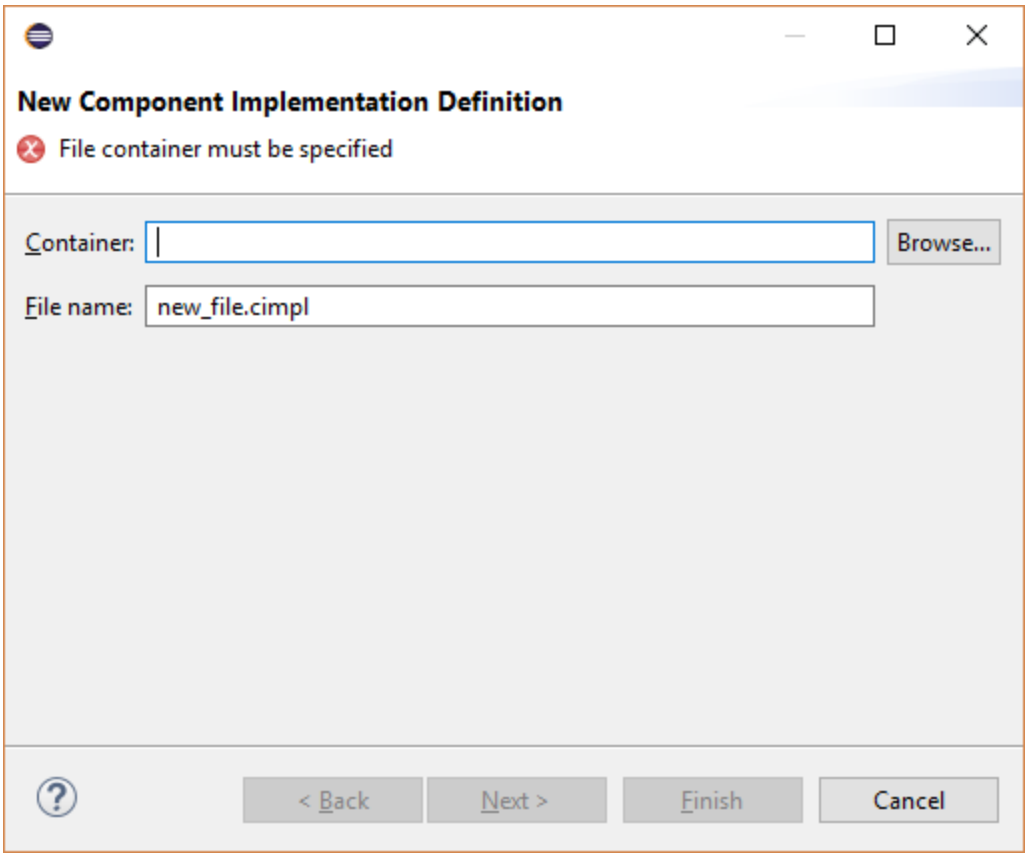
4.2.1.1 Table_Impl

The Component Implementation of Table consists of:

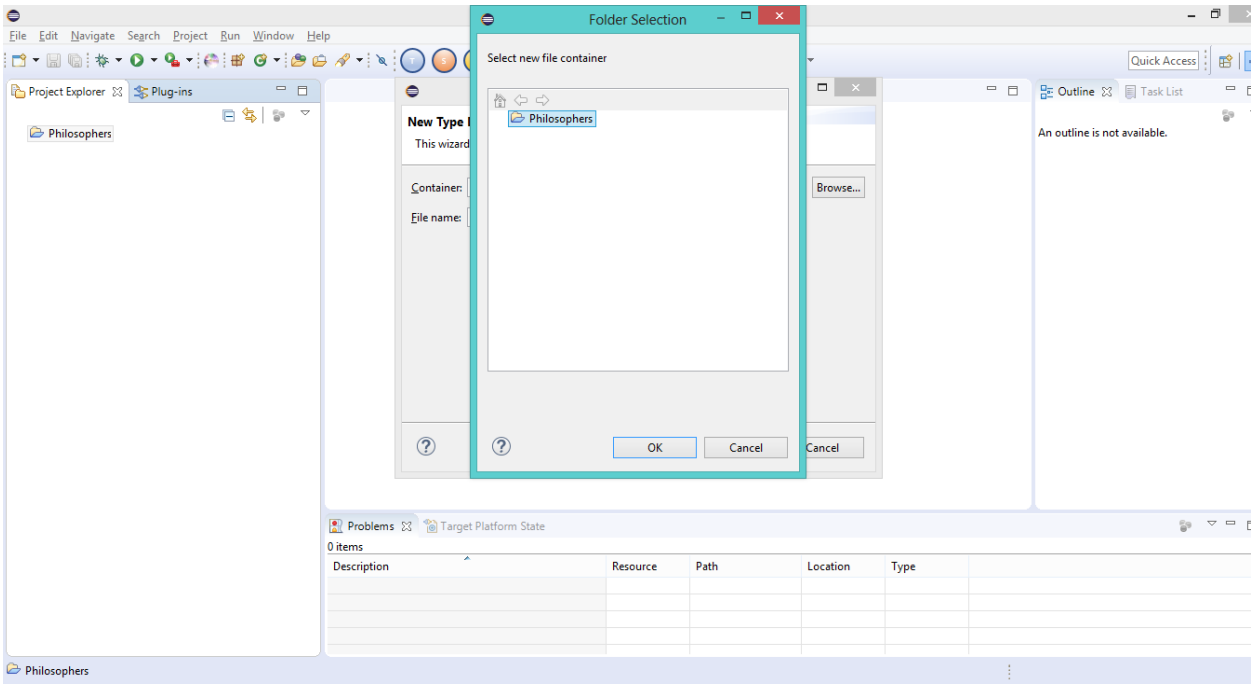
1) svc_Chopsticks service with take and surrender service operations is imported from Component Definition
2) Table_modMain_t as a Module Type with supervision enabled
3) Two Module Operations
    a. take – Type Request Received with Parameters which – Dining:Chopstick_id and who – Dining:Philosopher_id as inputs and Parameter taken – boolean8 as output
    b. surrender – Type Request Received with Parameters which – Dining:Chopstick_id and who – Dining:Philosopher_id as inputs
4) Table_modMain_C as a Module Implementation with Language as C
5) Table_modMain_Instance as a Module Instance of that Implementation
6) Wiring done for:
    a. Table_modMain_t/take with svc_Chopsticks/take
    b. Table_modMain_t/surrender with svc_Chopsticks/surrender

4.2.1.1.1 Process

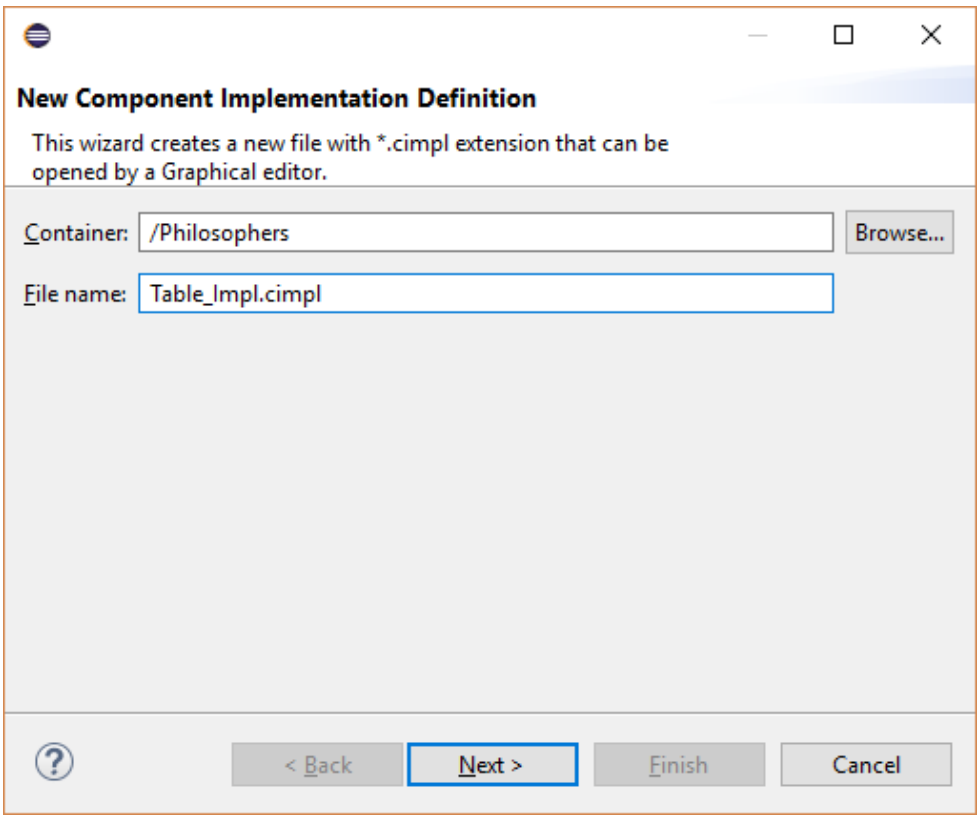
Select Component Implementation Editor from Tool bar which opens the below wizard.



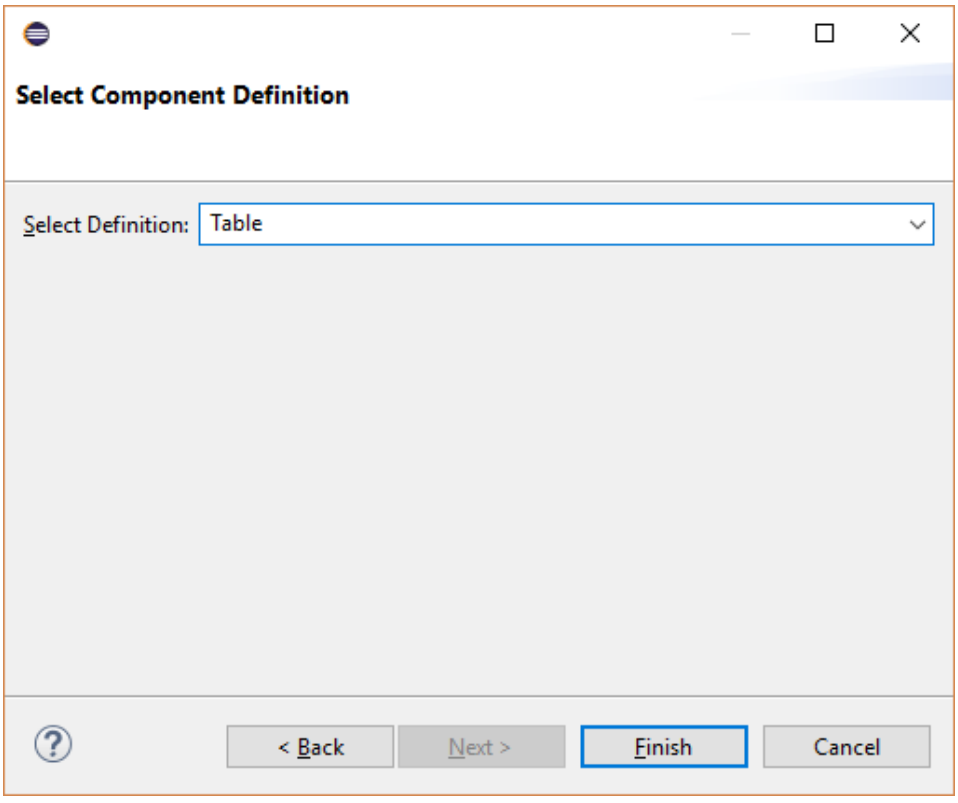
Select the container name as the top-level Project folder Philosophers



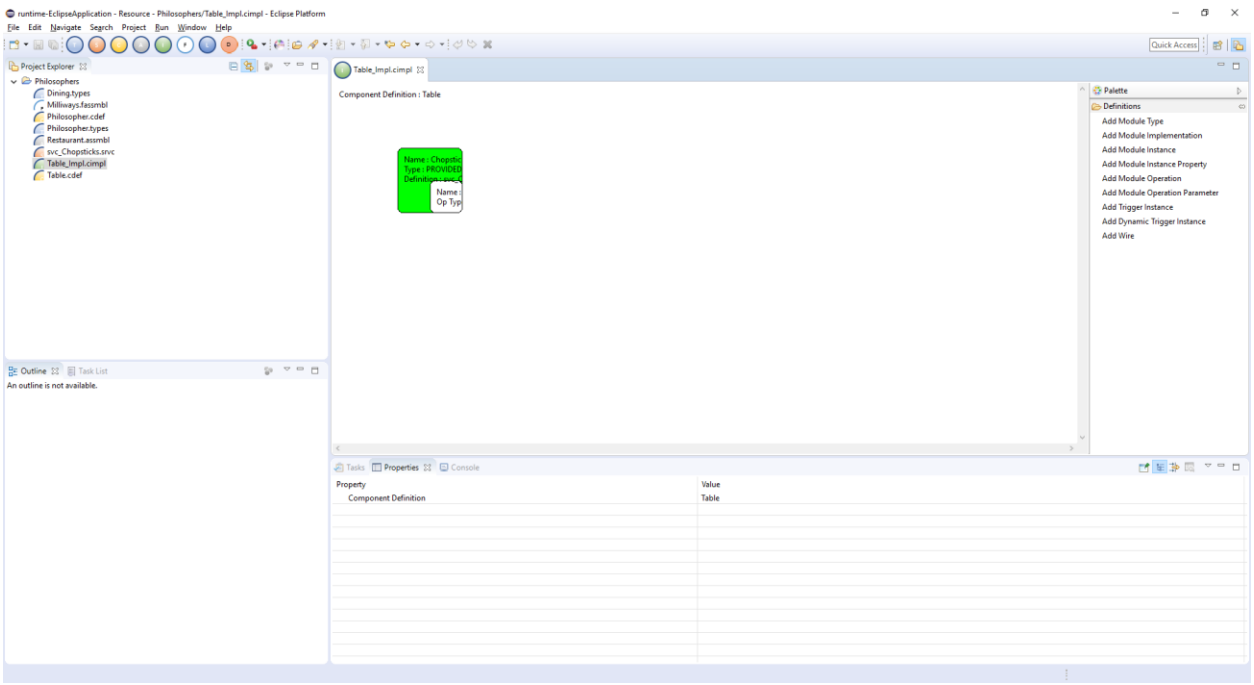
Change the file name to Table_Impl.cimpl.



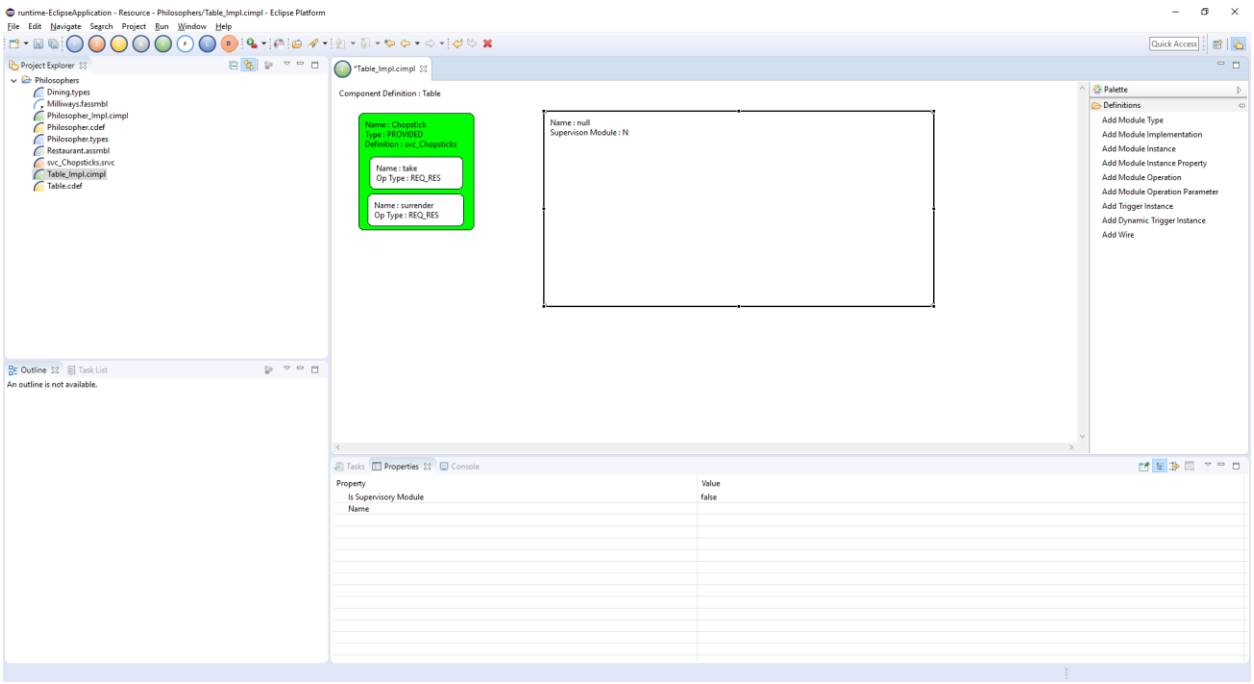
Select Next to pick the Table component definition for the implementation and click Finish



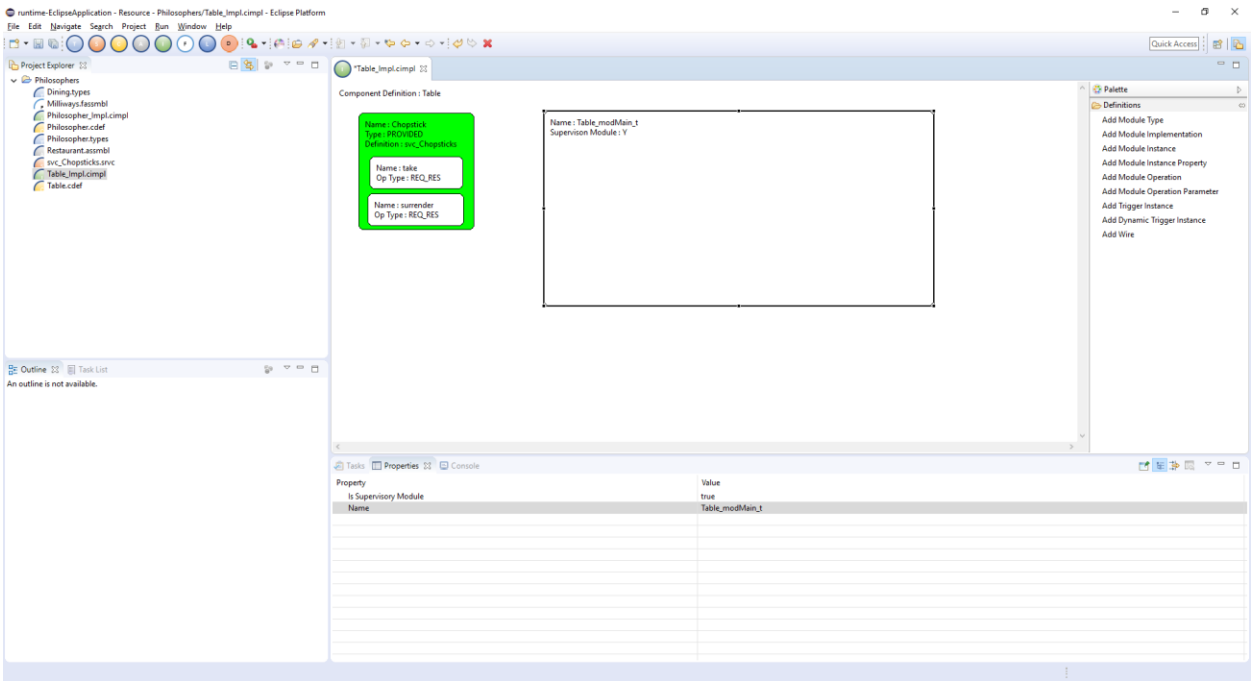
The Initial Screen is as below (Make sure that the Palette is drawn out for us to be able to add design elements):



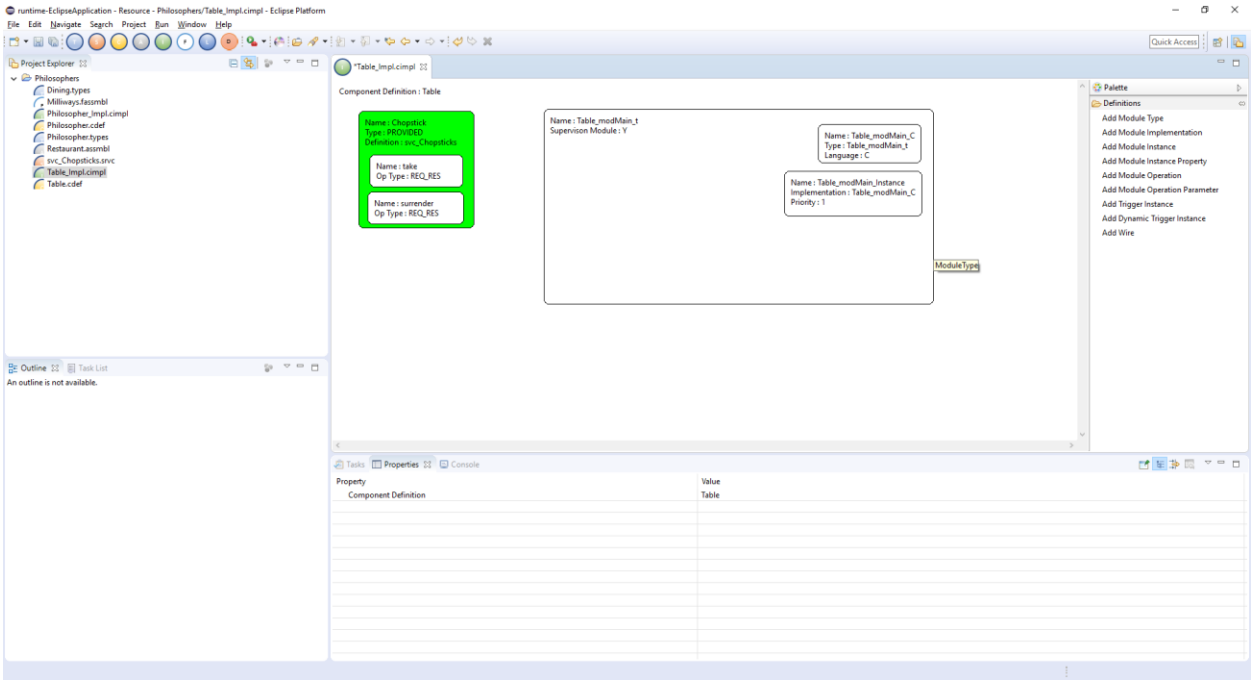
Rearrange the components for visibility. Add Module Type from Side Palette.



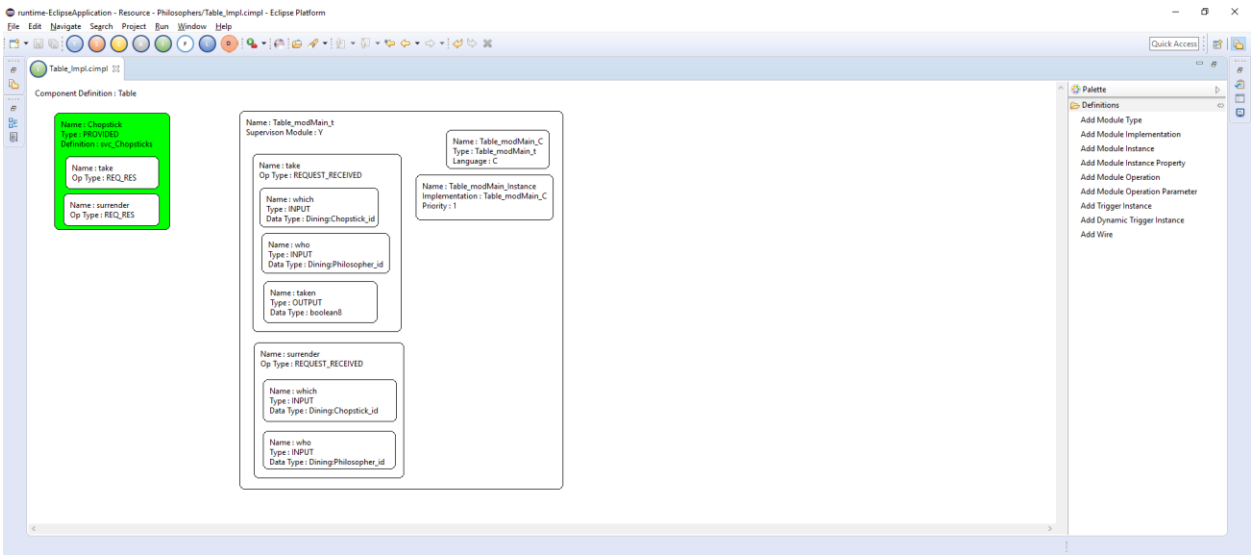
This adds an empty Module type. Using Properties modify the Name to Table_modMain_t and Supervisory Module to true



Add the Module Implementation and Module Instance constructs from the palette and set the names to Table_modMain_C and Table_modMain_Instance. Enter other property details and select Table_modMain_C as the Implementation on the instance.



Add the Module Operations take (with Parameters which, who, and taken) and surrender (with parameters (which and who).

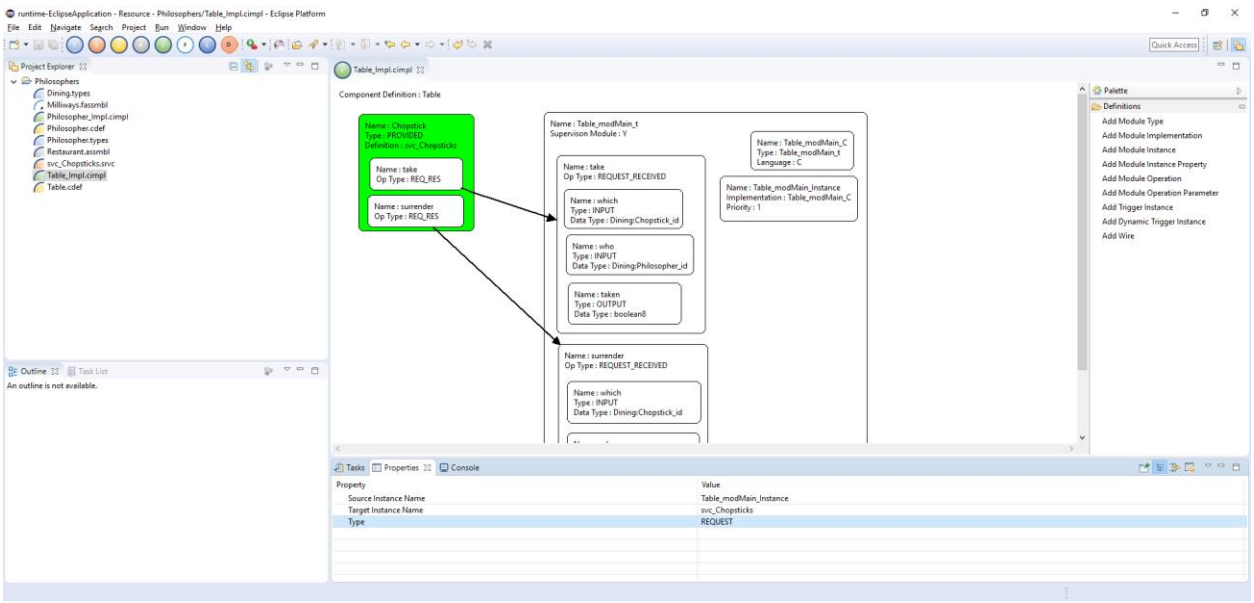


Complete the wiring by selecting the wire from palette and connect Module Instance Operations to the Service Operations:

- Chopsticks/take - Table_modMain_Instance/take
- Chopsticks/surrender - Table_modMain_Instance/surrender

*NOTE 1: Wires should be **started** from an Operation that is sending/writing something and **completed** on an Operation that is receiving/reading that thing. This is usually easiest to determine by checking the Module Operation type.*

*NOTE 2: To set which Module Instance (for a Module Type) a wire is connecting to the wire must be selected and the Module Instances set in its properties. Simple designs are likely to only have one Module Instance for each Module Type – so the choice is usually obvious. The design diagram does not make it apparent whether wires have had these properties set or not, so care must be taken to ensure this step is completed. **Forgetting to complete this step is an easy way to get code generation problems that are tricky to track down**.*



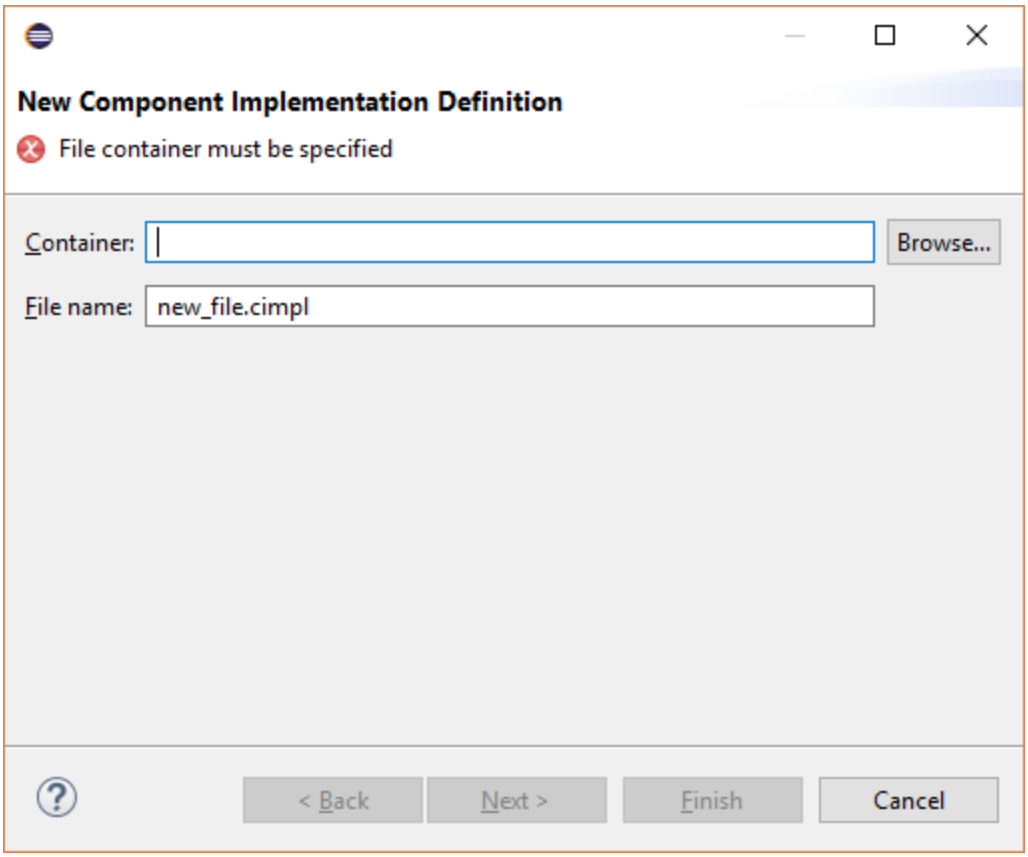
4.2.1.2 Philosopher_Impl

The Component Implementation of Philosopher consists of:

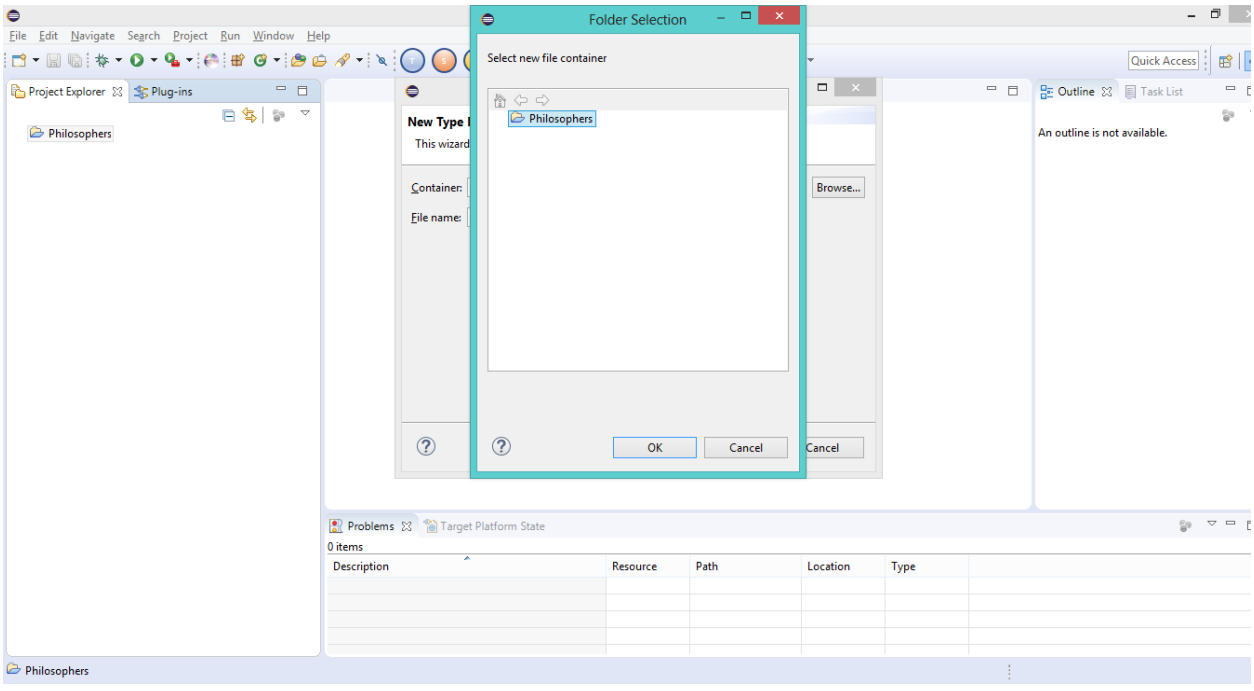
1) svc_Chopsticks service with take and surrender service operations is imported from Component Definition
2) Philosopher_modMain_t as a Module Type with supervision enabled
3) Three Module Operations
    a. take – Type Request Sent with Parameters which – Dining:Chopstick_id and who – Dining:Philosopher_id as inputs and Parameter taken – boolean8 as output
    b. surrender – Type Request Sent with Parameters which – Dining:Chopstick_id and who – Dining:Philosopher_id as inputs
    c. tick – Type Event Received
4) Philosopher_modMain_C as a Module Implementation with Language as C
5) Philosopher_modMain_Instance as a Module Instance of that Implementation
6) Add Module Instance Property Id of type Dining:Philosopher_id and set the value to $ID
7) Add Trigger Instance Clock with Relative Priority 1
8) Wiring done for:
    a. Philosopher_modMain_t/take with svc_Chopsticks/take
    b. Philosopher_modMain_t/surrender with svc_Chopsticks/surrender
    c. Clock/out with Philosopher_modMain_t/tick

4.2.1.2.1 Process

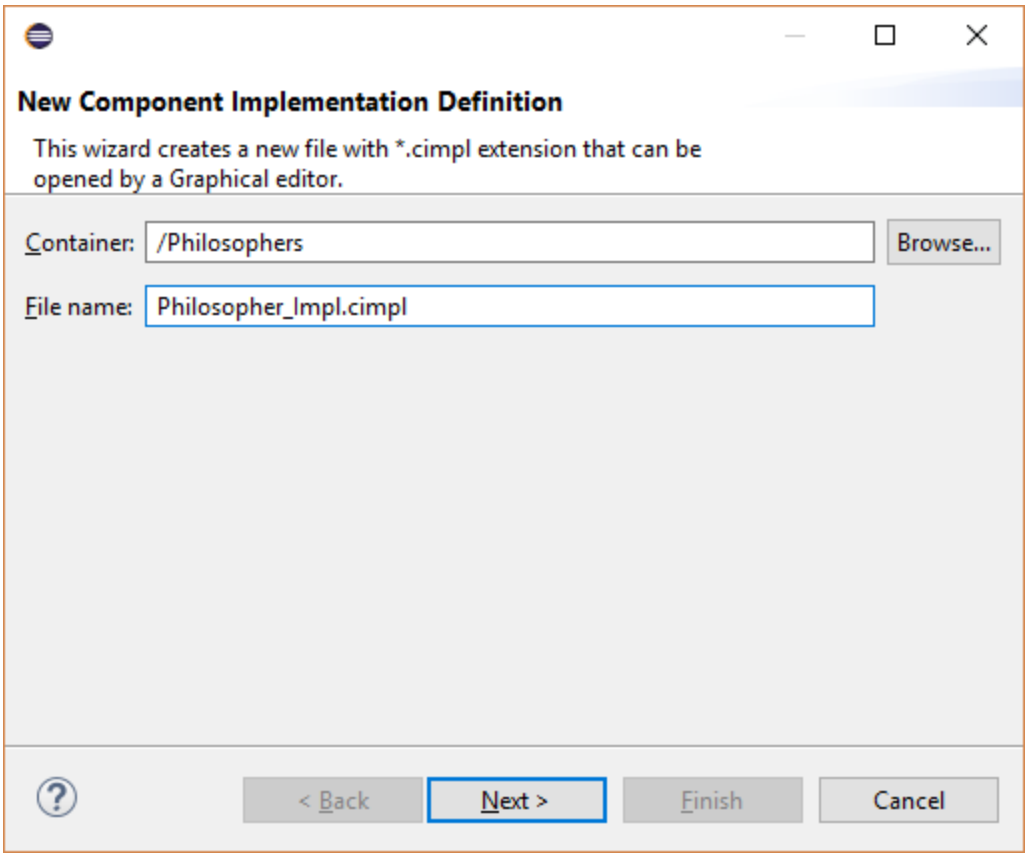
Select Component Implementation Editor from Tool bar which opens the below wizard.



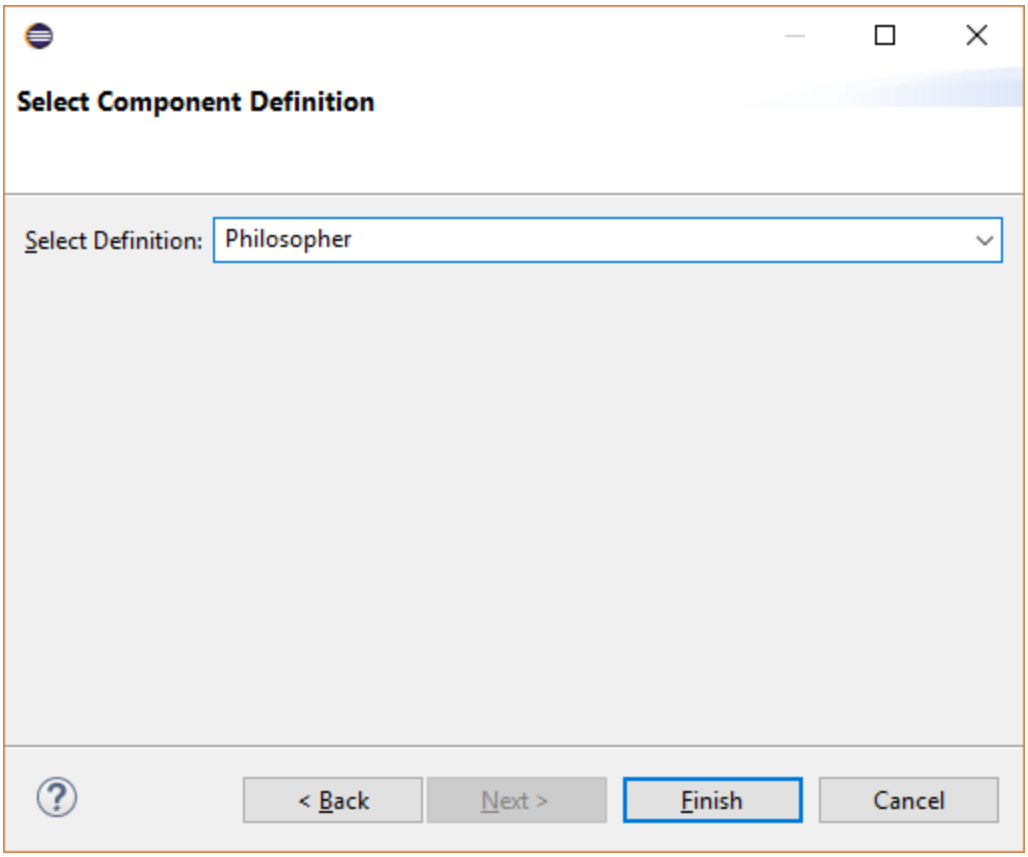
Select the container name as the top-level Project folder Philosophers



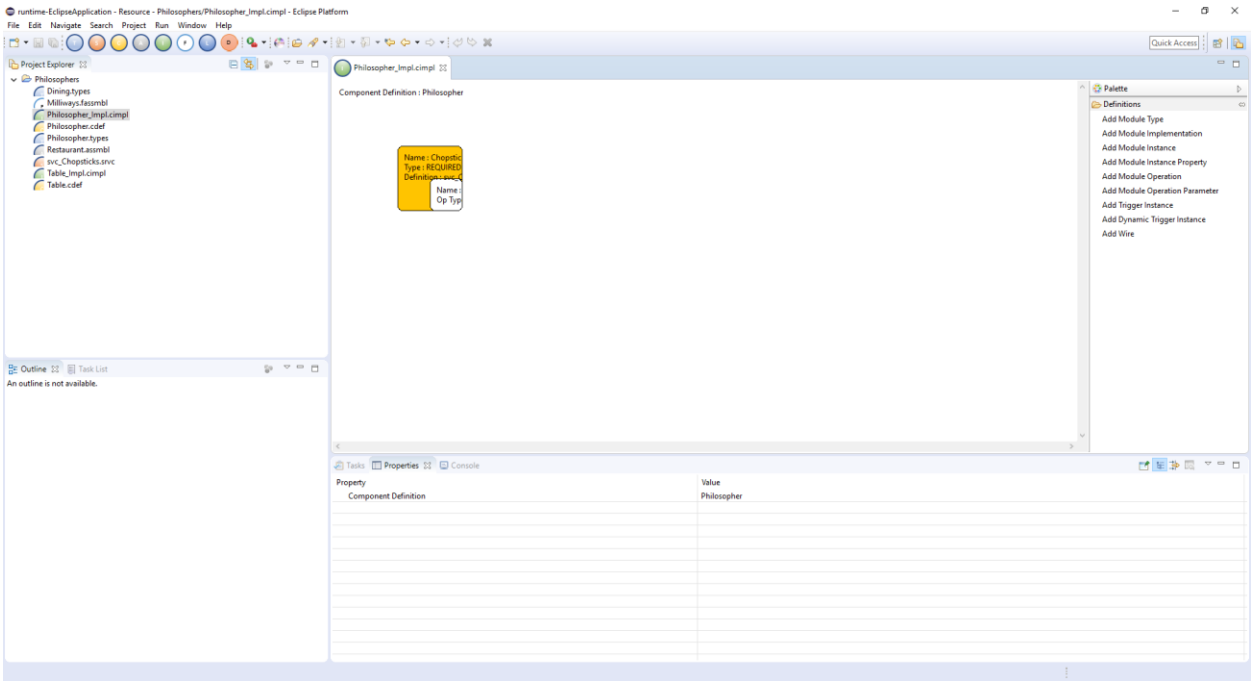
Change the file name to Philosopher_Impl.cimpl.



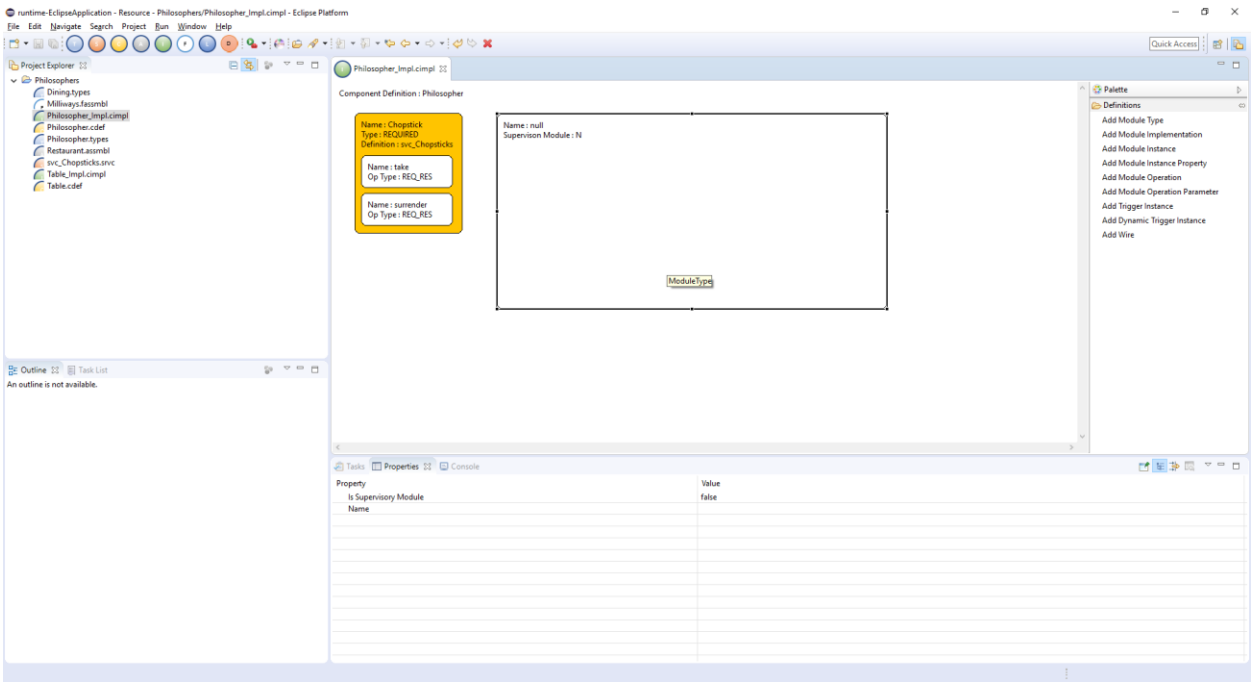
Select Next to pick the Philosopher component definition for the implementation and click Finish



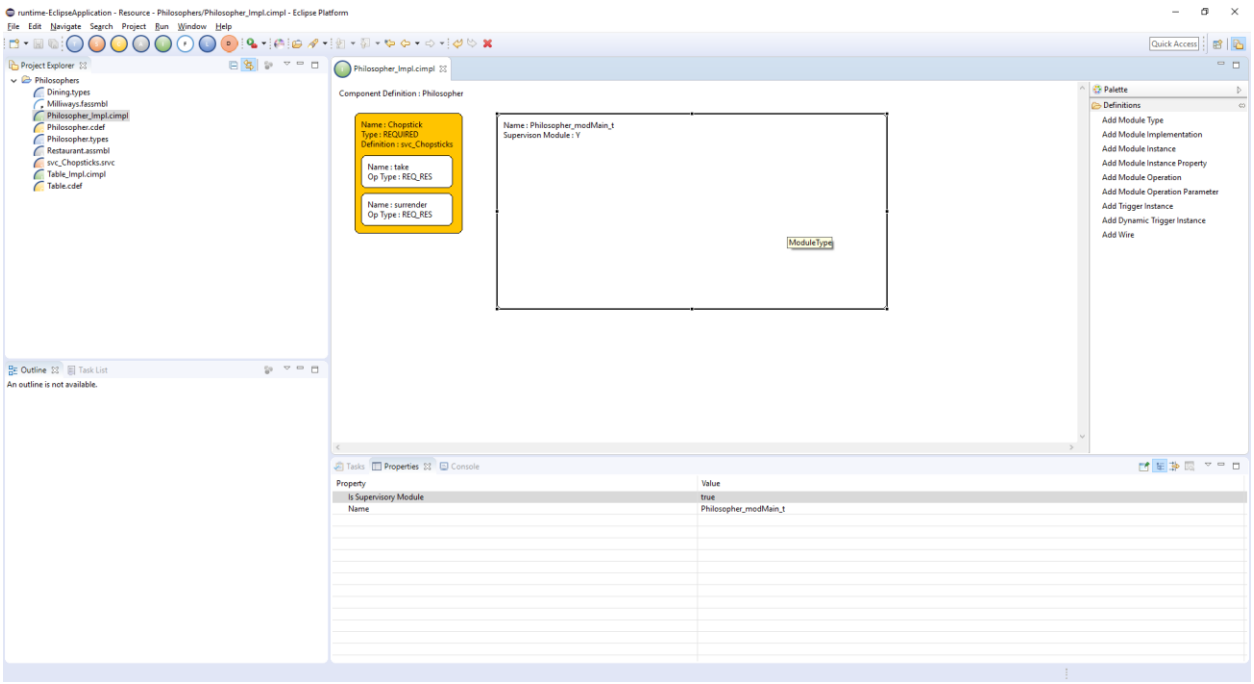
The Initial Screen is as below (Make sure that the Palette is drawn out for us to be able to add design elements):



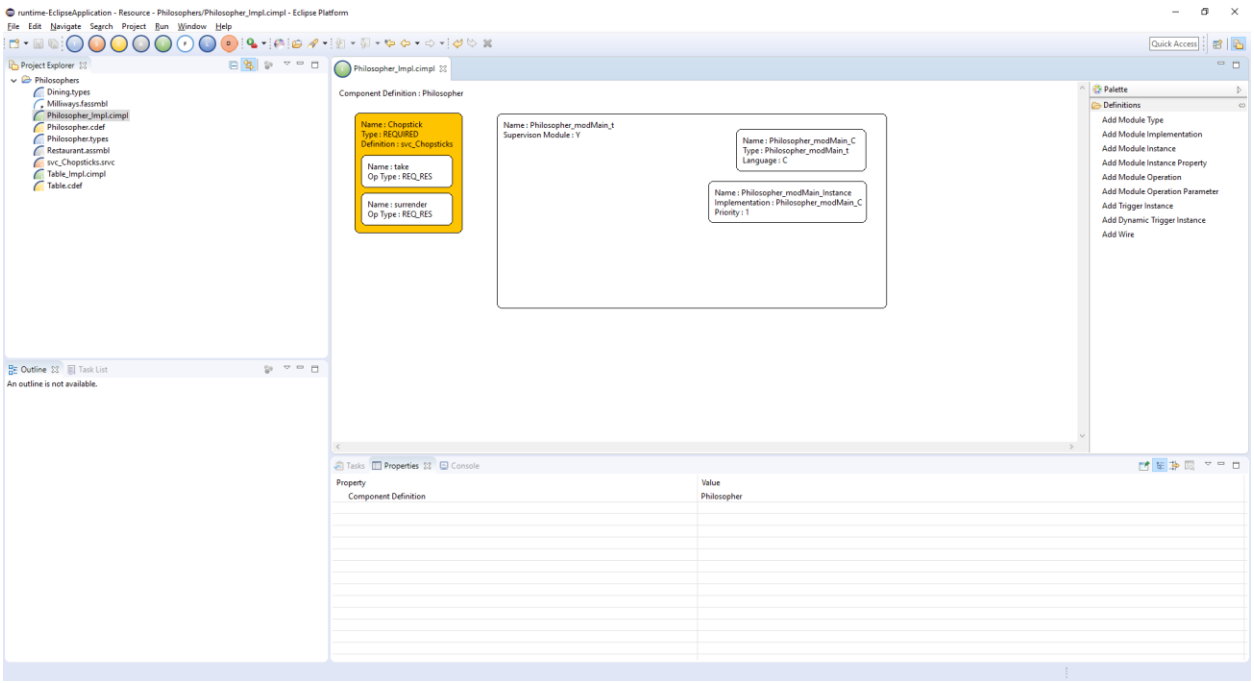
Rearrange the components for visibility. Add Module Type from Side Palette.



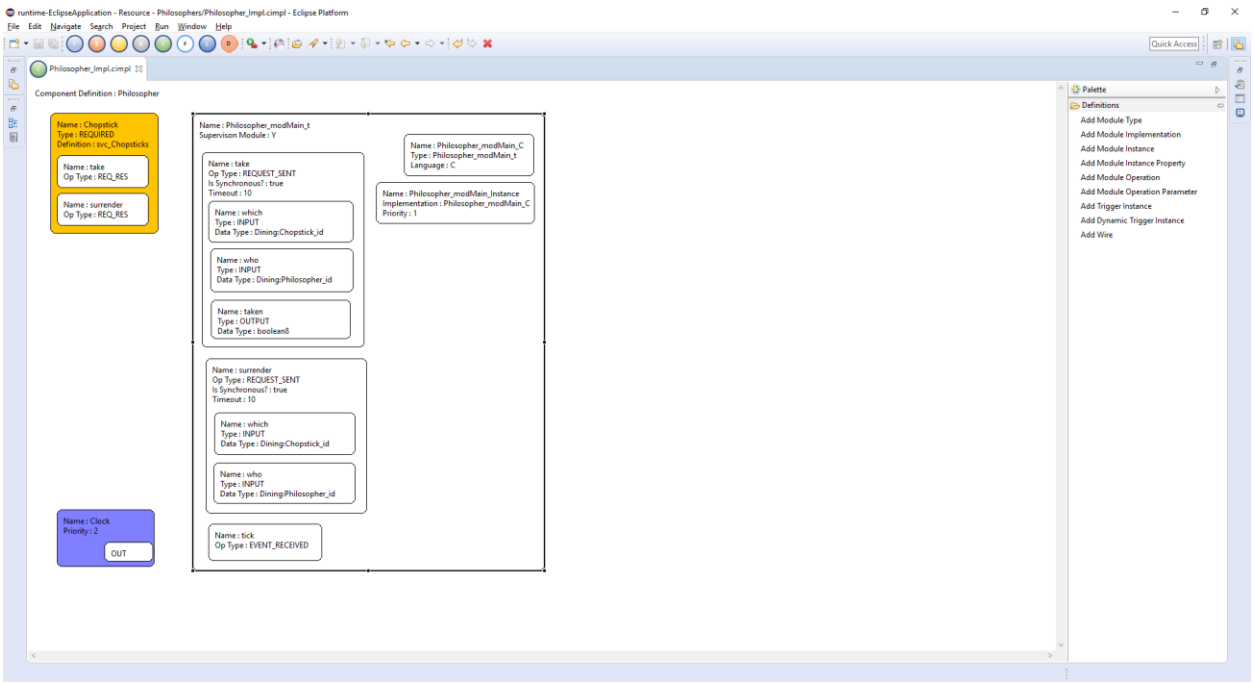
This adds an empty Module type. Using Properties modify the Name to Philosopher_modMain_t and Supervisory Module to true



Add the Module Implementation and Module Instance constructs from the palette and set the names to Philosopher_modMain_C and Philosopher_modMain_Instance. Enter other property details and select Philosopher_modMain_C as the Implementation on the instance.

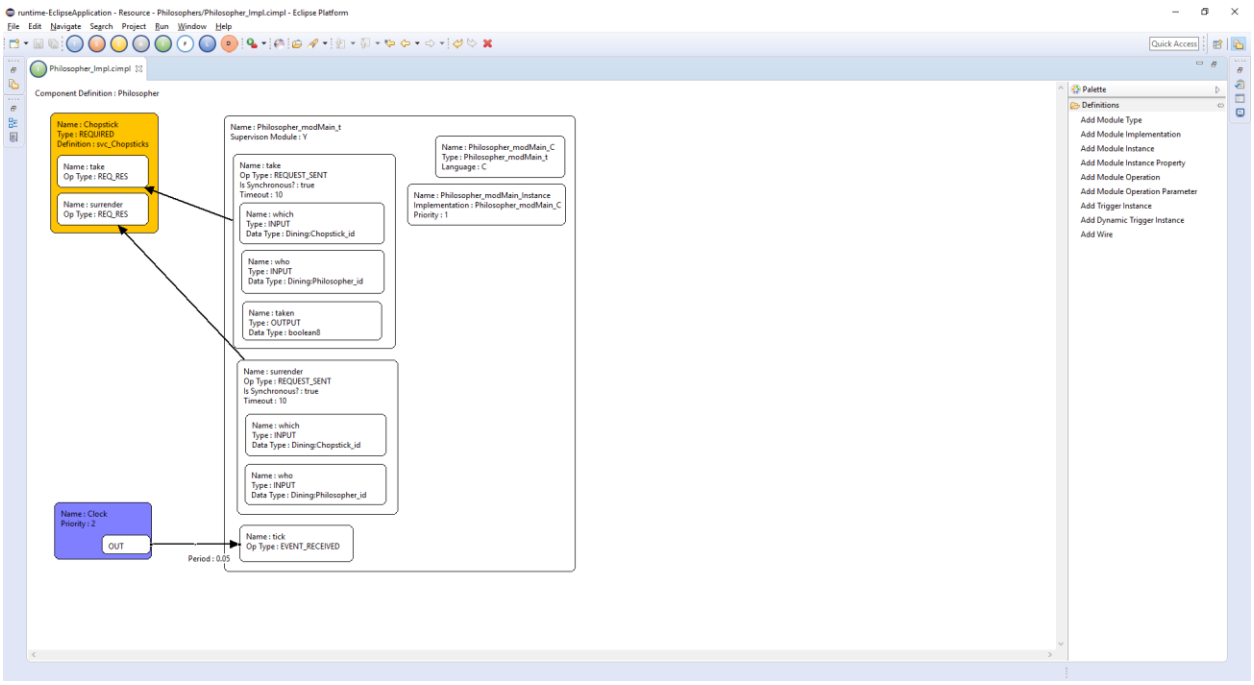


Add the Module Operations take (with Parameters which, who, and taken), surrender (with parameters (which and who), tick. Add Trigger Instace Clock.



Complete the wiring by selecting the wire from palette and set Module Instance to the instance name

- Philosopher_modMain_Instance/take – Chopsticks/take
- Philosopher_modMain_Instance/surrender – Chopsticks/surrender
- Clock/OUT - Philosopher_modMain_Instance/tick

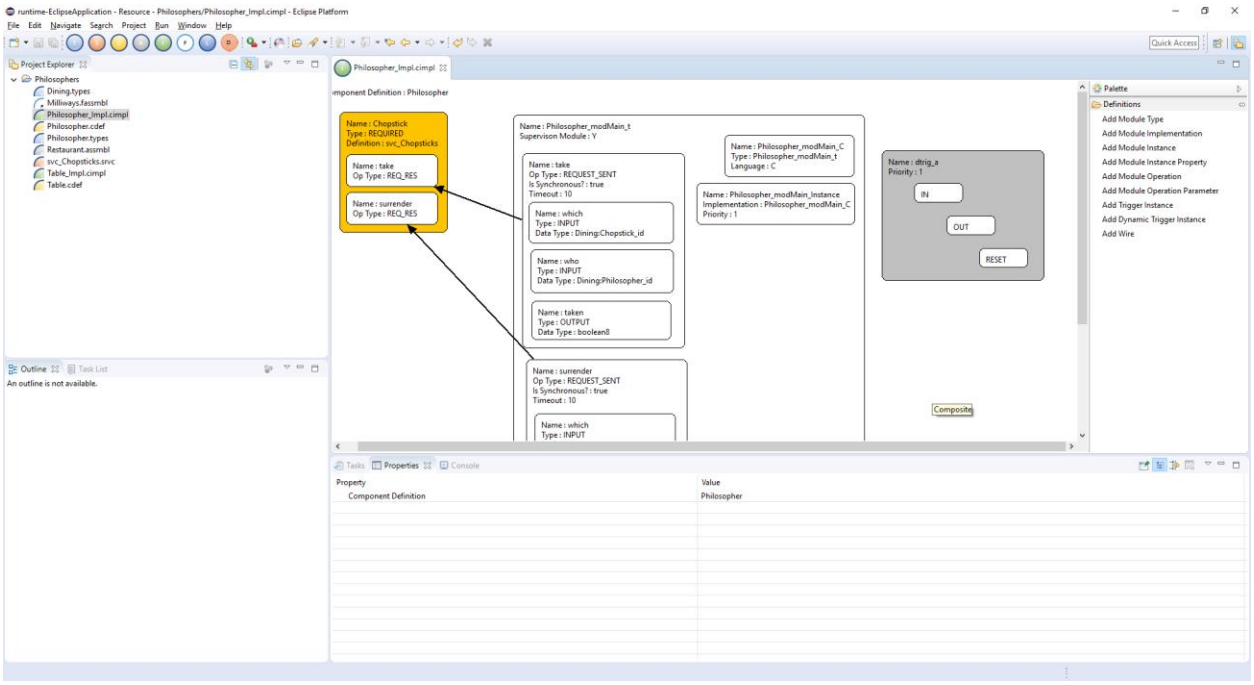


4.2.2 Other Definitions

Apart from the constructs mentioned in Dining Philosophers problem, we have the below:

4.2.2.1 Dynamic Trigger Instance

Select the Dynamic Trigger Instance from the Palette and add it onto the Canvas.



The Dynamic Trigger instance has 3 terminals that will be used to wire to other constructs.

4.3 Final Assembly Editor

ECOA terms Final Assembly as a construct to assign Implementation to Component Instances wired during Initial Assembly. A final assembly is derived from an initial assembly.

4.3.1 Scenario

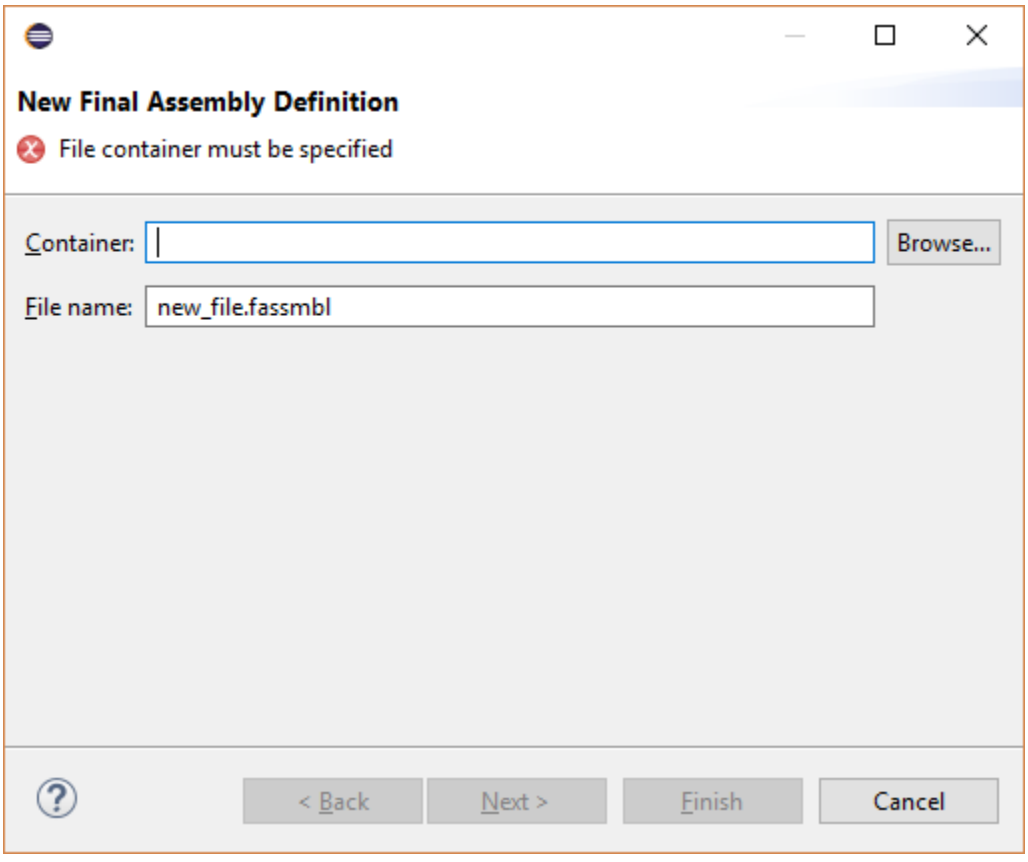
The Dining Philosophers problem defines Milliways as an assembly for Restaurant initial assembly.

4.3.1.1 Milliways Assembly

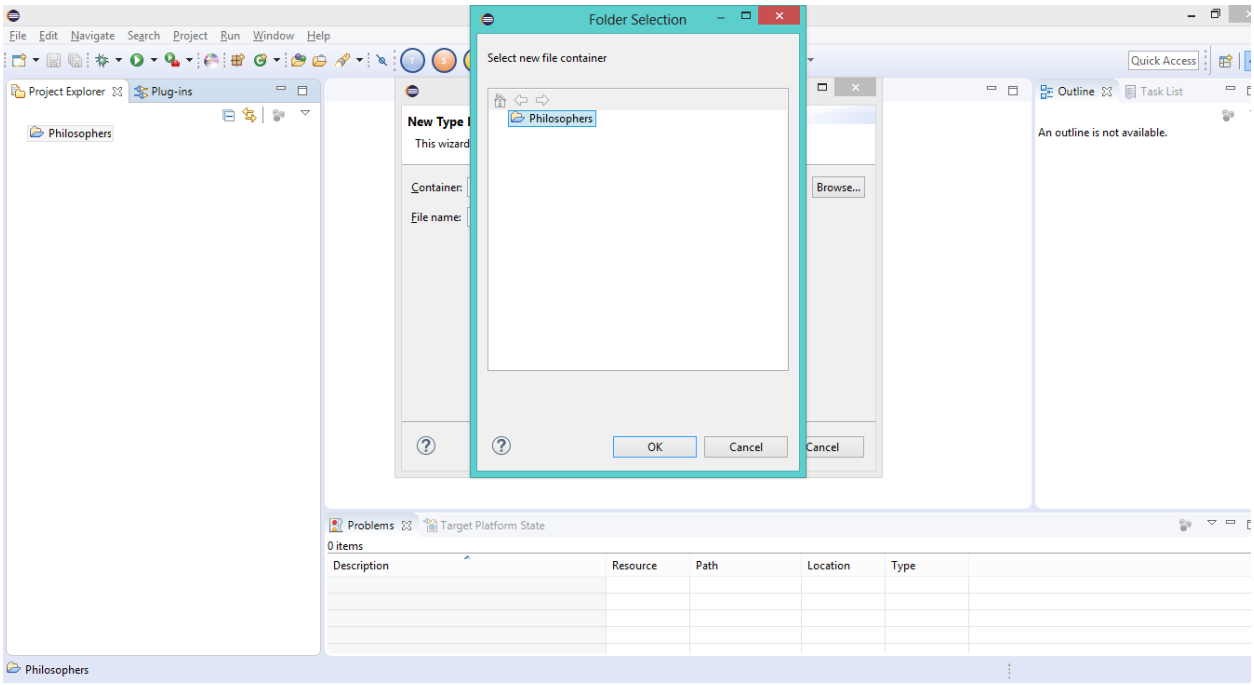
The assembly assigns Philosopher_Impl to the Philosophers and Table_Impl to Table definitions on the initial assembly.

4.3.1.1.1 Process

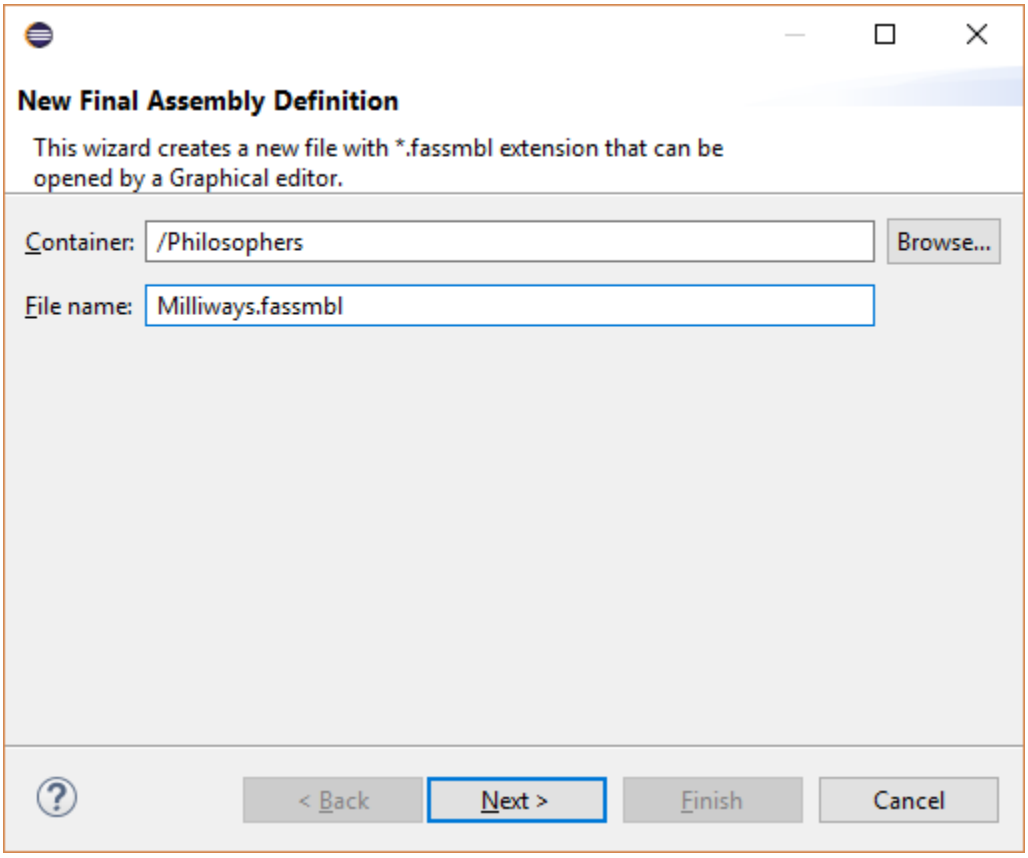
Select Final Assembly Editor from Tool bar which opens the below wizard.



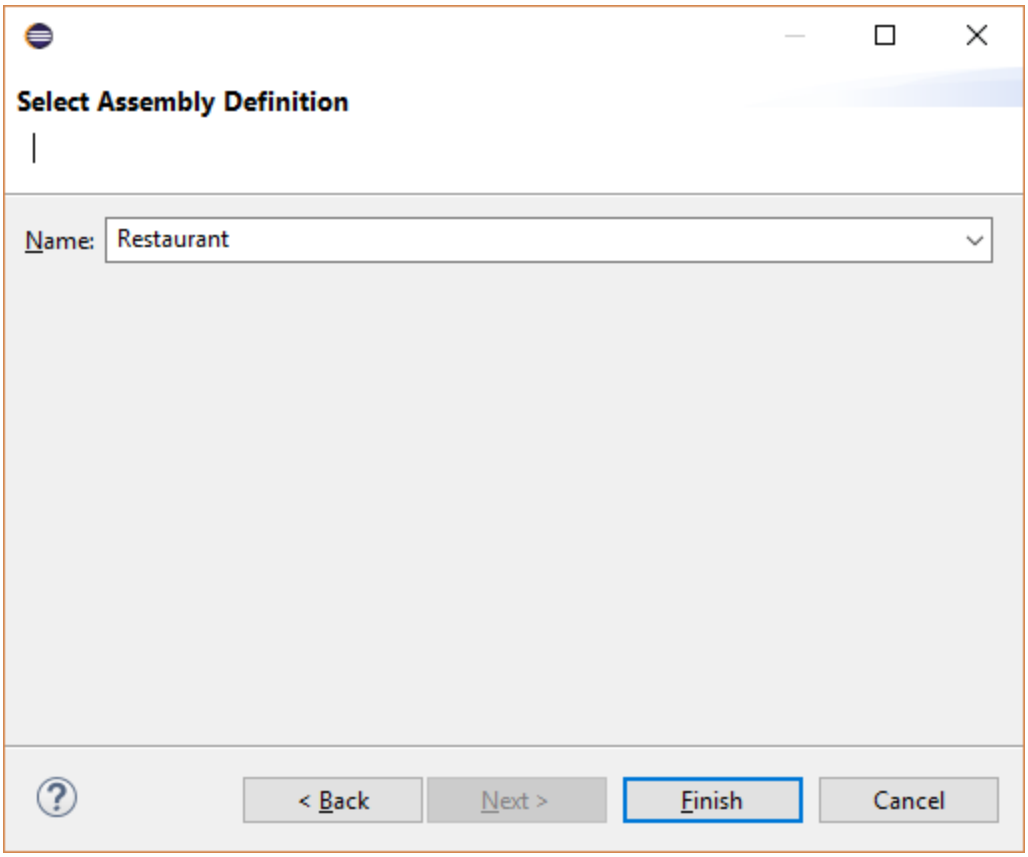
Select the container name as the top-level Project folder Philosophers



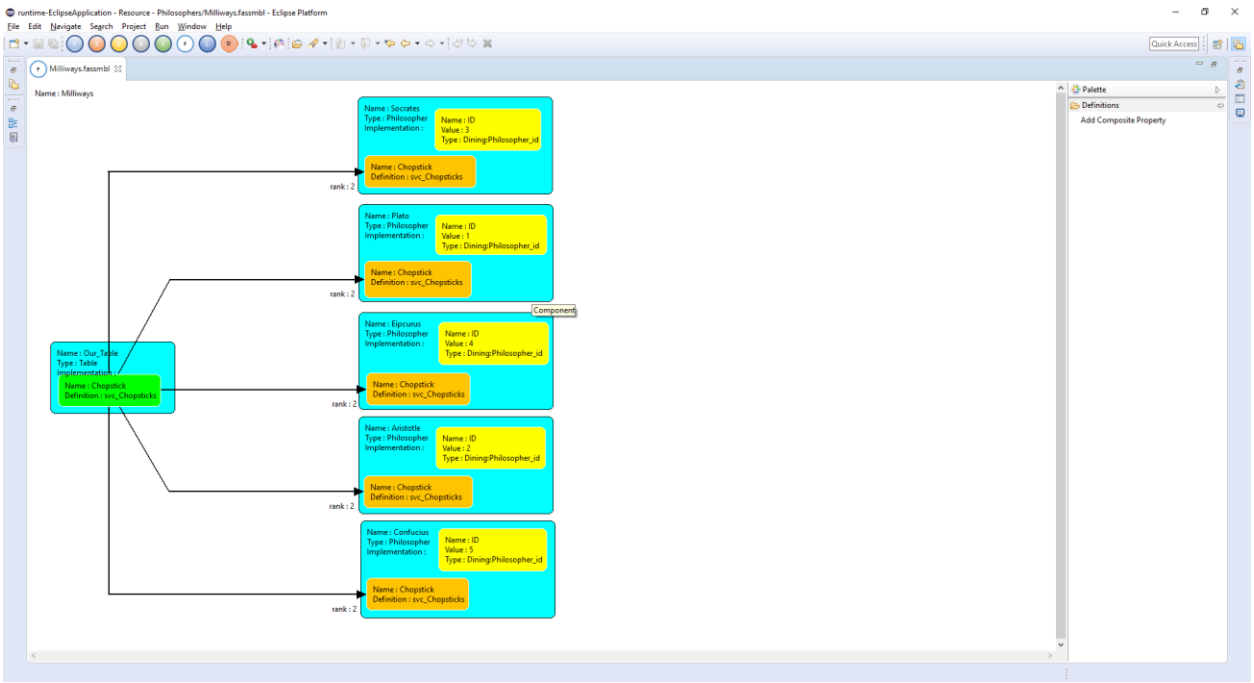
Change the file name to Milliways.assmbl.



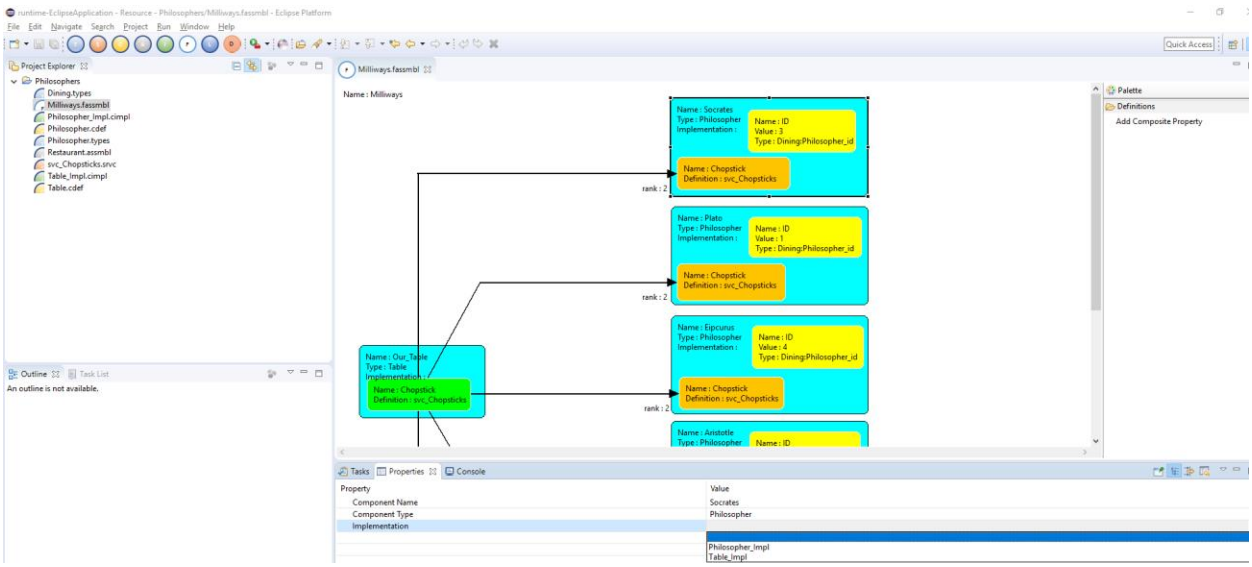
On the next page, select the Initial Assembly definition and click Finish.



The Initial Screen imported from Initial Assembly is as below:



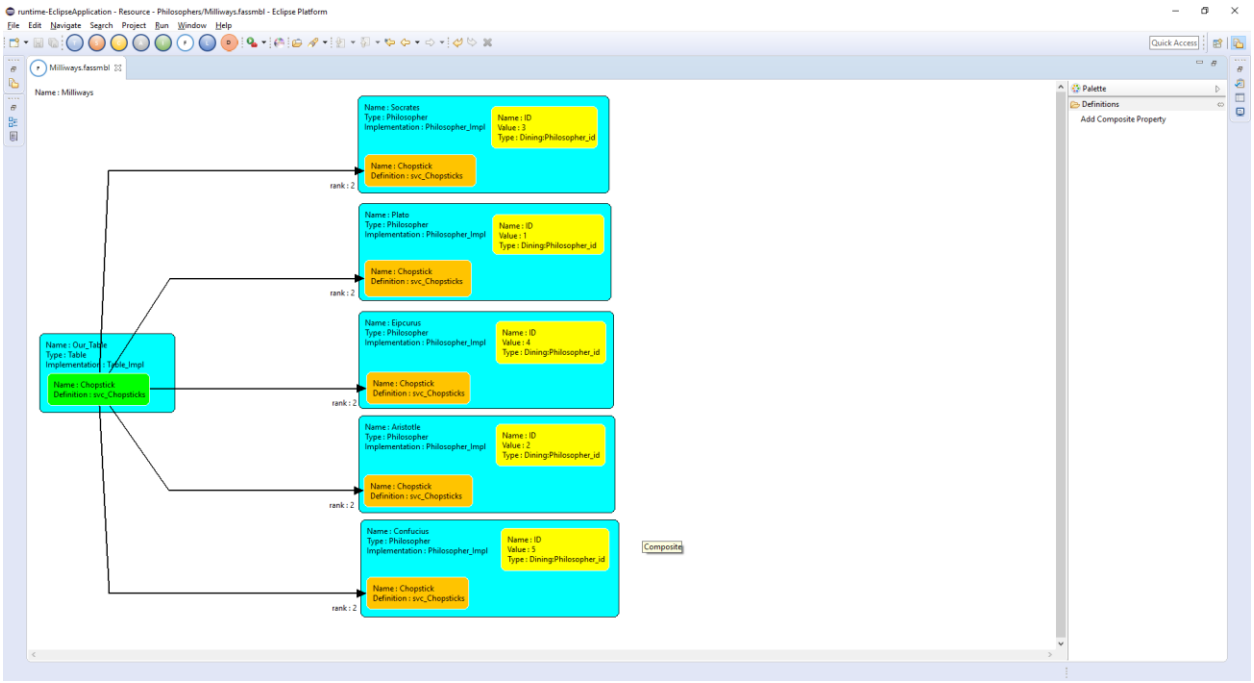
Select each of the blocks to open property editors (Double click if not opened by default):



Select the Implementation for corresponding components.

- Plato – Philosopher_Impl
- Aristotle – Philosopher_Impl
- Socrates – Philosopher_Impl
- Epicurus – Philosopher_Impl
- Confucius – Philosopher_Impl
- Our_Table – Table_Impl

The below is the screen after modifications:



4.4 Logical Systems Editor

ECOA Terms Logical System as a construct to define the Platform Configuration. The basic constructs of a Logical Systems Editor are:

1) Logical Computing Platform
2) Logical Computing Node
3) Logical Processor

4.4.1 Scenario

The Dining Philosophers problem define Logical Systems as below:

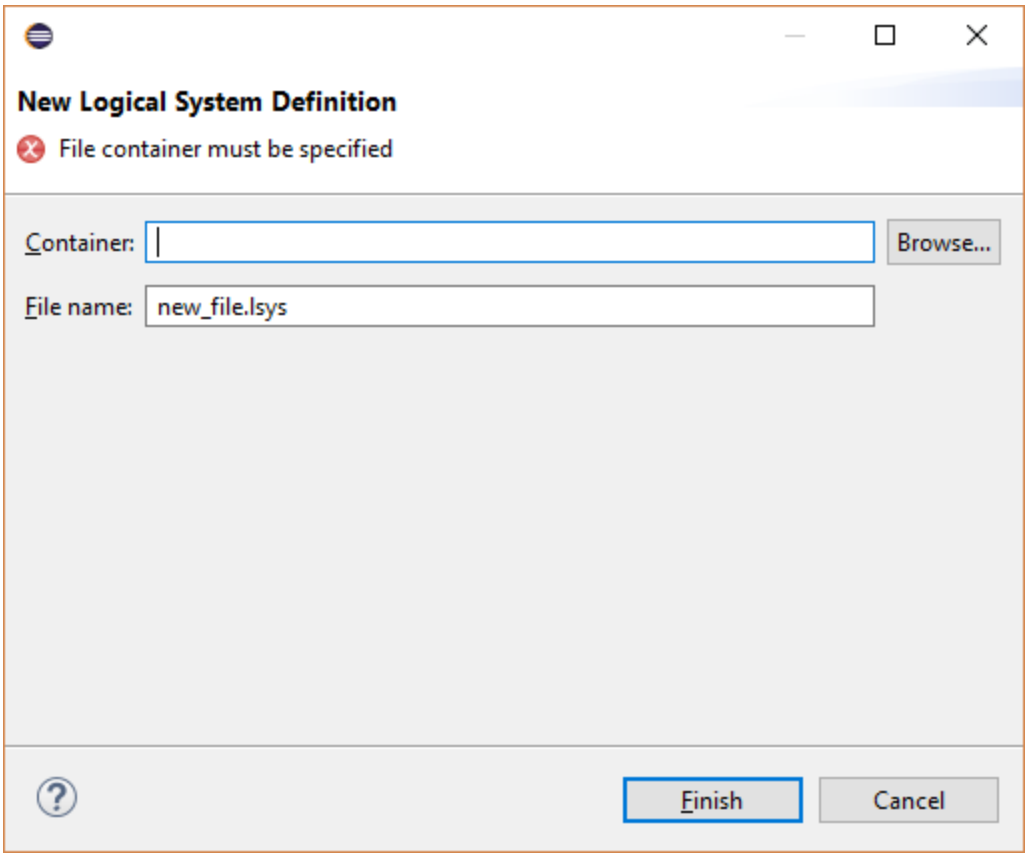
- Name: Desktop
- Computing Node CPU
    - endianness LITTLE
    - os linux
    - availableMemory 1
    - moduleSwitchTime 10
- Processor
    - number 1
    - type X86_64
    - stepDuratiuon 1020

4.4.1.1 Desktop Logical System

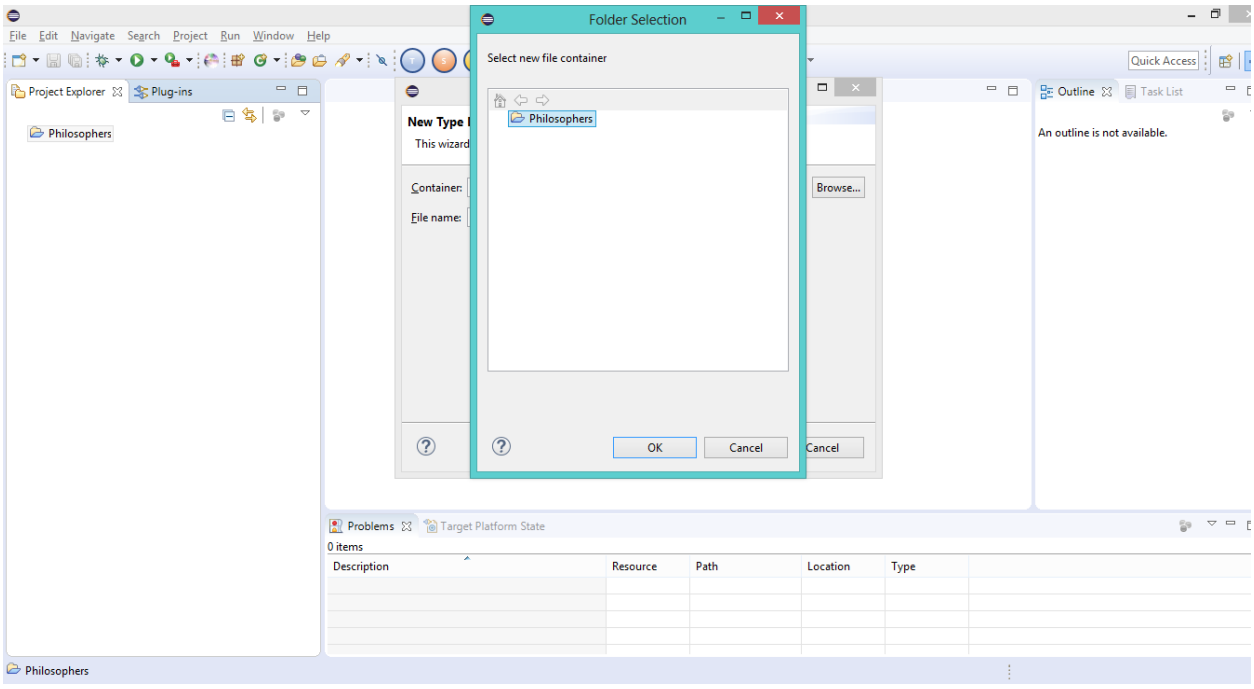
The Desktop Logical system is the container for CPU computation node and processor.

4.4.1.1.1 Process

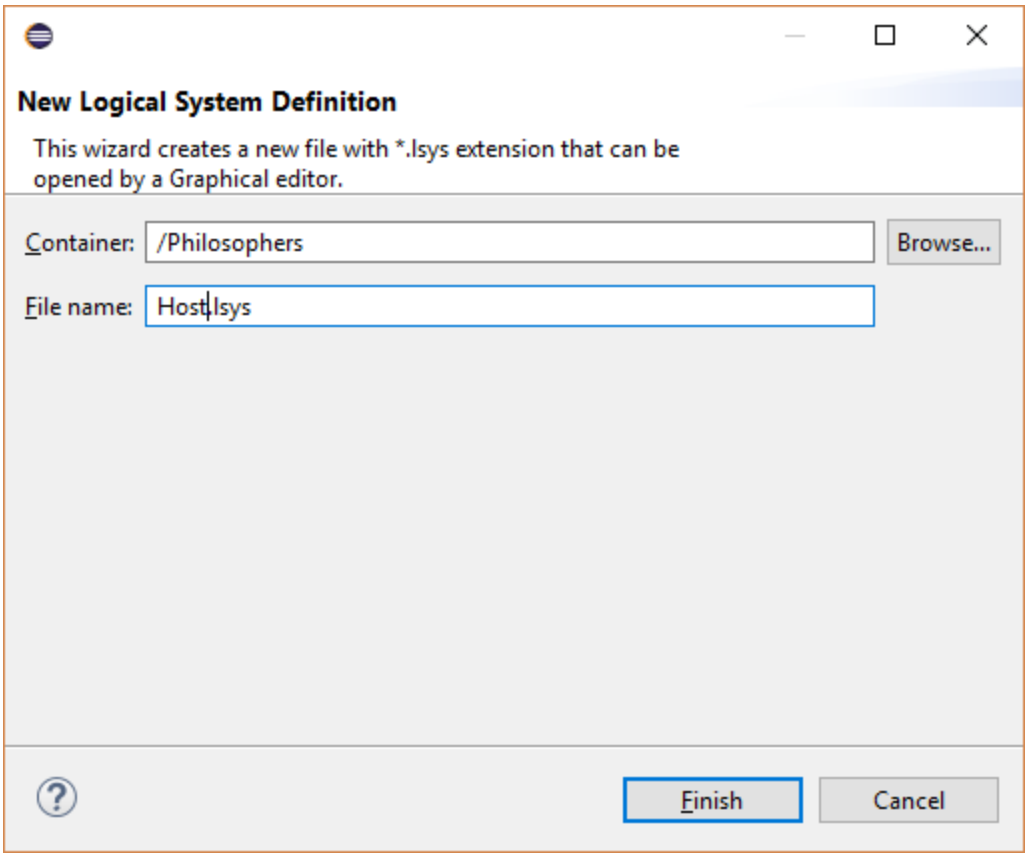
Select Logical Systems Editor from Tool bar which opens the below wizard.



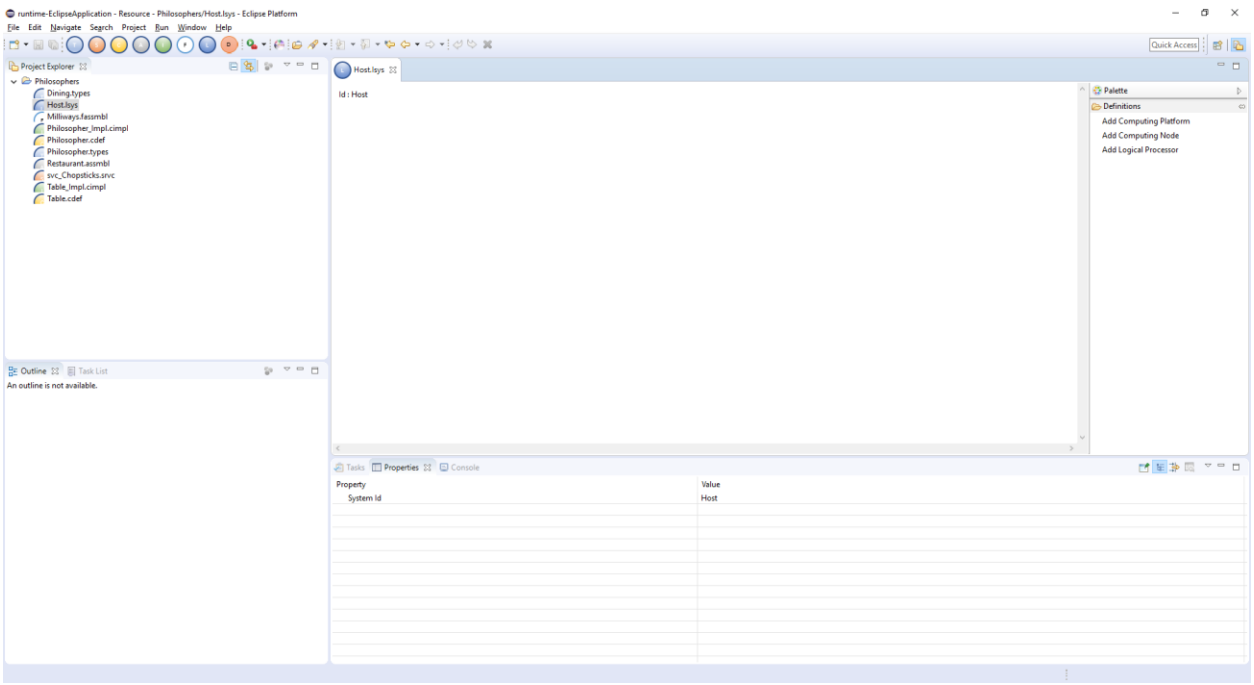
Select the container name as the top-level Project folder Philosophers



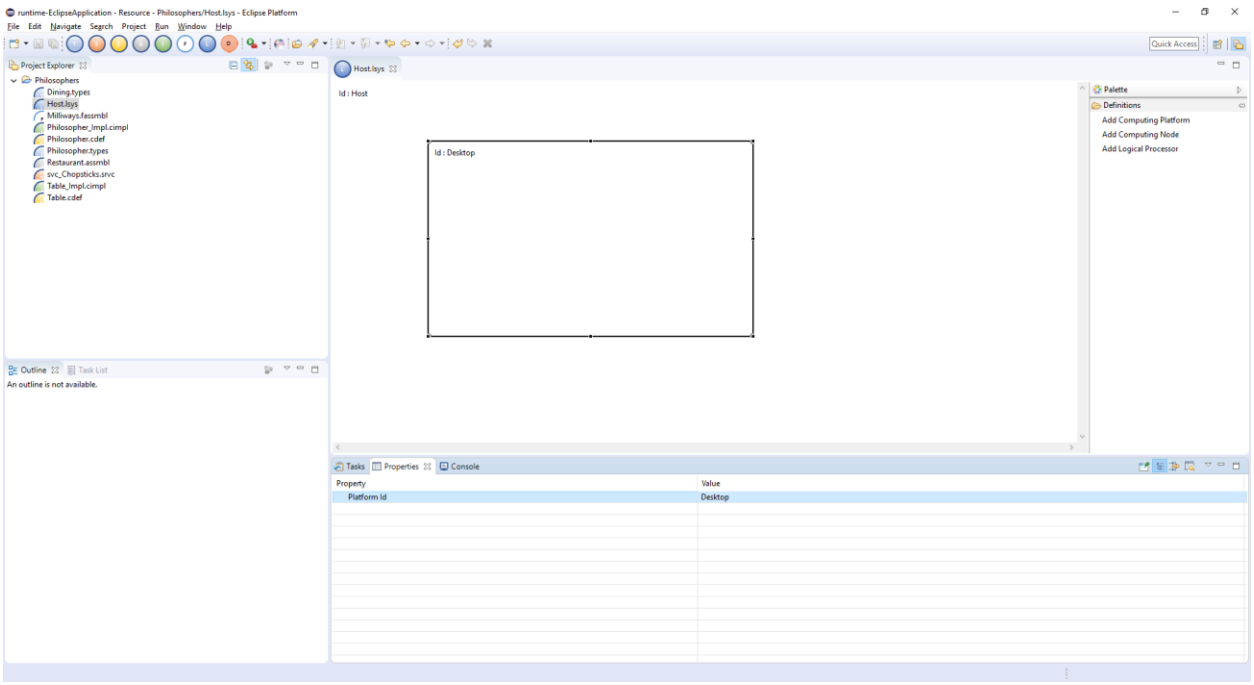
Change the file name to Host.lsys.



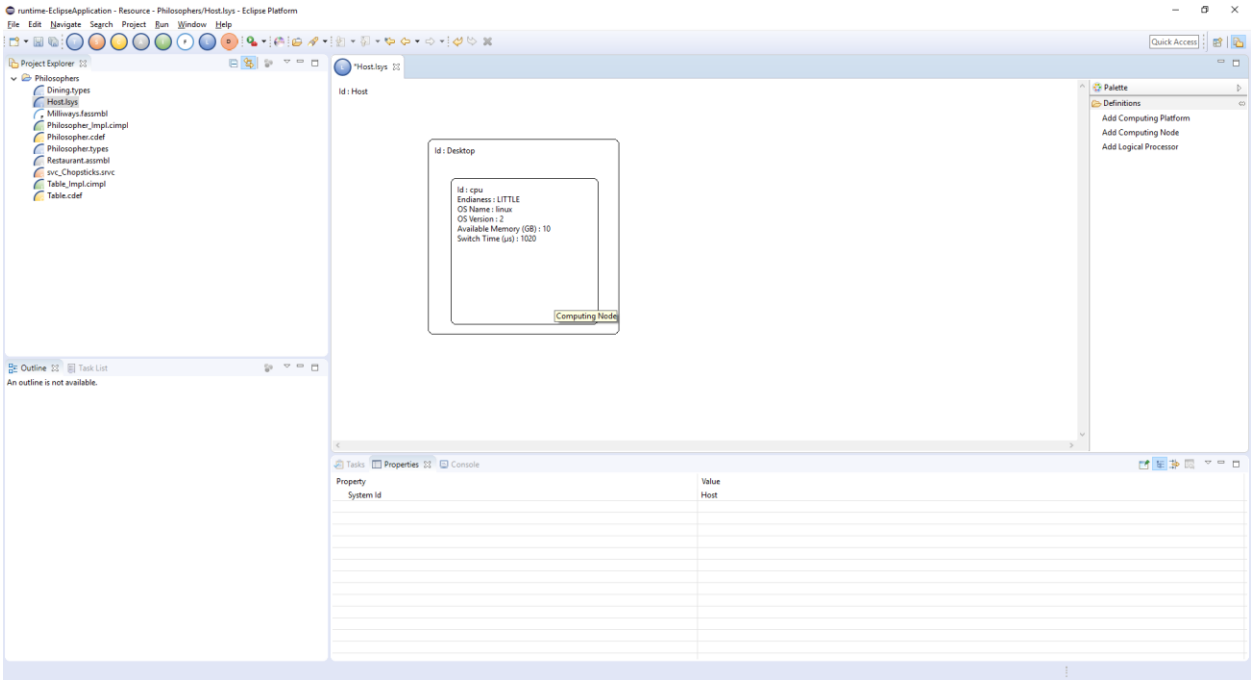
The Initial Screen is as below (Make sure that the Palette is drawn out for us to be able to add design elements):



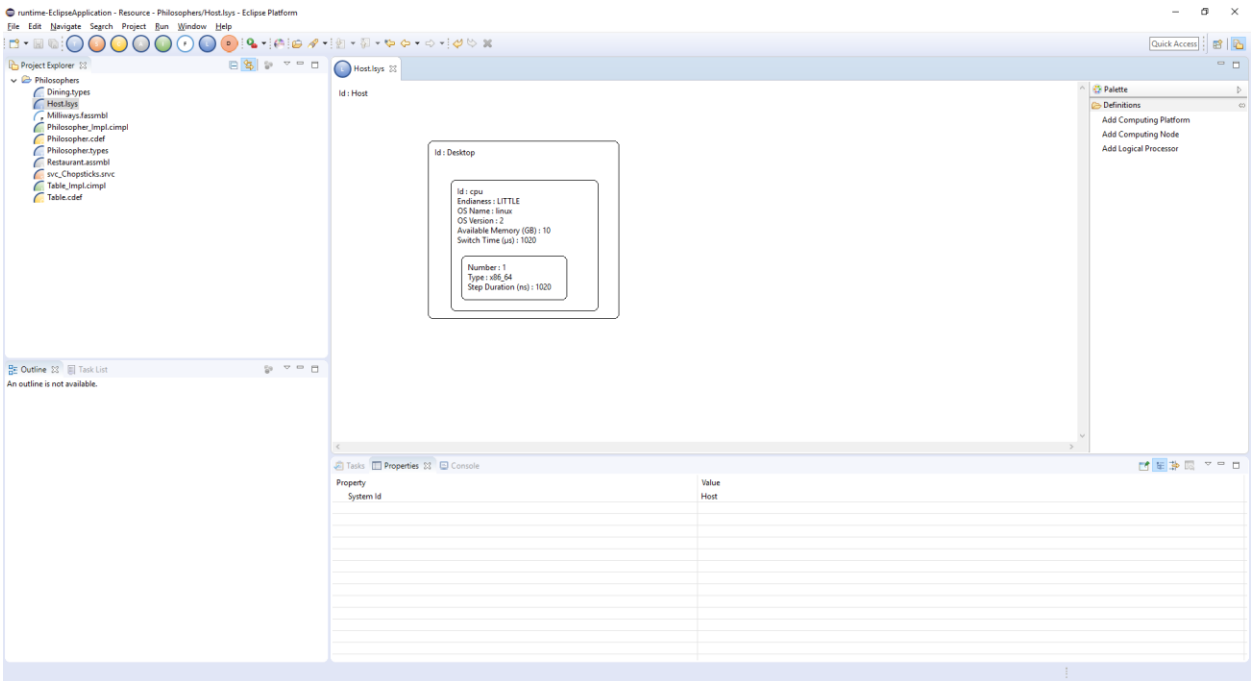
Add the Logical Computing Platform onto the Canvas from the palette and set Id to Desktop



Add Computing Node and enter the details and extend the diagram to accommodate Processor



Add Processor and enter details.



4.5 Deployment Editor

ECOA terms Deployment editor as the construct to assign the Component Instances to Platform Configurations for deployment. The constructs in Deployment Editor are:

1) Protection Domain
2) Platform Configuration
3) Deployed Trigger Instance
4) Deployed Module Instance
5) Computation Node Configuration

4.5.1 Scenario

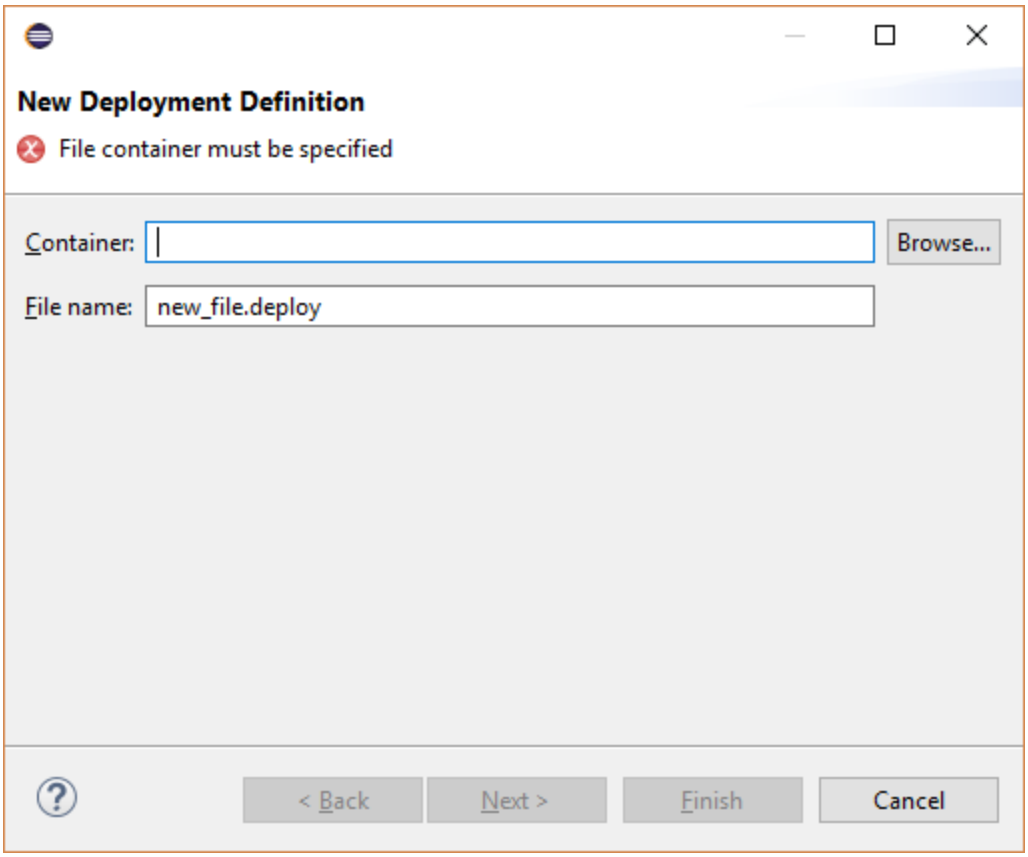
The Dining Philosophers problem defines deployment based on the Desktop Logical System and Milliways final assembly.

4.5.1.1 Host Deployment

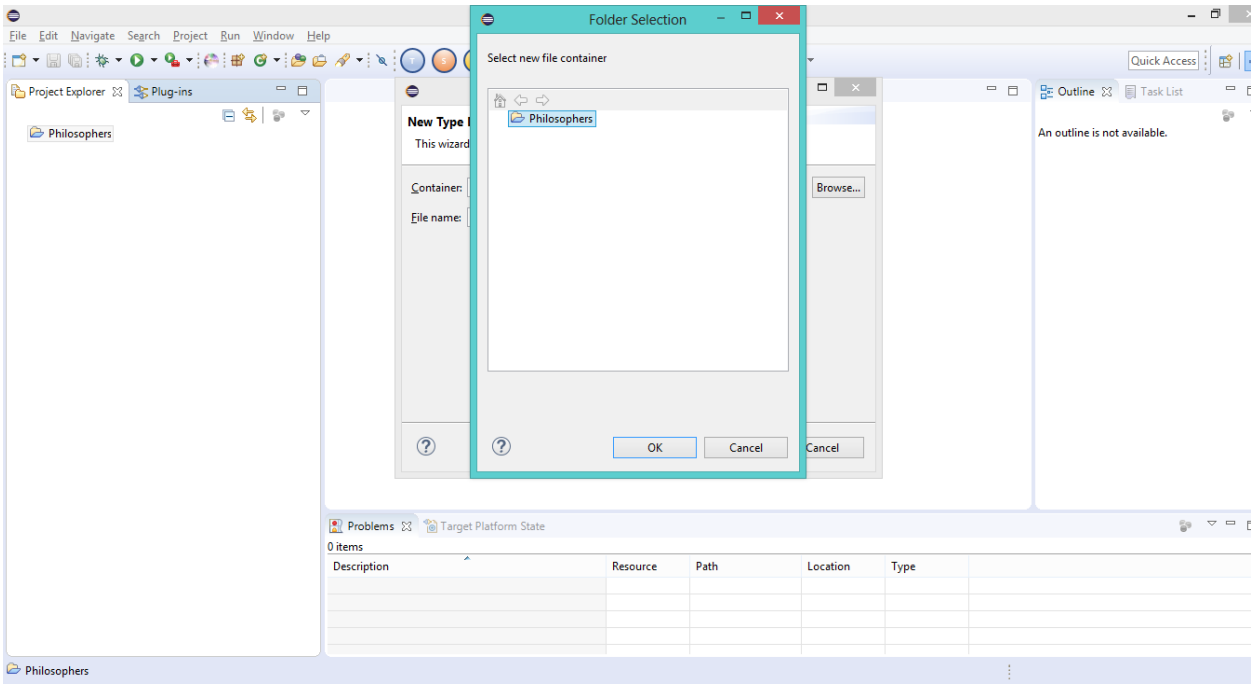
The Host deployment contains separate protection domains for each of the component implementation and a platform / compute node configuration.

4.5.1.1.1 Process

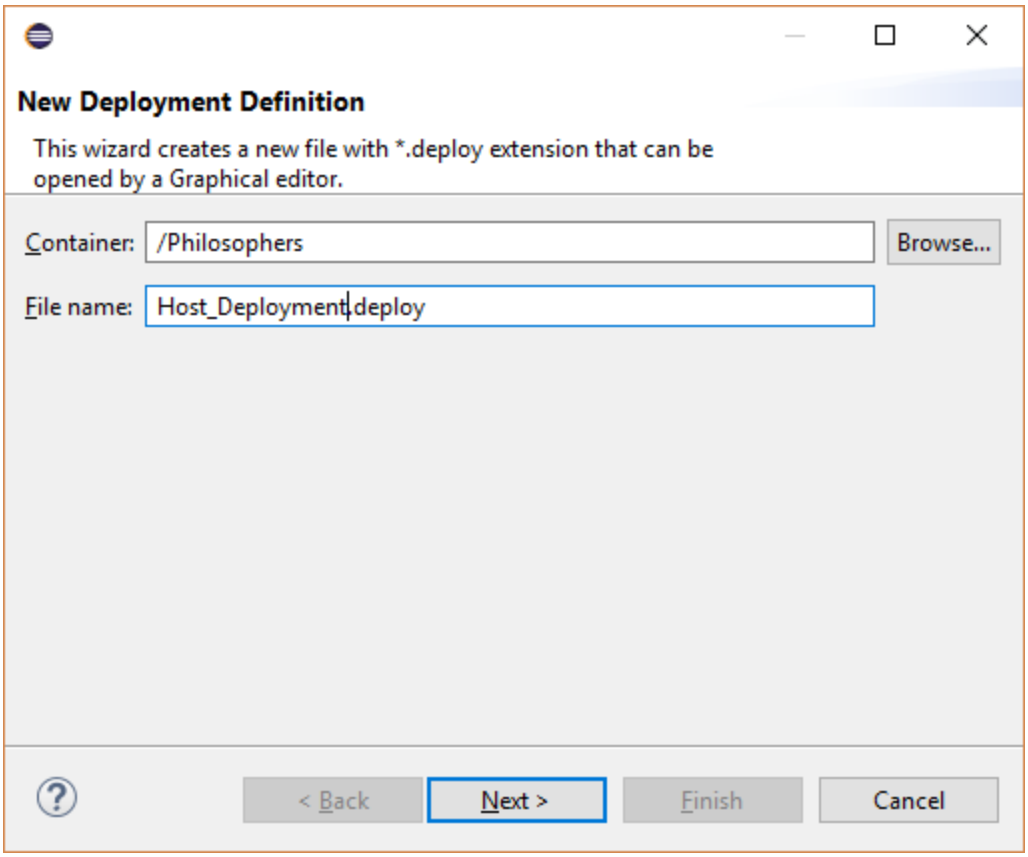
Select Deployment Editor from Tool bar which opens the below wizard.



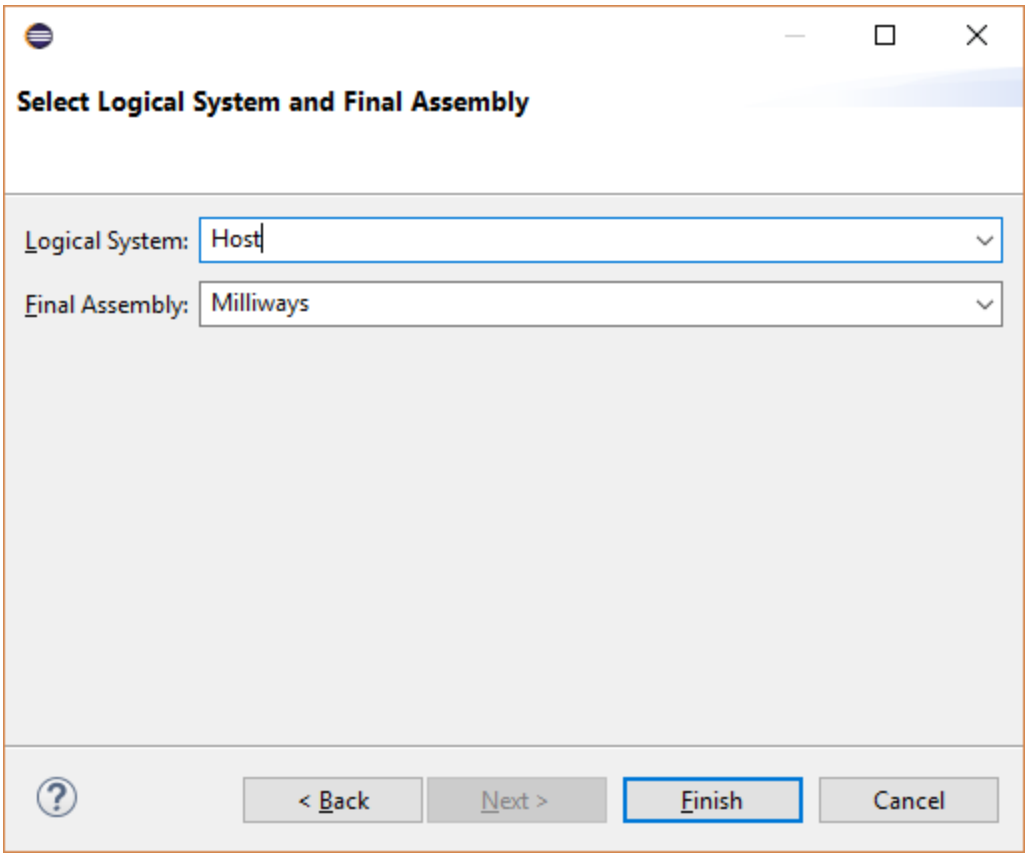
Select the container name as the top-level Project folder Philosophers



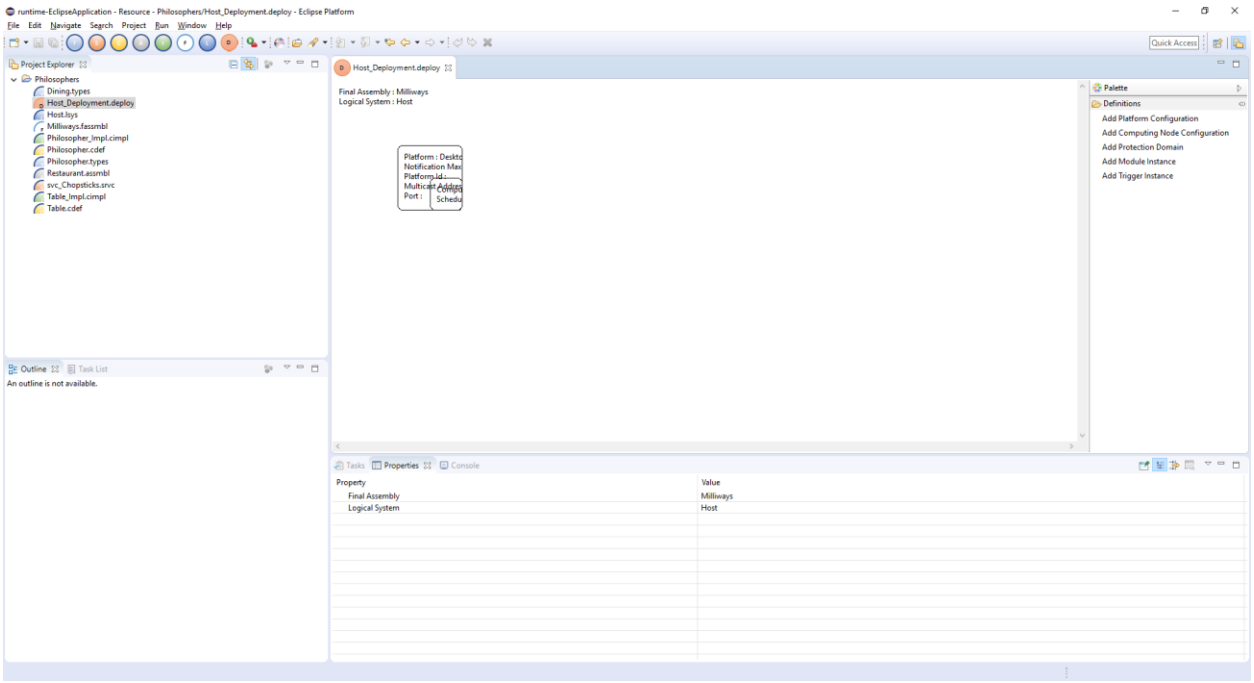
Change the file name to Host_Deployment.deploy.



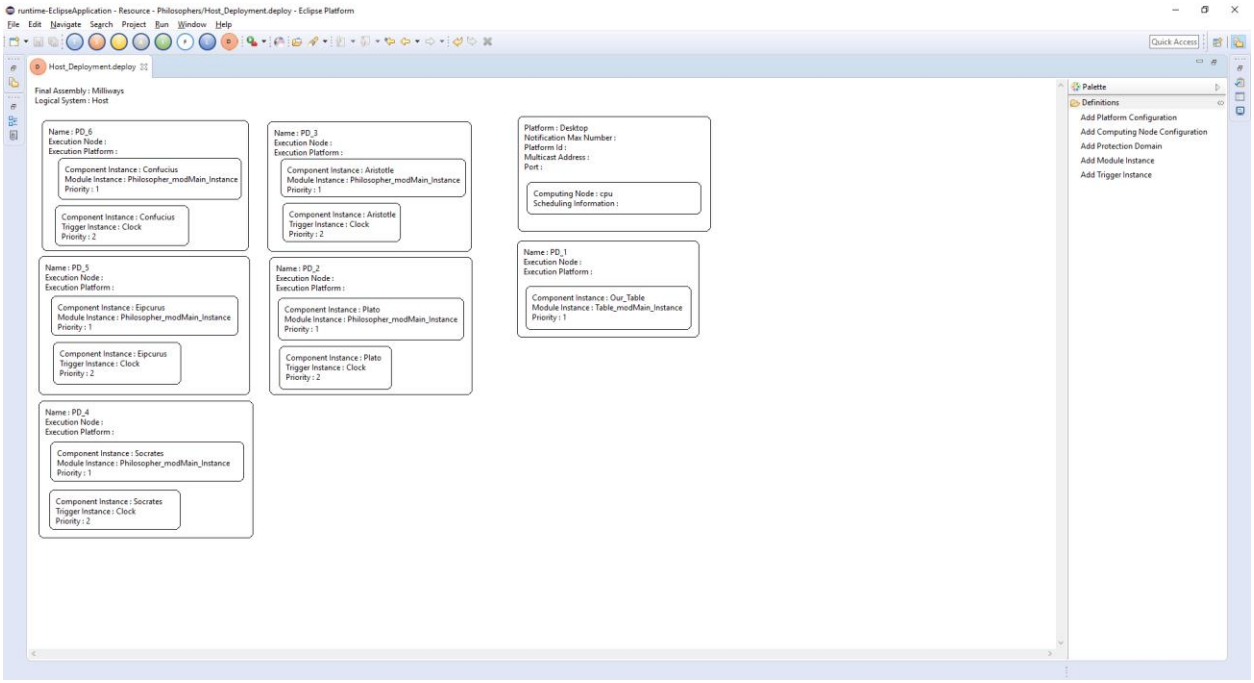
On the next page, select the Final Assembly and Logical Systems definition and click Finish.



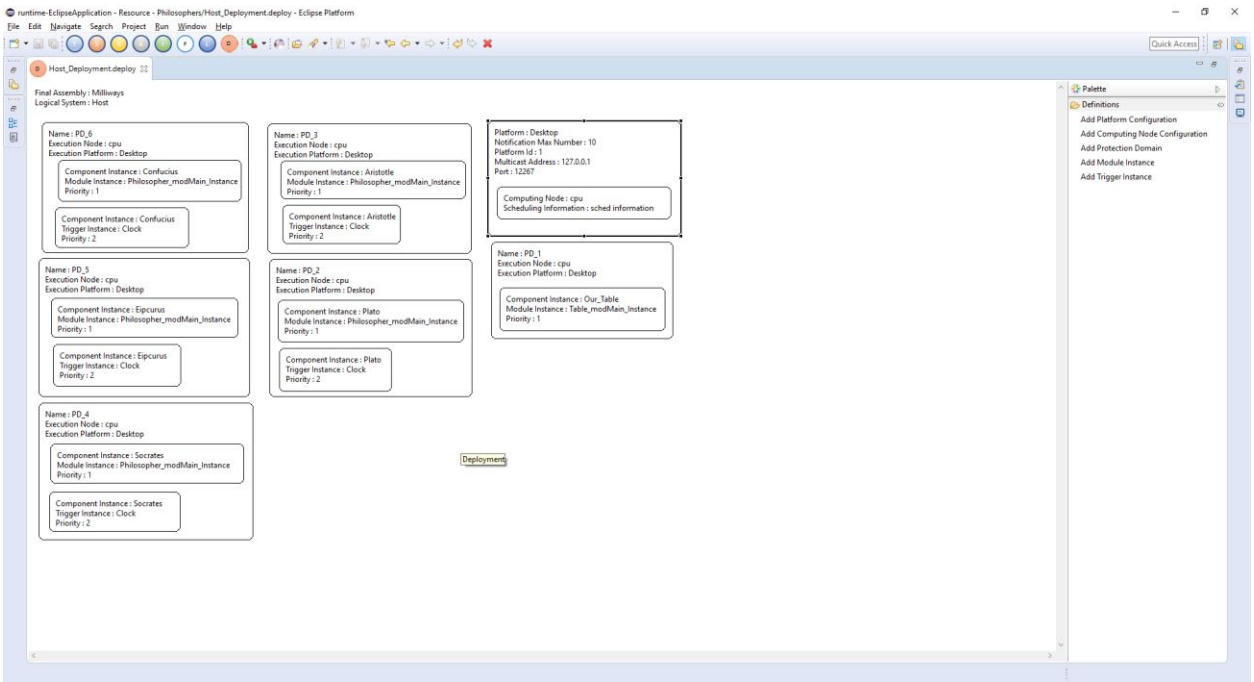
The Initial Screen is as below (Make sure that the Palette is drawn out for us to be able to add design elements):



Re-arrange the components so that they are visible.



Populate the missing information to make the information complete.

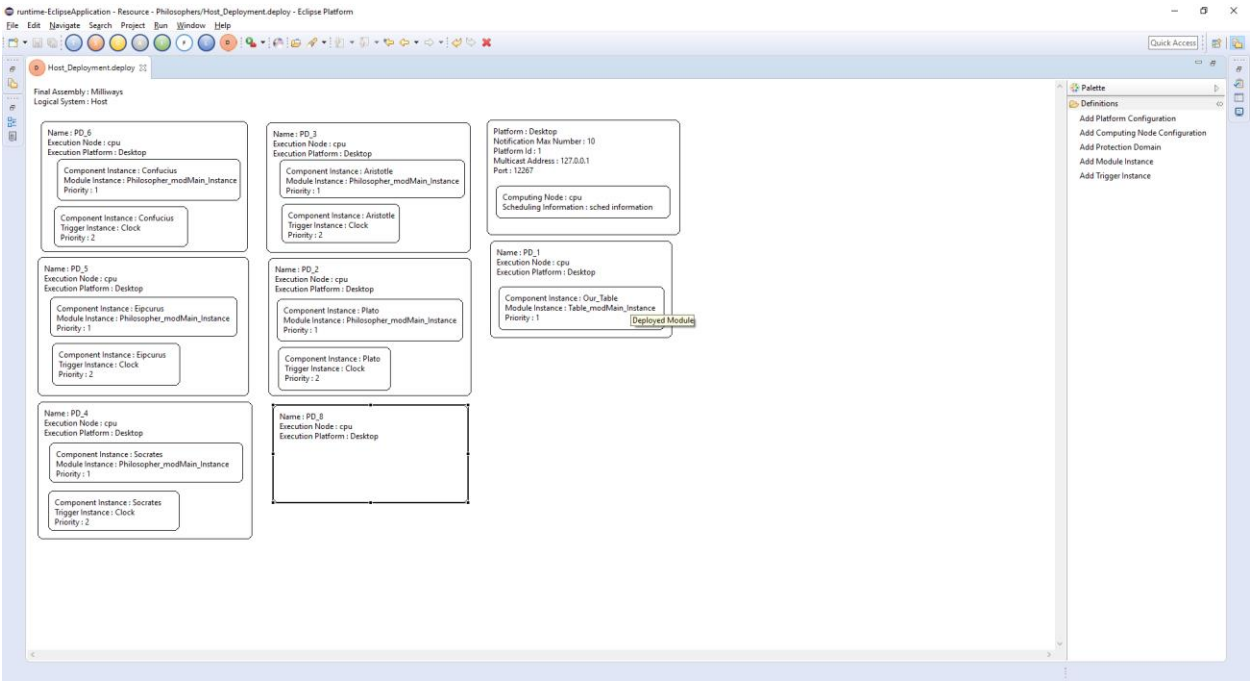


4.5.2 Other Definitions

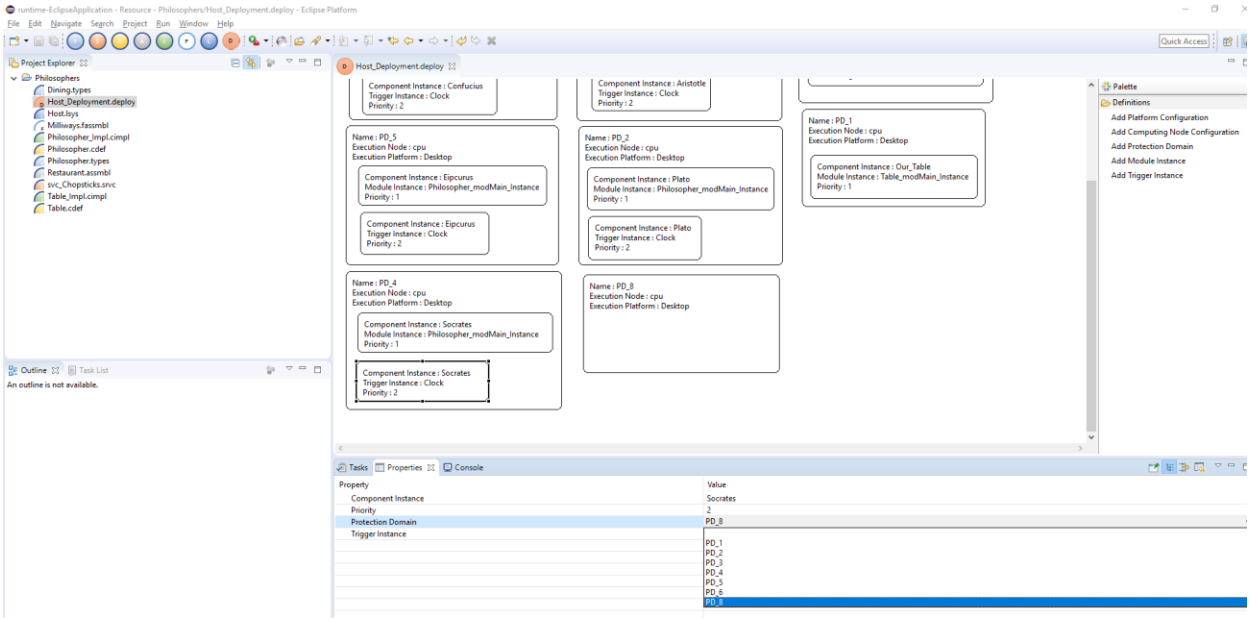
4.5.2.1 Moving Between Protection Domains

Once the deployment editor is populated with information from Final Assembly and Logical System, there might be a requirement to re-group these constructs.

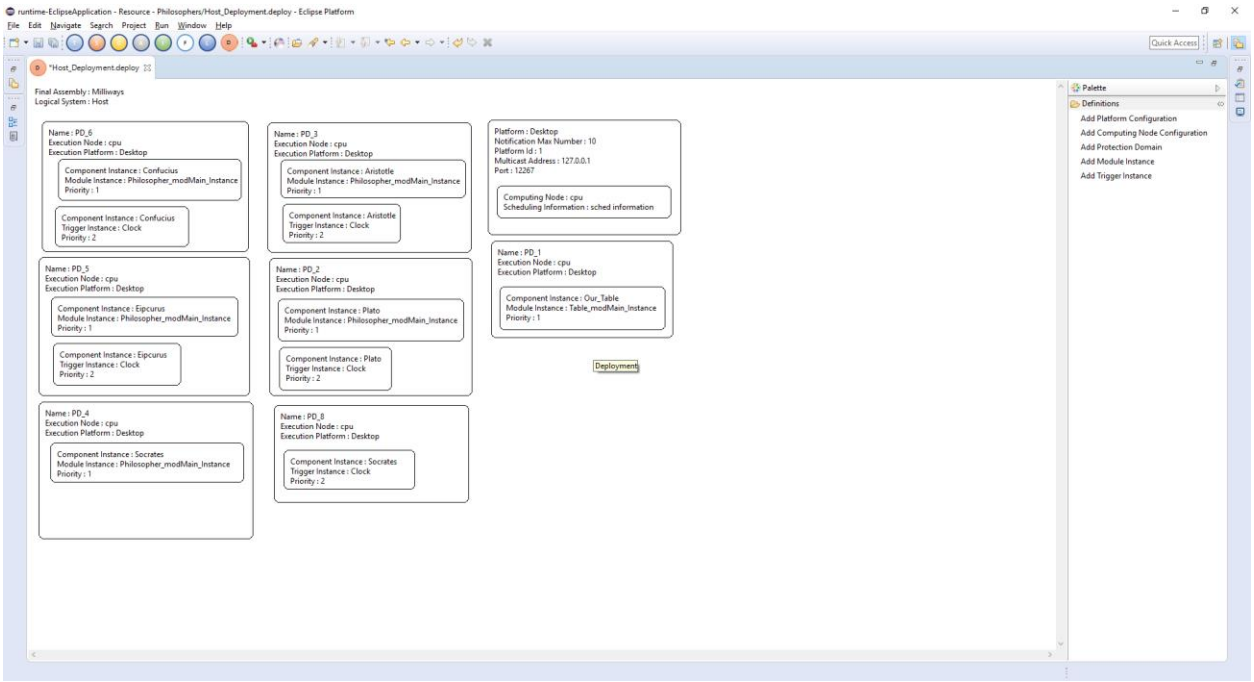
Select the New Protection Domain from the right-hand side palette and add to the canvas and fill in the details:



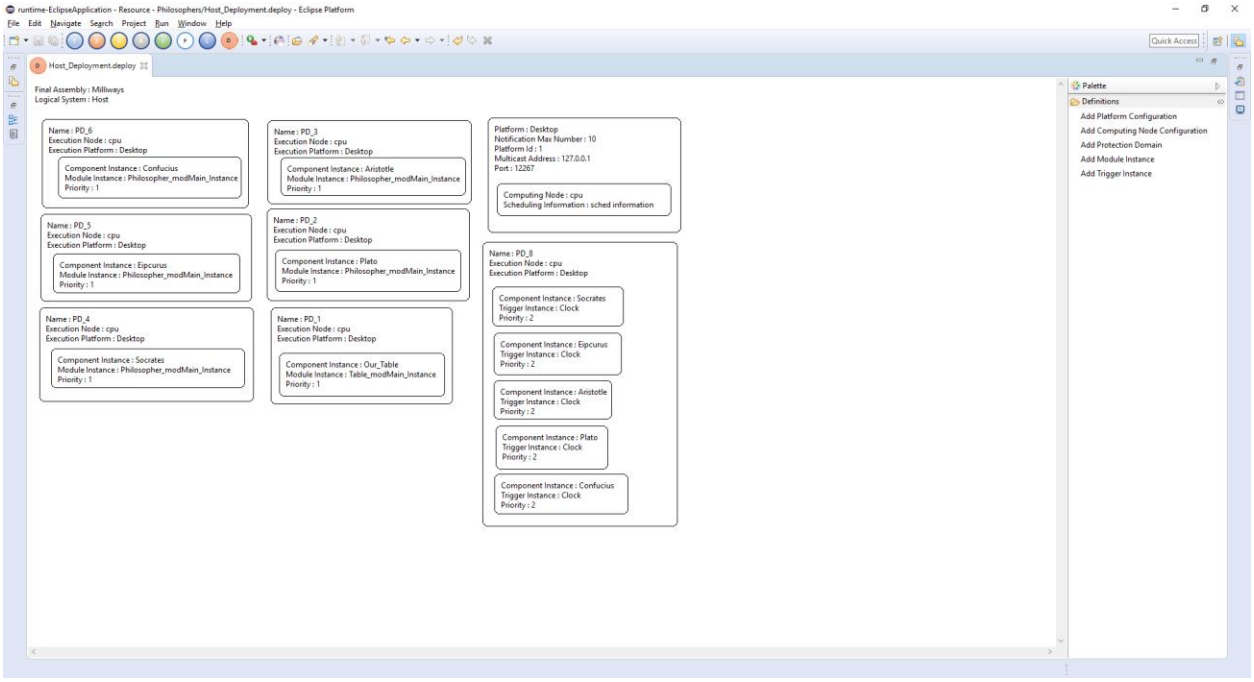
Select the Deployed instance that you want to move and modify the Protection Domain Property to the newly created Protection Domain name. The object gets transferred.



On moving out of control you can see that the deployed instance has been moved from one protection domain to another.



Moving all Trigger instances to the new protection domain and rearranging, the editor is as below:


runtime-EclipseApplication - Resource - Philosophers/Host_Deployment.deploy - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Quick Access
Host_Deployment.deploy
Final Assembly : Milliways
Logical System : Host
Name : PD_6
Execution Node : cpu
Execution Platform : Desktop
Component Instance : Confucius
Module Instance : Philosopher_modMain_Instance
Priority : 1
Name : PD_3
Execution Node : cpu
Execution Platform : Desktop
Component Instance : Aristotle
Module Instance : Philosopher_modMain_Instance
Priority : 1
Platform : Desktop
Notification Max Number : 10
Platform Id : 1
Multicast Address : 127.0.0.1
Port : 12267
Computing Node : cpu
Scheduling Information : sched information
Name : PD_5
Execution Node : cpu
Execution Platform : Desktop
Component Instance : Eipcurus
Module Instance : Philosopher_modMain_Instance
Priority : 1
Name : PD_2
Execution Node : cpu
Execution Platform : Desktop
Component Instance : Plato
Module Instance : Philosopher_modMain_Instance
Priority : 1
Name : PD_8
Execution Node : cpu
Execution Platform : Desktop
Component Instance : Socrates
Trigger Instance : Clock
Priority : 2
Component Instance : Eipcurus
Trigger Instance : Clock
Priority : 2
Component Instance : Aristotle
Trigger Instance : Clock
Priority : 2
Component Instance : Plato
Trigger Instance : Clock
Priority : 2
Component Instance : Confucius
Trigger Instance : Clock
Priority : 2
Name : PD_4
Execution Node : cpu
Execution Platform : Desktop
Component Instance : Socrates
Module Instance : Philosopher_modMain_Instance
Priority : 1
Name : PD_1
Execution Node : cpu
Execution Platform : Desktop
Component Instance : Our_Table
Module Instance : Table_modMain_Instance
Priority : 1
Palette
Definitions
Add Platform Configuration
Add Computing Node Configuration
Add Protection Domain
Add Module Instance
Add Trigger Instance

5 Additional Features

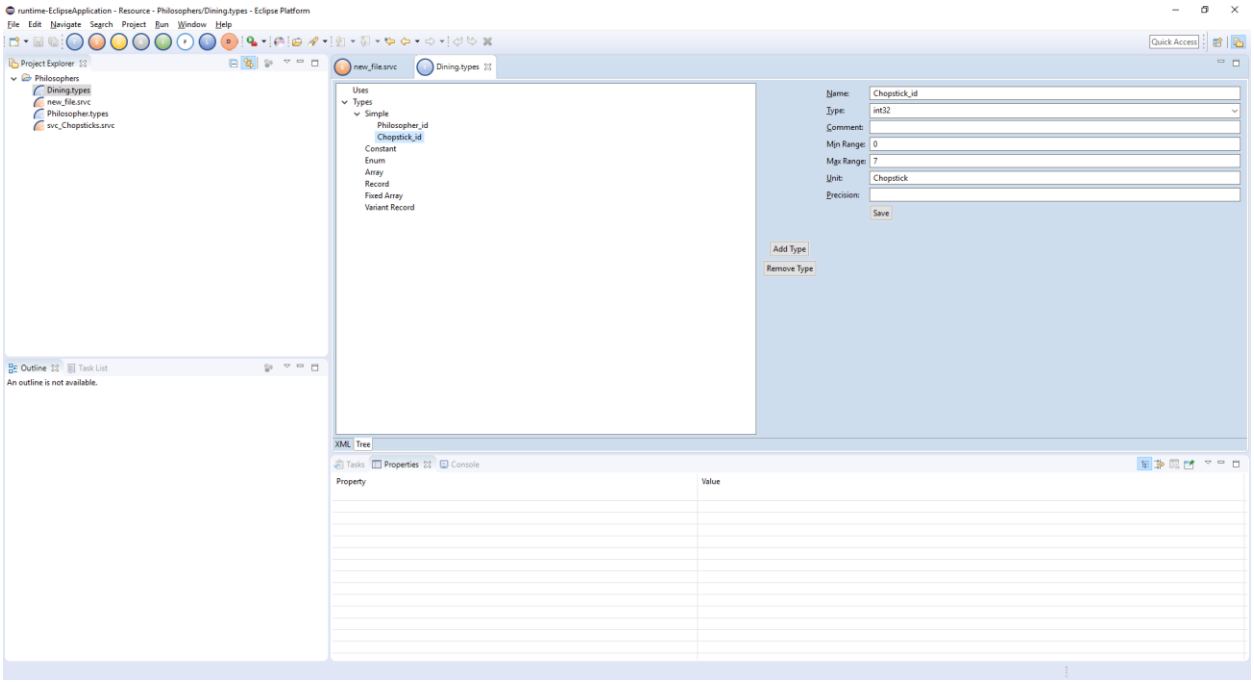
5.1 Edit and Remove on Tree Editors

After you create design elements using the three tree based editors (Type, Service and Component Definition), you can select any of the level 3 nodes on the trees (which are the design elements), you will have the ability to modify the attributes or remove by selecting remove option.

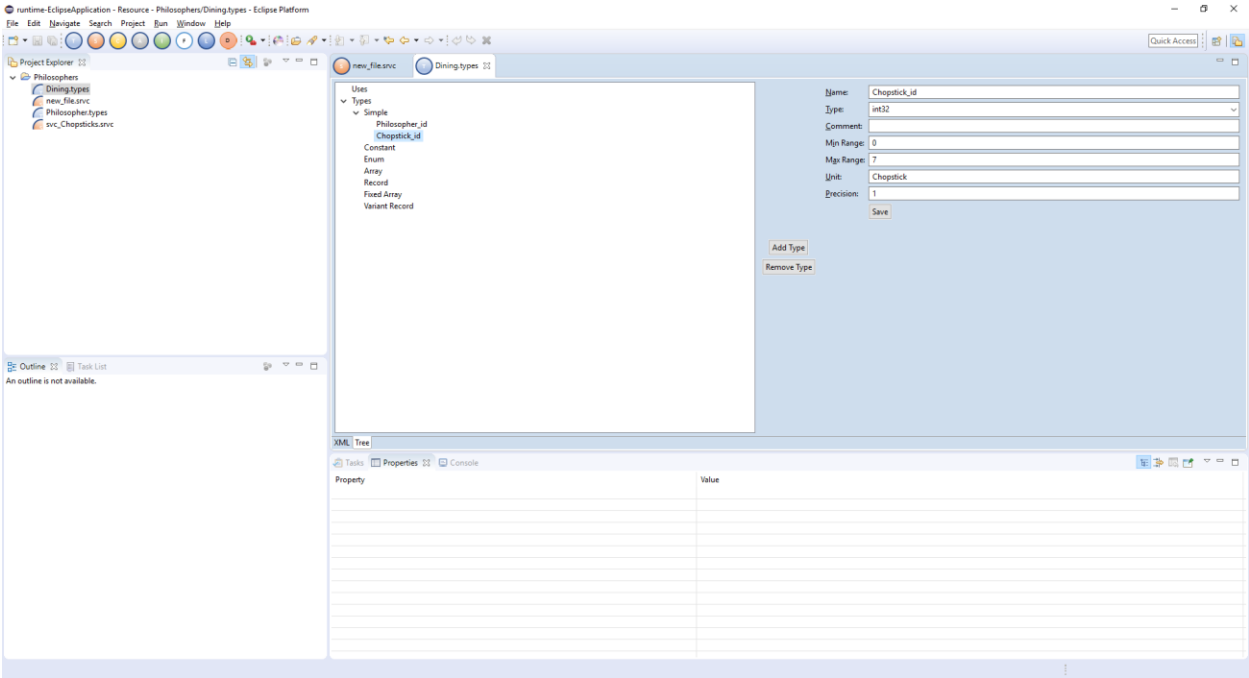
5.1.1 Edit

For Example: On the Types Editor to add a new value to the precision attribute of simple type, we do the following:

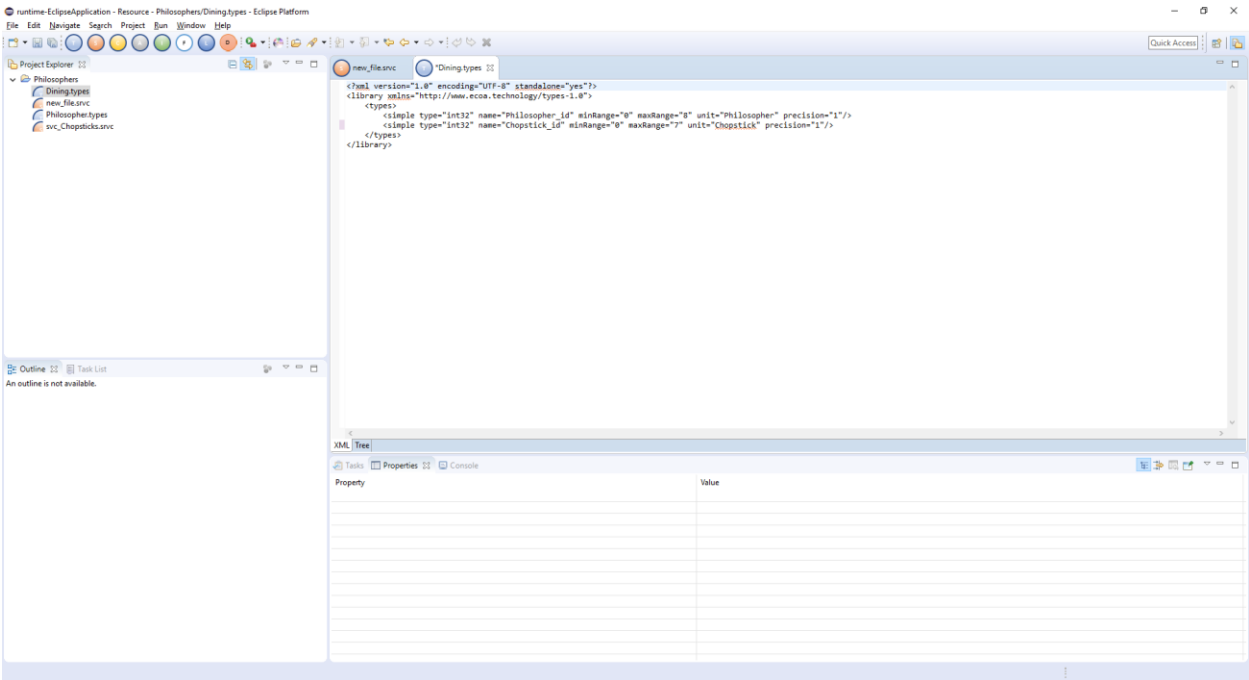
Select the Simple Type to be Modified



Modify the Precision Attribute and click Save.



Verify the generated XML.

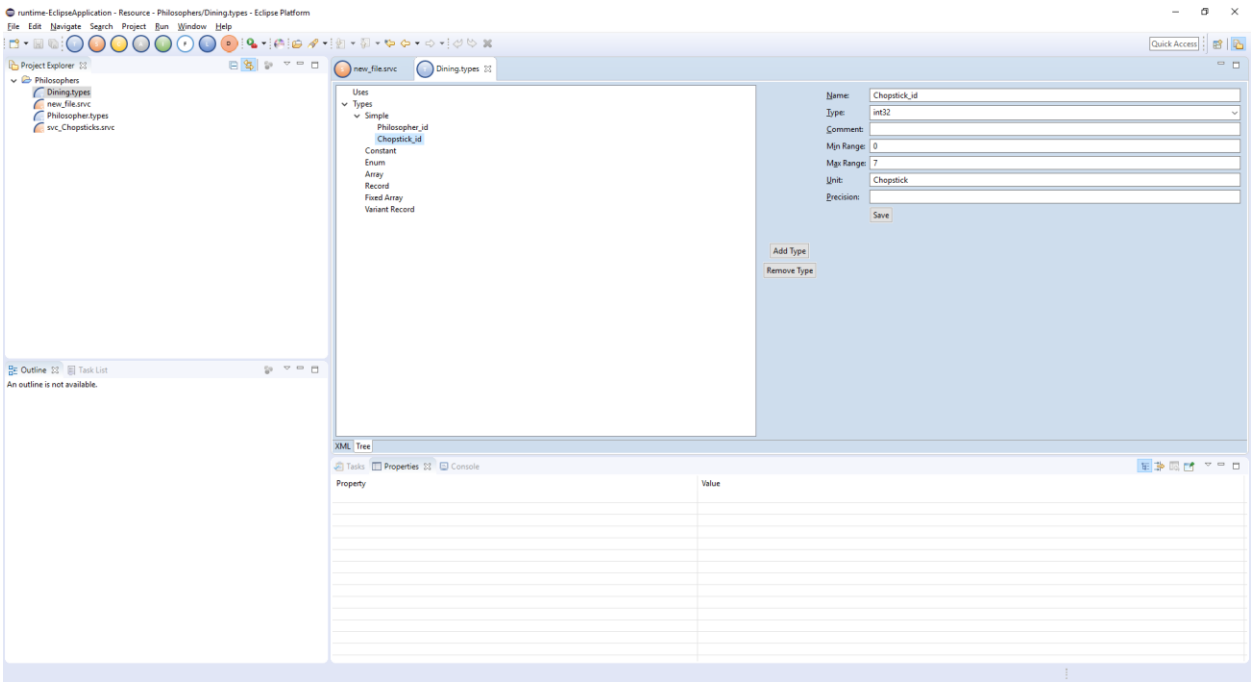


Similar process can be followed with any definition on the Tree Editors.

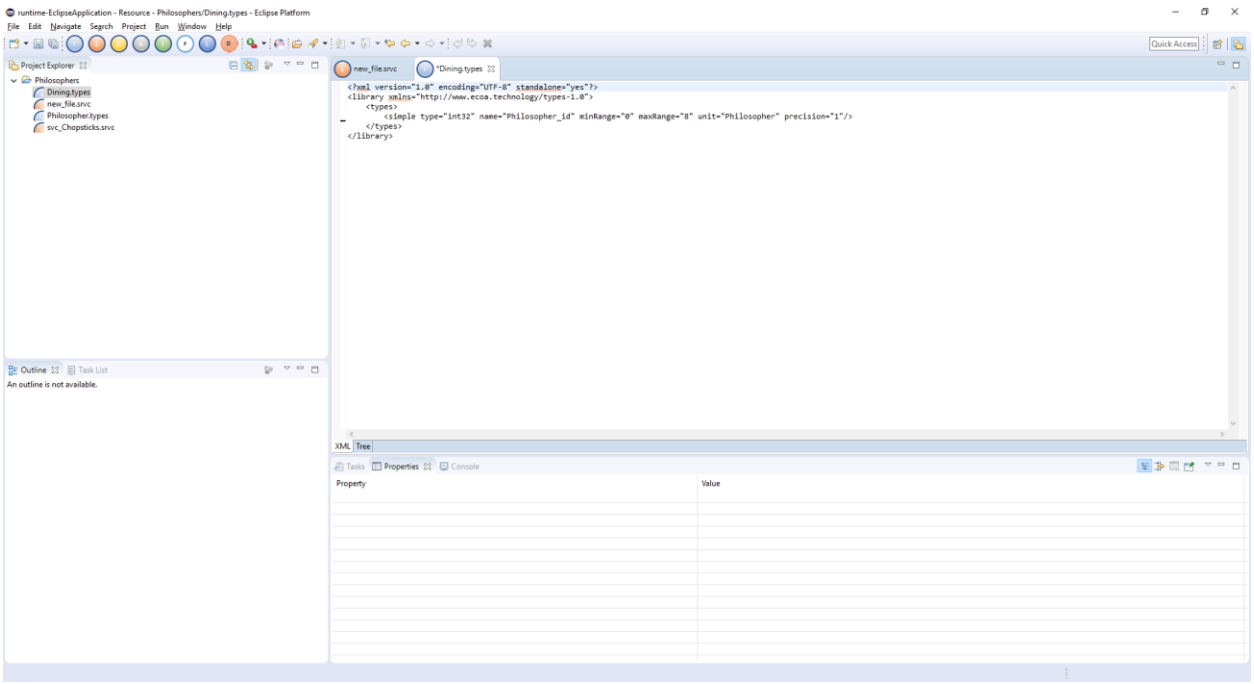
5.1.2 Remove

For Example: On the Types Editor to remove a type, we do the following:

Select the Simple Type to be Removed



Select Remove Type. Verify the generated XML

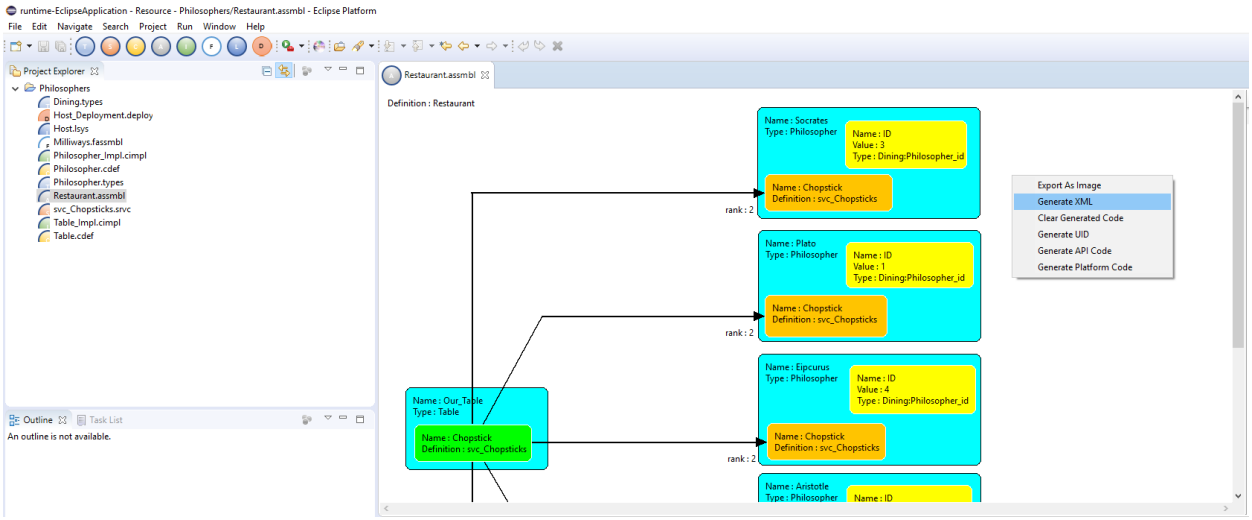


Similar process can be followed with any definition on the Tree Editors.

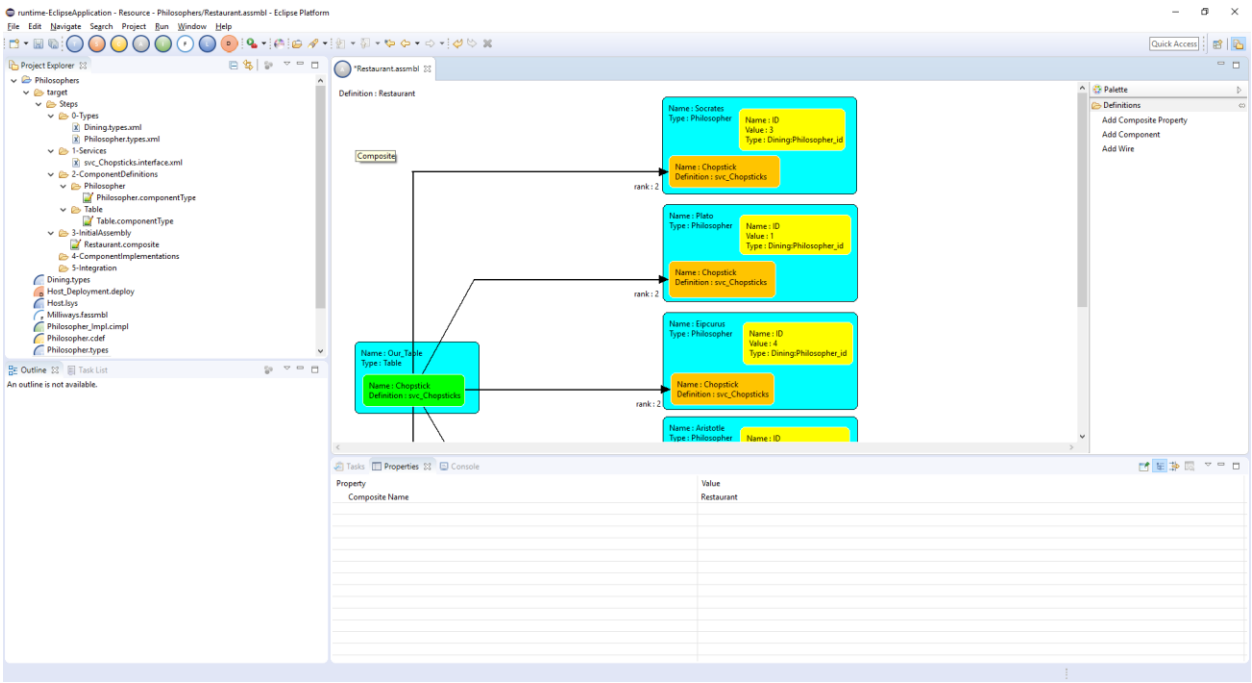
5.2 Generation of XML

Once the design of all graphical content is completed, open editor by editor and on the context menu select Generate XML Option.

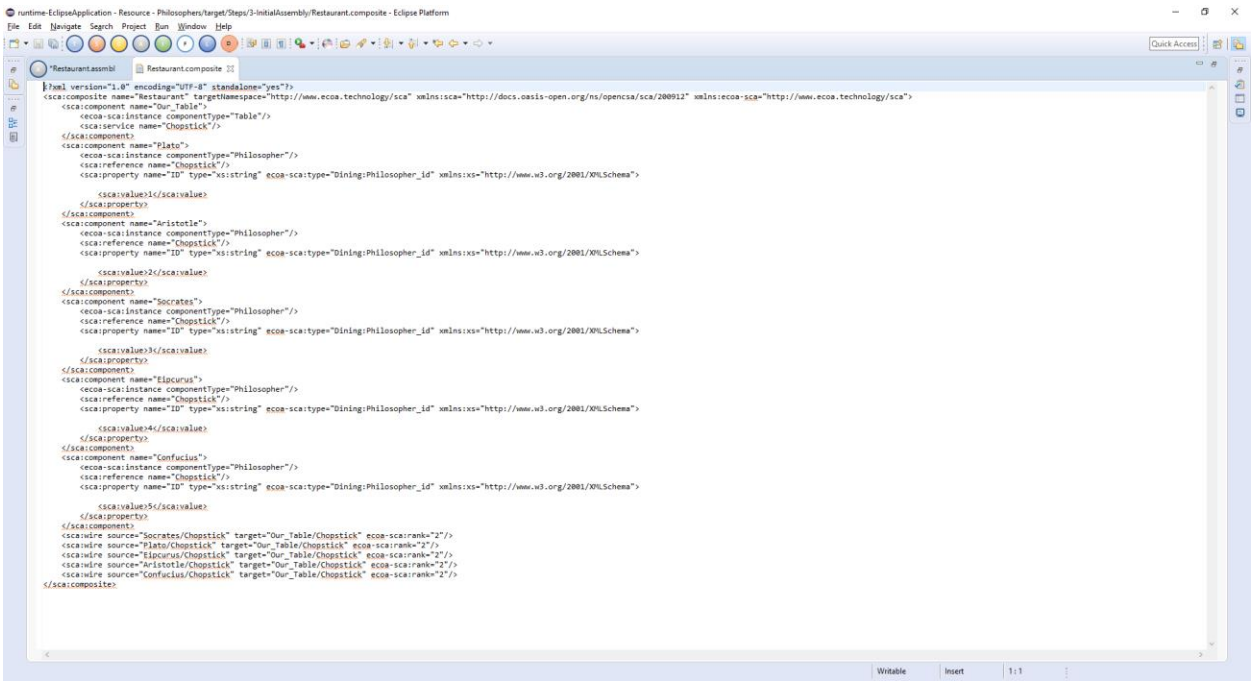
For Initial Assembly:



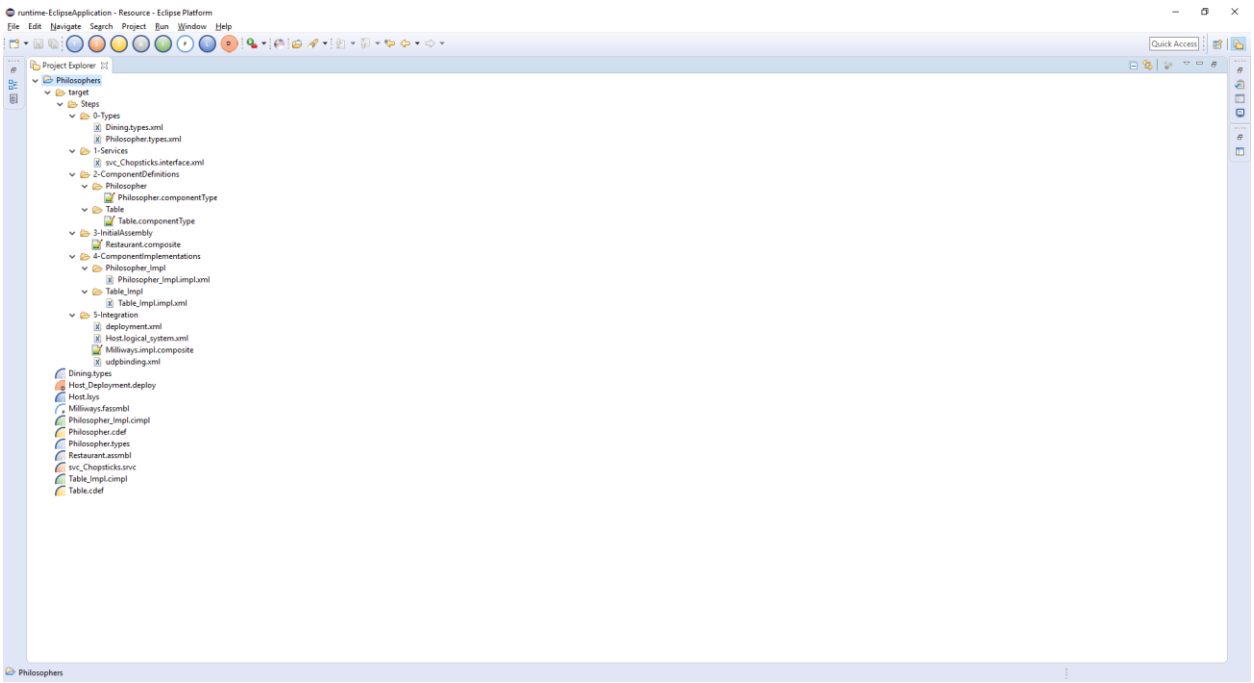
A target folder will be created under the project structure and the generated files will be placed under that folder.



The generated XML File:



Repeat the steps for each of Component Implementation, Final Assembly, Logical System, Deployment editors. The completed folder structure of target is as below:

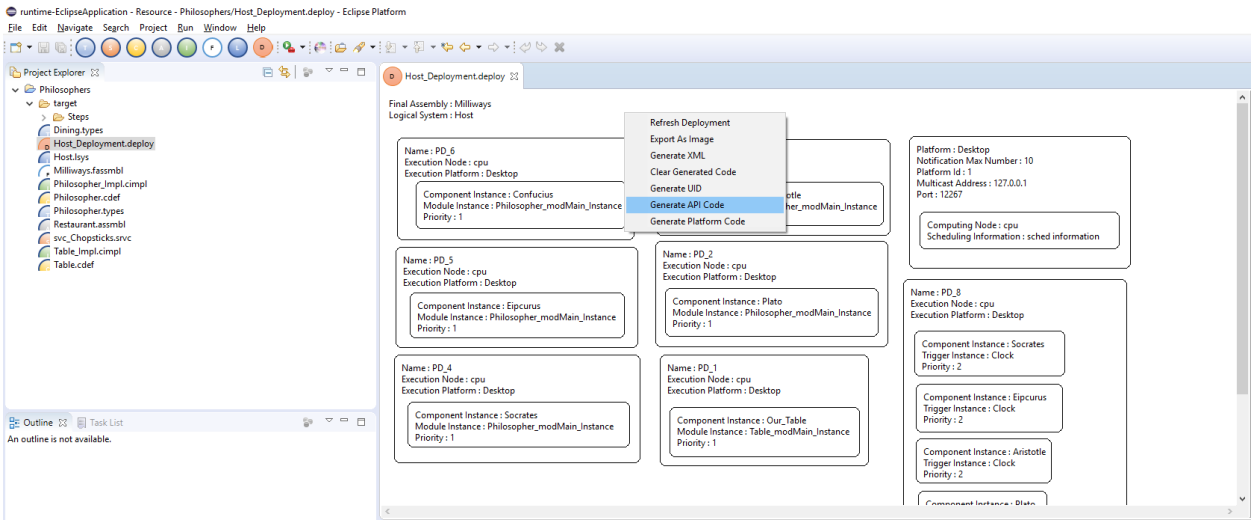


The Complete Generated XML is as below:

Steps.zip

5.3 Generation of API Codegen

Select any editor and select Generate API Codegen option from the Context Menu



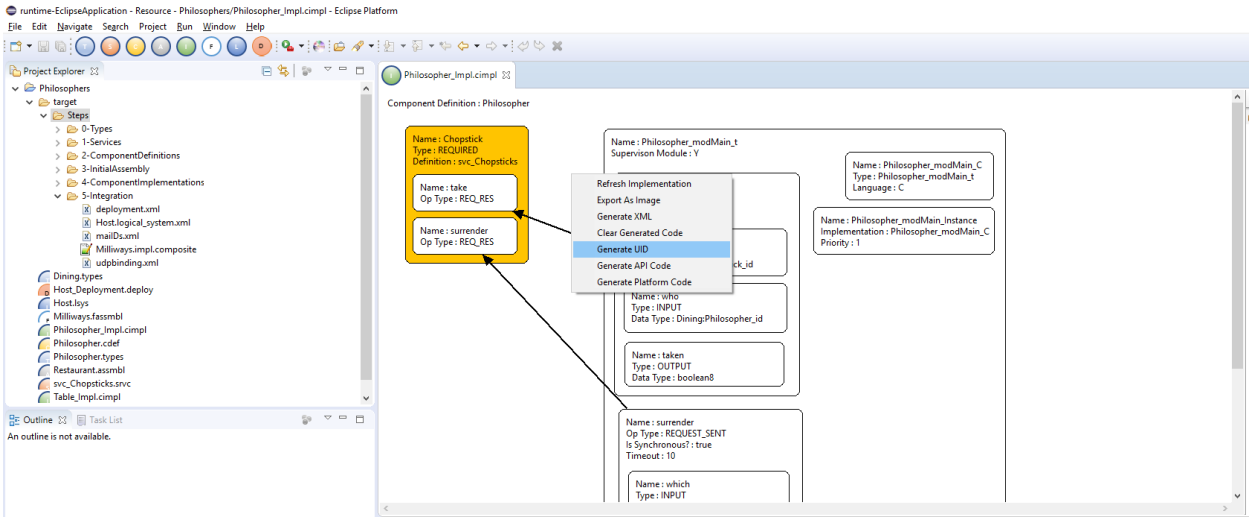
And generated code will be merged onto the steps folder. Below is the Steps folder with Generated API Code.

Steps.zip

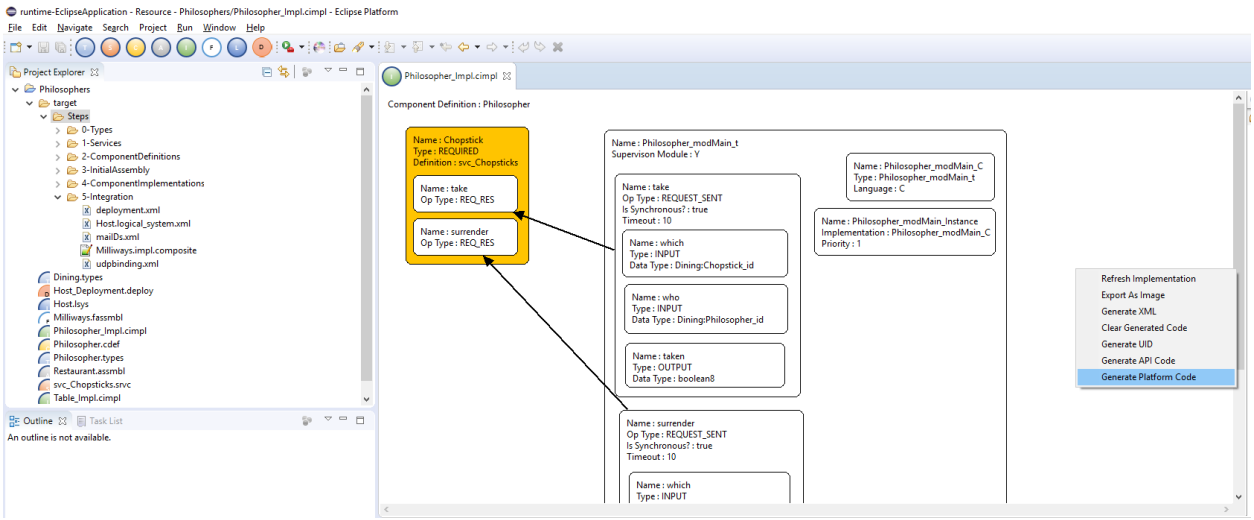
In case of any errors run Clear Generated Code from Context Menu and Retry.

5.4 Generation of Platform Codegen

Select Any Editor and select generate UID option from Context Menu.



Select any editor and select Generate Platform Codegen option from the Context Menu



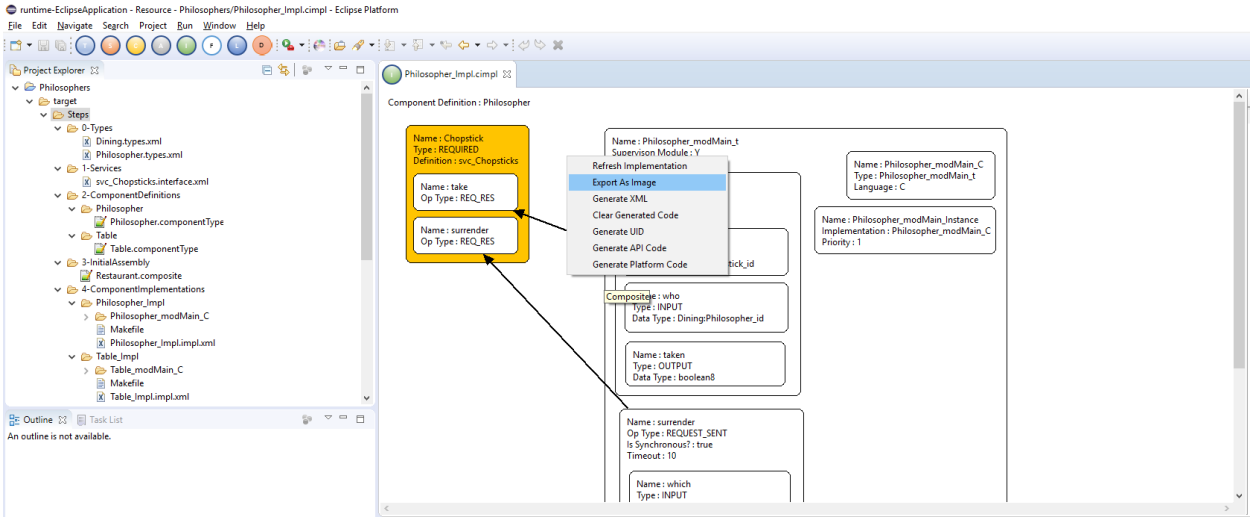
And generated code will be merged onto the steps folder. Below is the Steps folder with Generated Platform Code.

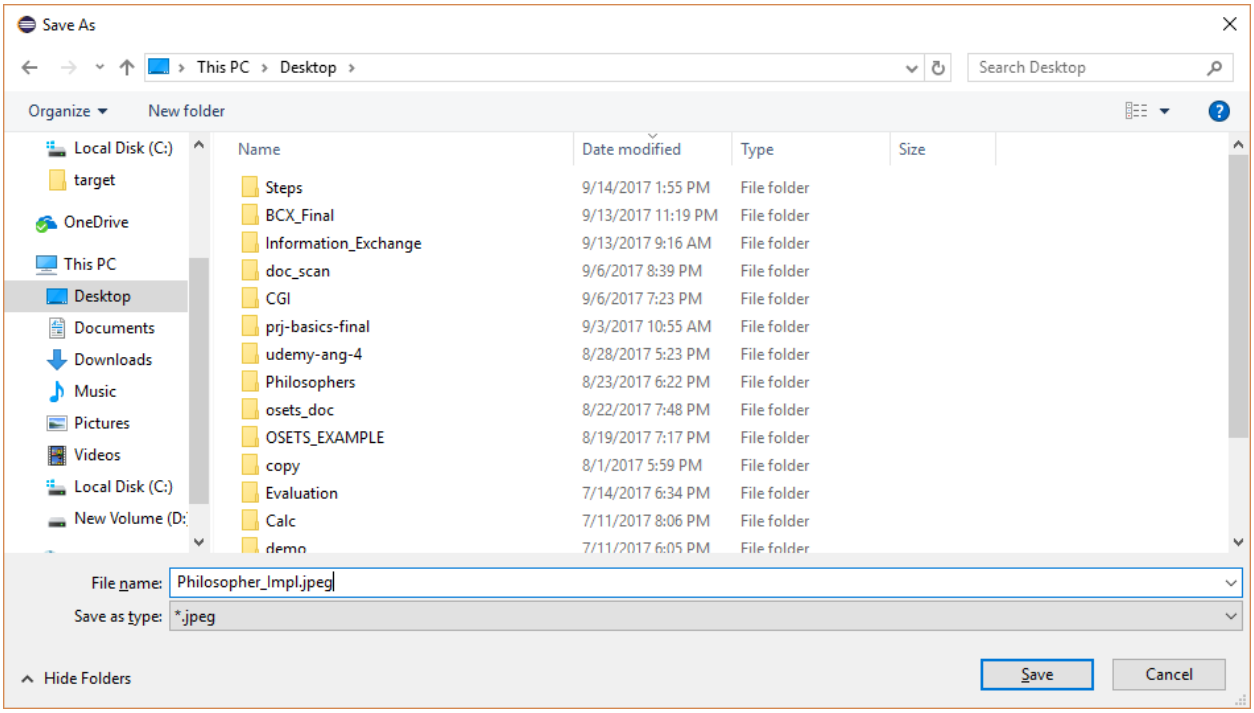
Steps.zip

In case of any errors run Clear Generated Code from Context Menu and Retry.

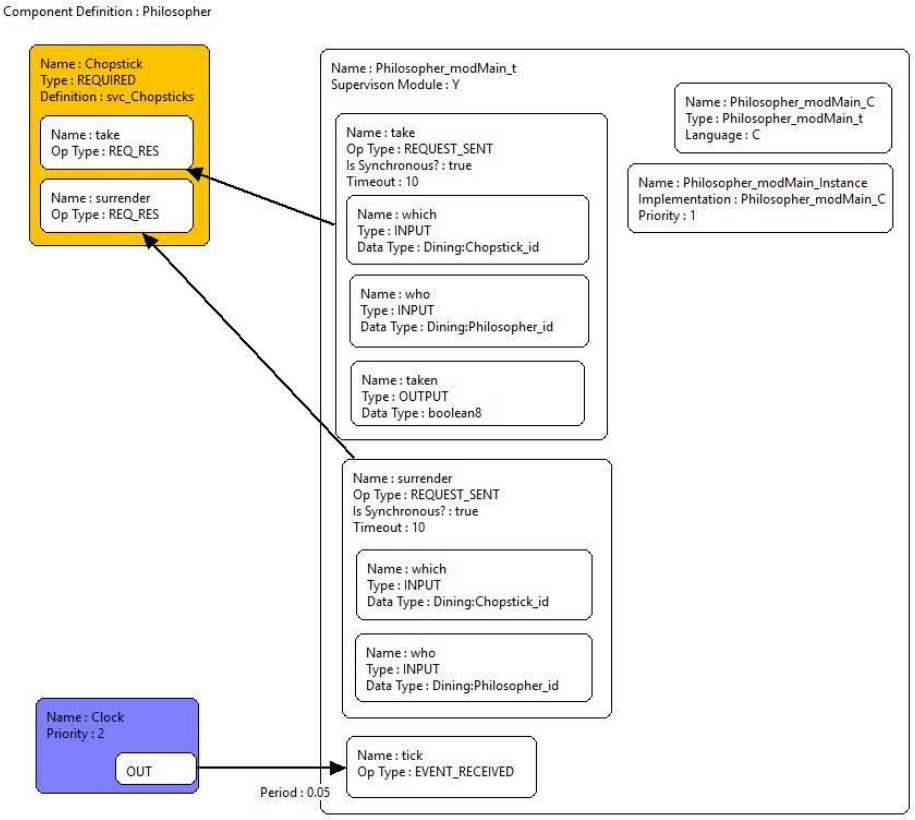
5.5 Export as Image

For any of the Graphical Editors, we have the option to export the editor as a JPEG Image. For this simply select Export as Image from the context menu. Provide a file location and the editor content will be sent out as an Image.





Generated Image File:



5.6 Refresh of Editors

While we are on our merry way implementing complex systems, one of the most common occurrences in a software system is Change. What if the underlying type changes, what if the underlying service definition changes, what if the underlying component definition changes etc.

When such changes occur, an intelligent modelling tool will provide capabilities to refresh the editors and check if any underlying information has changed. These changes can be of two types:

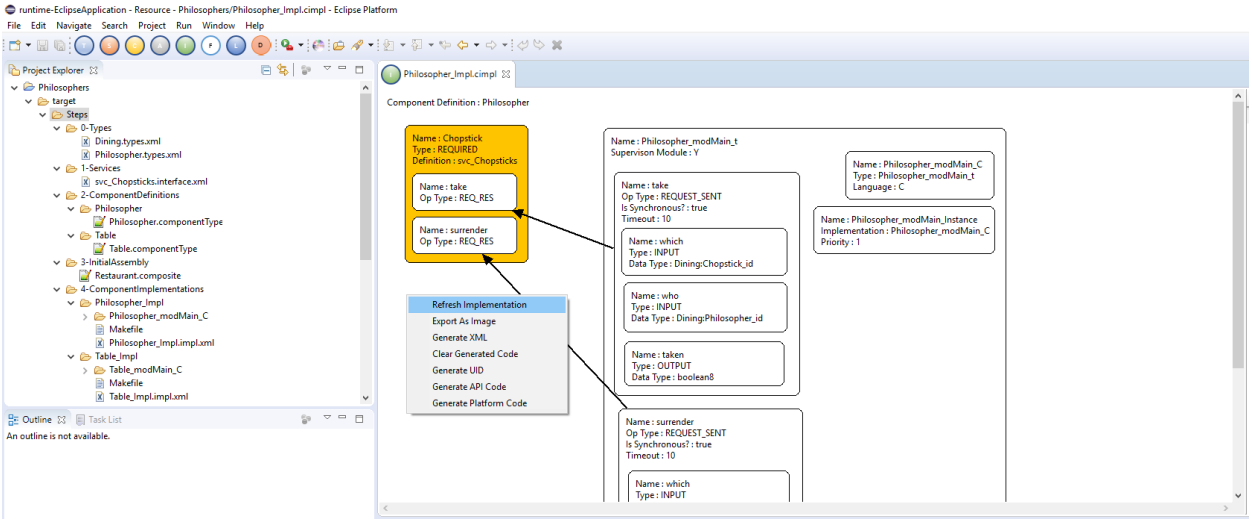
1) Non-conflicting changes – does not conflict with any of the existing design elements (like component property added on component definition)
2) Conflicting changes – conflicts with existing design (like service operation deleted or modified)

Keeping the current timelines and possibility, we have implemented Just non-conflicting changes described in the next 3 sections.

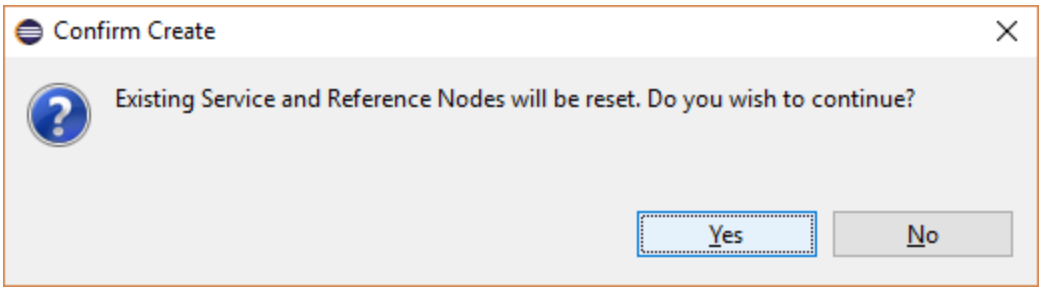
5.6.1 Refresh of Component Implementation from Definition

This refreshes the Service and Service Operations and Properties from the underlying Component definition onto the component implementation. Any existing links established to service operation nodes will be deleted.

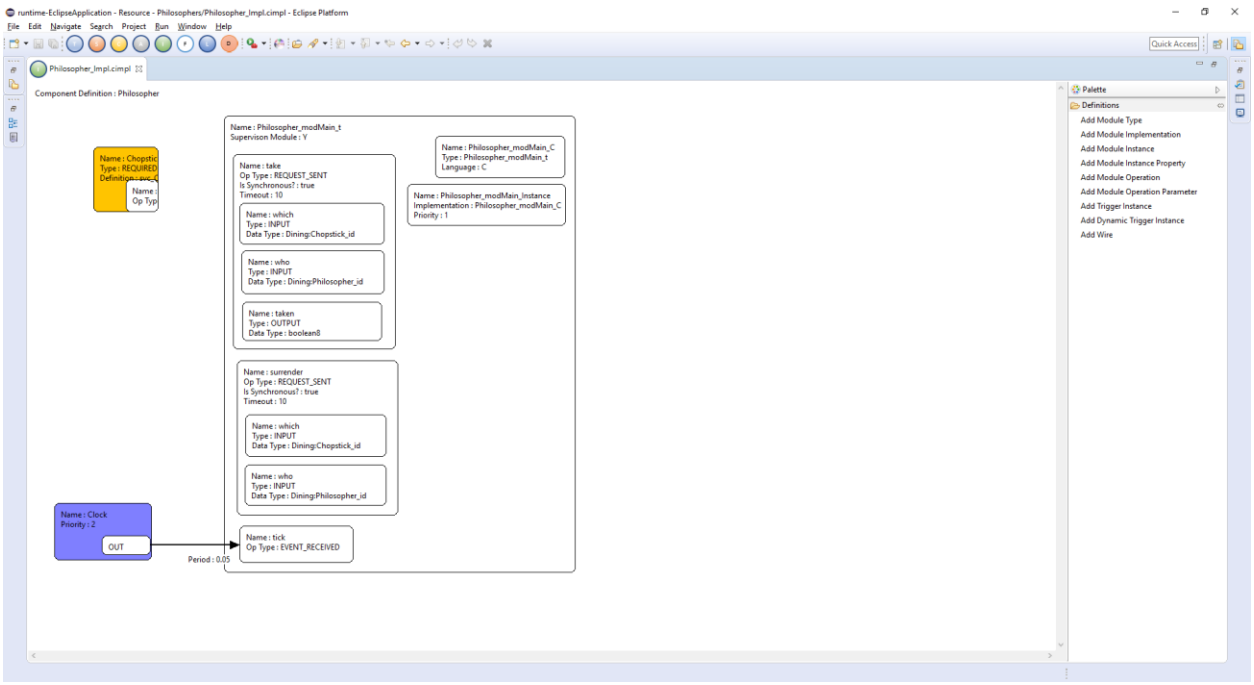
Select Refresh Implementation from the context menu.



Confirm the change:



The refreshed editor is as below:

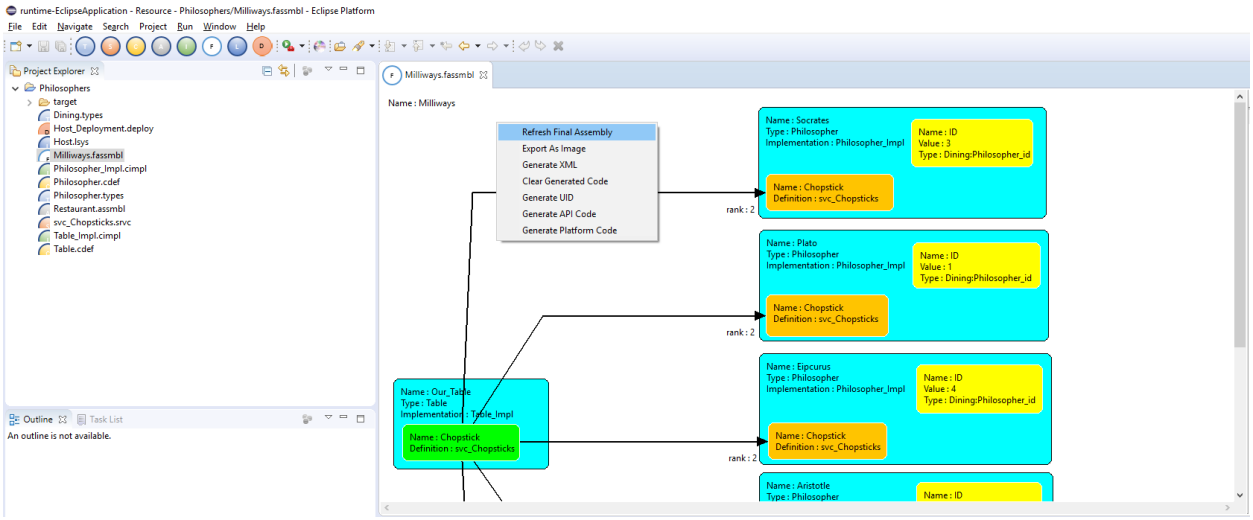


Note that only Service and Service Operations and related links are impacted.

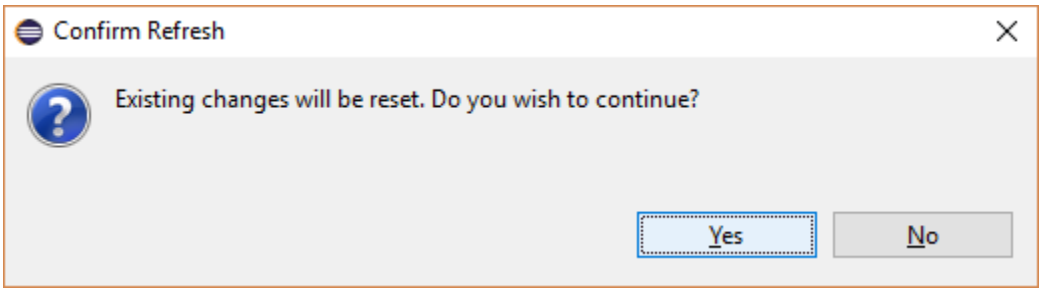
5.6.2 Refresh of Final Assembly from Initial Assembly

This refreshes the Final Assembly from the underlying Initial Assembly. All Implementation assignments to components are deleted and a complete refresh is done.

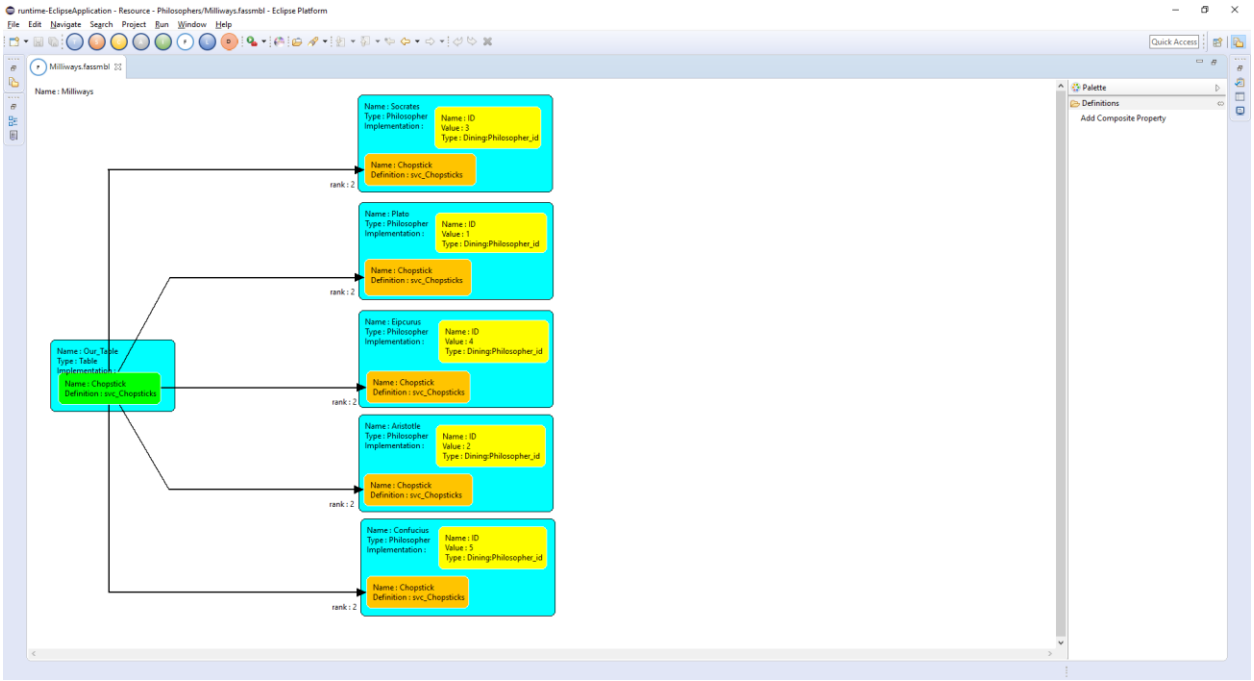
Select Refresh Final Assembly option from the context menu.



Confirm the refresh.



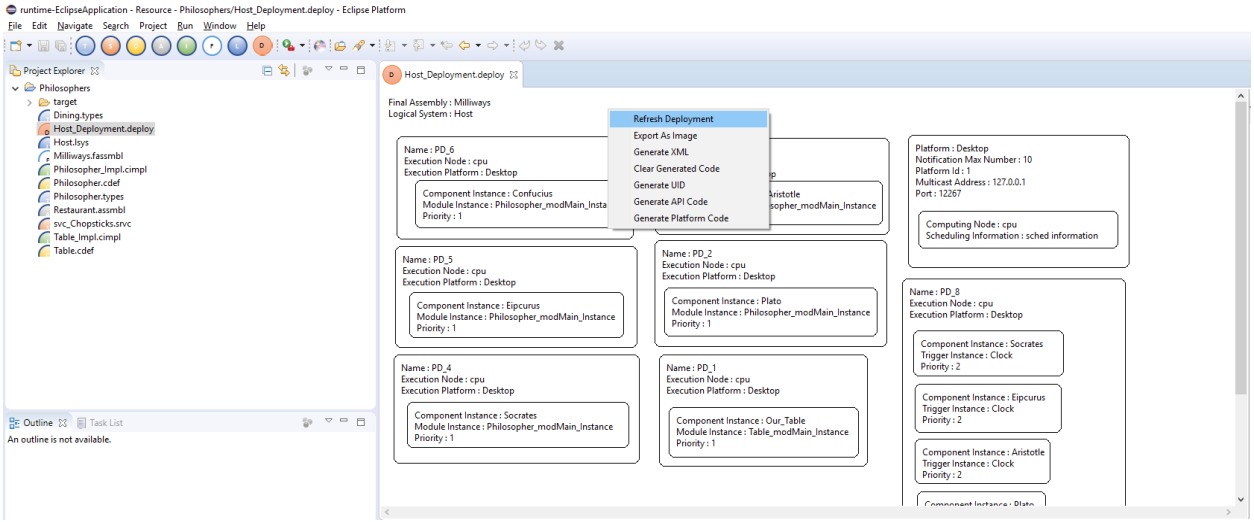
The refreshed editor is below:



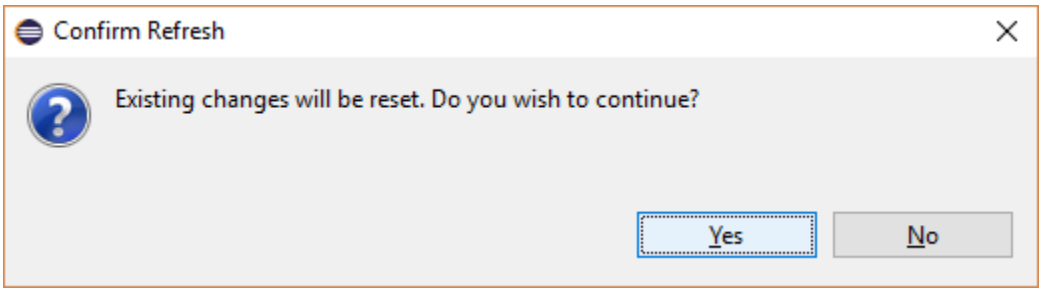
5.6.3 Refresh of Deployment from Logical System and Final Assembly

This refreshes Deployment from changes done to Final Assembly and Logical Systems. Note that this is a non-reversible change. Any modifications done will be replaced and must be redone.

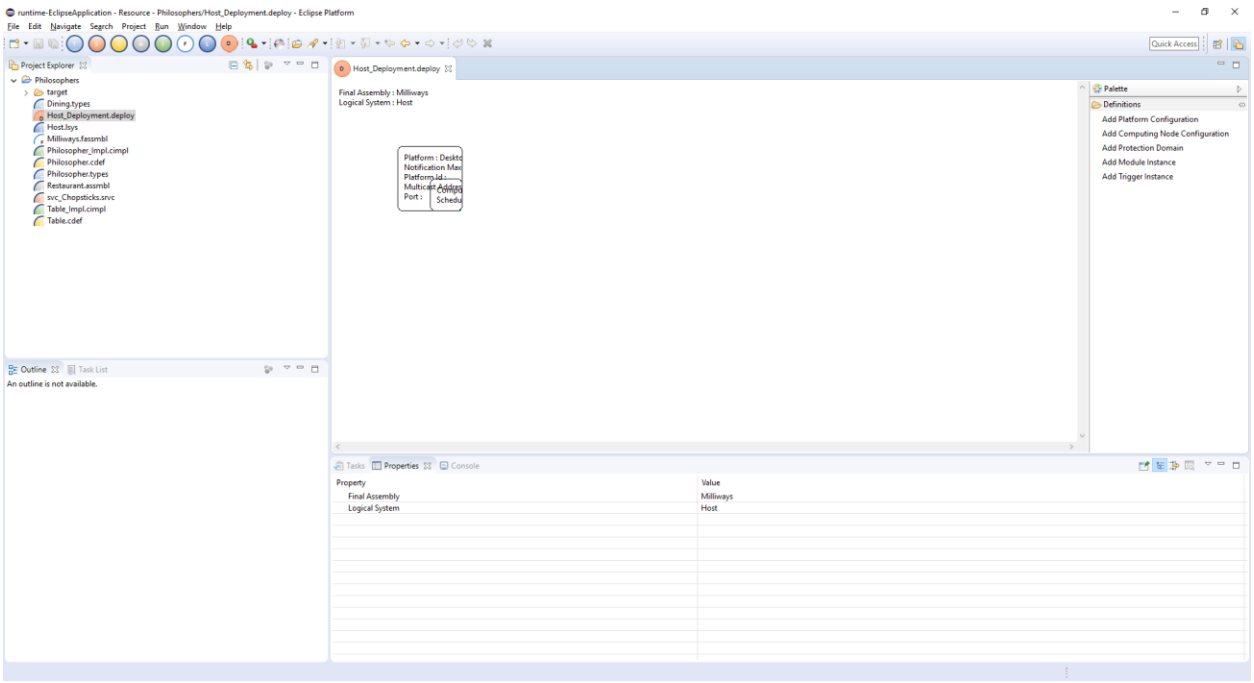
Select Refresh Deployment from the Context Menu



Confirm the Refresh:



The Refreshed editor is as below:



Note that the elements have to be re-arranged.

6 Epilogue

6.1 Identifying Toolbar Options

This is to Identify and familiarize to the OSETS Eclipse Wizard Initiators. The below are the Toolbar actions and their Identification:

1)  - Types Editor: Can also be initialized using: Right Click on Project -> Select New Other -> Opens Dialog -> Select ECOA Wizards -> Select Types Editor
2)  - Services Editor: Can also be initialized using: Right Click on Project -> Select New Other -> Opens Dialog -> Select ECOA Wizards -> Select Services Editor
3)  - Component Definition Editor: Can also be initialized using: Right Click on Project -> Select New Other -> Opens Dialog -> Select ECOA Wizards -> Select Component Definition Editor
4)  - Initial Assembly Editor: Can also be initialized using: Right Click on Project -> Select New Other -> Opens Dialog -> Select ECOA Wizards -> Select Initial Assembly Editor
5)  - Component Implementation Editor: Can also be initialized using: Right Click on Project -> Select New Other -> Opens Dialog -> Select ECOA Wizards -> Select Component Implementation Editor
6)  - Final Assembly Editor: Can also be initialized using: Right Click on Project -> Select New Other -> Opens Dialog -> Select ECOA Wizards -> Select Integration - Final Assembly Editor
7)  - Logical Systems Editor: Can also be initialized using: Right Click on Project -> Select New Other -> Opens Dialog -> Select ECOA Wizards -> Select Integration – Logical Systems Editor
8)  - Deployment Editor: Can also be initialized using: Right Click on Project -> Select New Other -> Opens Dialog -> Select ECOA Wizards -> Select Integration – Deployment Editor

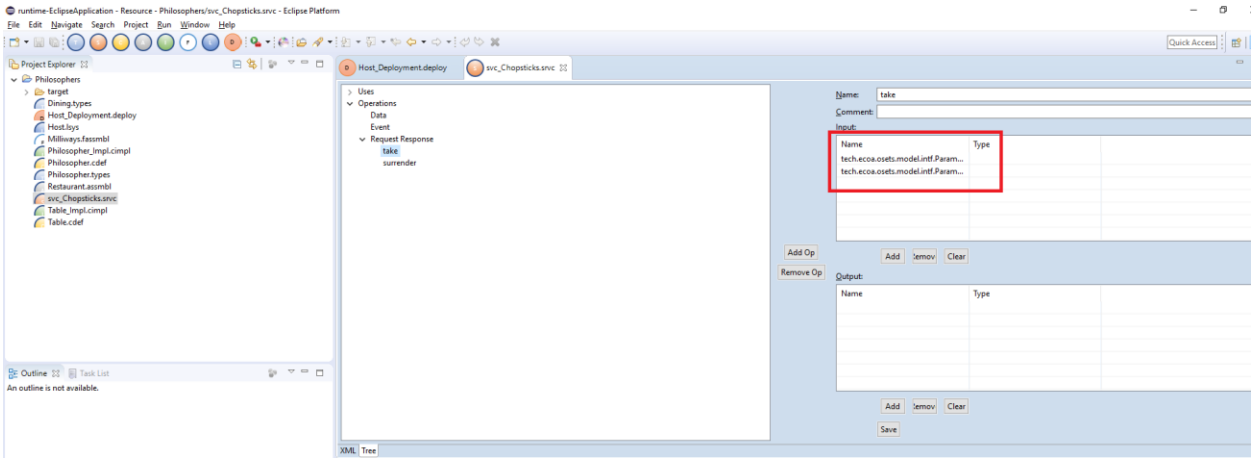
Appendix A Known Issues

A.1 Types Editor

1) Nesting of Libraries is not Supported
2) Only basic XSD level validations are done. Name duplications are left out and the users must follow the discipline and name the types differently.
3) If a reference to a constant is to be entered as a value this must be entered manual using the correct ECOA format - %<*constant name*>%; and this entry is not checked against defined constants.
4) No re-ordering of Types. They are added in the order that they are created. This might get tricky in complex situations where there are dependencies in the same file as a simple type on a record. Again, user disciple should be followed.

A.2 Service Editor

1) Elements in Parameter Table are showing as Objects but not Strings



Workaround is to double click on the cells so that the view is refreshed.

2) Operation parameters, types are not in order. Need to implement filter.

A.3 Component Definition Editor

A.4 Initial Assembly Editor

1) Duplicate Undo confirmation dialogs when changing the component type. Framework limitation.

A.5 Component Implementation Editor

1) The editor presents Module Type, Implementation, and Instance relationships along with Module Wiring. Because Module Operations are defined as part of the Module Type, the wiring appears to go to Module Types rather than Module Instances. The Module Instance actually connected to is a property of the wire (in the design tool). For simple designs where there are

only single instances of a Module Type there should be no issues; however, complex designs making use of multiple instances of a Module Type the resulting wiring may be unclear.
2) There is no graphical indicator on a wire to make it clear whether the source/target Module Instance property has been set.
3) When adding parameters to a Module Operation there is no mechanism for altering the order they appear in the XML (and subsequently the code). Module parameters **must** be added to an operation in the order desired.
4) When adding parameters to a Module Event operation – the tool offers the choice of setting them as outputs – but this would be invalid.
5) It is not possible to add additional parameters to a Dynamic Trigger Instance. If parameters need to be used the user will have to modify the XML after it has been generated.

A.6 Final Assembly Editor

1) Implemented dropdowns for Component Implementation Selection. Additional validation of Component Type and restricting Implementations of that Type is not done.

A.7 Logical System Editor

A.8 Deployment Editor

1) Deployment Editor, implemented dropdown selections for platform and computing node. But must implement computing nodes related to platform after selection.

A.9 Code Generation

1) API code generation is done for all Component Implementations in the project for which XML has been generated. Selectable Code Generation is not an option.
2) The use of constant references to define a minimum range value for a type leads to erroneous code in Defaulter.c. The constant reference is placed in the code literally using the ECOA syntax %<*constant_name*>%, instead of being properly formed as <*library_name*>__<*constant_name*>.
3) Using variant records results in errors in Defaulter.c, ELI_In_deserialiser.c and ELI_Out_serialiser: references to the constants ECOA__TRUE and ECOA__FALSE are implemented as ECOA__true and ECOA__false.
4) The use of Dynamic Trigger Instances leads to errors in container support code:

   Files: <*component_instance*>_<*dynamic_trigger_instance*>_Controller.h and <*component_instance*>_<*dynamic_trigger_instance*>_Controller.c
   Change #include statement reference
   <*component_implementation*>_<*dynamic_trigger_instance*>_DynTrigModule.h
   should be
   <*component_instance*>_<*dynamic_trigger_instance*>_DynTrigModule.h.
   File: <*component_instance*>_<*dynamic_trigger_instance*>_Controller.c
   Use of
   <*component_instance*>_<*dynamic_trigger_instance*>_OUT_UID
   should be

<*component_implementation*>_<*dynamic_trigger_instance*>_OUT_UID

In routine <*component_instance*>_<*dynamic_trigger_instance*>__out_send use of

<*component_implementation*>_..._event_received

should be

<*component_instance*>_..._event_received

File: <*protection_domain*>_Timer_Event_Handler.c

Add #include statement references

<component_instance>_<dynamic_trigger_instance>_Controller.h.

A.10 General

1) All Graphical Editor properties are sorted alphabetically. This is a current restriction in eclipse framework.
2) Duplicate Name Validations are not in place. Suggested as enhancement as guiding rules were not completely established.

Appendix B Dining Philosophers Example Script

This annex describes a series of ECOA design steps that implement a version of the Dining Philosophers example given on the ECOA website (http://www.ecoa.technology/).

B.1 Data Libraries

Create Library **Dining** with Simple types

- **Philosopher_id** based on **int32** range **1**..**8**. with units of '**Philosopher**' and precision = **1**; and
- **Chopstick_id** based on **int32** range **0**..**7** with units of '**Chopstick**'.

Create Library **Philosopher** with Enumerated type

- **State** based on **uint8** with (UNDEFINED, GETTINGSTICKS, EATING, SURRENDERING, THINKING)

B.2 Services

Create a Service **svc_Chopsticks** with two Request-Response operations:

- **take** with input parameters **which** (**Dining_types:Chopstick_id**) and **who** (**Dining_types:Philosopher_id**); and output parameter **taken** (**boolean8**).
- **surrender** with input parameters **which** (**Dining_types:Chopstick_id**) and **who** (**Dining_types:Philosopher_id**).

B.3 Component Definition

Create a Component Definition **Table** that provides service **Chopstick** of type **svc_Chopsticks**.

Create a Component Definition **Philosopher** that requires service **Chopstick** of type **svc_Chopsticks**; and has property **Person_ID** of type **Dining_types:Philosopher_id**.

B.4 Initial Assembly

Create Initial Assembly **Restaurant** and add Component Instance **Our_Table** of type **Table**.

Add Component Instances **Plato**, **Aristotle**, **Socrates**, **Epicurus**, **Confucius** of type **Philosopher**. Set the **Person_ID** property for each of these to **1**, **2**, **3**, **4**, **5** respectively.

Wire **Our_Table**'s Provided Service to each of the **Philosopher**'s Required Service.

B.5 Component Implementation

Create a Component Implementation **Table_Imp** of Component Definition **Table**.

Add a Module Type **Table_modMain_t** as a **supervision module** with Module Operations:

- **take** (Request_Received - with in parameters **which** : **Dining_types:Chopstick_id**, **who** : **Dining_types:Philosopher_id** and out parameter **taken** : **boolean8**); and
- **surrender** (Request_Received - with in parameters **which** : **Dining_types:Chopstick_id**, **who** : **Dining_types:Philosopher_id**).

> *NOTE: The order in which Module Parameters are added is important for code-generation where parameters are passed by position.*

Add a Module Implementation **Table_modMain_C** of **Table_modMain_t**, with programming language set to '**C**'.

Add a Module Instance **Table_modMain_Instance** of **Table_modMain_C** with a (relative) Priority of **1**.

Connect Module Wiring

- **Chopstick/take** to **Table_modMain_Instance/take**, and
- **Chopstick/surrender** to **Table_modMain_Instance/surrender**.

NOTE: Need to set Module Instance on the wires.

Create a Component Implementation **Philosopher_Imp** of Component Definition **Philosopher**.

Add a Module Type **Philosopher_modMain_t** as a **supervision module** with Module Operations:

- **take** (Request_Sent – **synchronous** with no timeout, in parameters **which** : **Dining_types:Chopstick_id**, **who** : **Dining_types:Philosopher_id** and out parameter **taken** : **boolean8**);
- **surrender** (Request_Sent - **synchronous** with no timeout, in parameters **which** : **Dining_types:Chopstick_id**, **who** : **Dining_types:Philosopher_id**); and
- **Tick** (Event_Received).

Add a Module Implementation **Philosopher_modMain_C** of **Philosopher_modMain_t**, with programming language set to '**C**'.

Add a Module Instance **Philosopher_modMain_Instance** of **Philosopher_modMain_C** with a (relative) Priority of **2**.

Add Module Property **Id** of type **Dining_types:Philosopher_Id**. Set Module Property value to **$Person_ID** (the Component Property value).

Add a Trigger Instance **Clock** with a (relative) Priority of **1**.

Connect Module Wiring

- **Philosopher_modMain_Instance/take** to **Chopsticks/take**,
- **Philosopher_modMain_Instance/surrender** to **Chopsticks/surrender**, and
- **Clock** to **Philosopher_modMain_Instance/tick** with a period **0.5**.

NOTE: Need to set Module Instance on the wires.

B.6 Logical System

Create a Logical System **Desktop** and add Computing Platform **Host**.

NOTE: OSETS only supports a simple Logical System – the ECOA XML does not provide a standardized way of capturing information for communication links between processors, nodes, and platforms.

Add the Computing Node **cpu**, to **Host**, with the properties endianness=**LITTLE**, os=**linux**, availableMemory =**1** (1 GB – a nominal value), moduleSwitchTime=**10** (10 µs - a nominal value).

NOTE: The Code Generators are not thought to use information other than the OS value – but they are required for the XML to be 'legal'. The OS value in this case needs to be linux to support code generation for cygwin.

Add a Logical Processor to **cpu** with the properties number=**1**, type="**X86_64**", stepDuratiuon=**1020** (1020 ns - a nominal value).

NOTE: The Code Generators are not thought to use this information – but they are required for the XML to be 'legal'.

B.7 Final Assembly & Deployment

Create Final Assembly **Milliways** based on Initial Assembly **Restaurant**.

Assign Component Implementations to Component Instances: **Table_Imp** => **Our_Table**; **Philosopher_Imp** => **Plato**, **Aristotle**, **Socrates**, **Epicurus**, and **Confucius**.

Create Deployment **Demo_1** using Final Assembly **Milliways** and logical System **Desktop**.

For Deployment **Demo_1** set the Platform Configuration for **Desktop** to have a Maximum Notifications of **8**.

Set Multicast Address as **127.0.0.1**, set Port as **60428**, set Platform Number as **1**.

Set Computing Node **Host** to have Scheduling Information "**Rate Monotonic**".

Assign Module Instances to Protection Domains - move all Component/Module Instances to Protection Domain **PD_1.**

Set Protection Domain **PD_1** to execute on Computing Platform/Node **Host/cpu**.

Allocate Component/Module Instances:

- **Our_Table/Table_modMain_Instance** priority **52**;
- **Plato/Philosopher_modMain_Instance** priority **50**;
- **Plato/Clock** priority **51**;
- **Aristotle/Philosopher_modMain_Instance** priority **50**;
- **Aristotle/Clock** priority **51**;
- **Socrates/Philosopher_modMain_Instance** priority **50**;
- **Socrates/Clock** priority **51**;
- **Epicurus/Philosopher_modMain_Instance** priority **50**;
- **Epicurus/Clock** priority **51**;
- **Confucius/Philosopher_modMain_Instance** priority **50**;
- **Confucius/Clock** priority **51**;

to Protection Doman **PD_1**.

B.8 ECOA XML and Code Generation

Generate XML for elements of the design

- **Restaurant**
- **Table_Imp**
- **Philosopher_Imp**
- **Desktop**
- **Milliways**
- **Demo_1**

Generate API framework code for **Table_Imp** and **Philosopher_Imp**.

> *NOTE: At this point it is possible to go on and insert operating code in the framework – but for this demonstration it is left until later.*

Generate UIDs for **Demo_1**.

Generate Platform code for **Demo_1**.

Insert Implementation Code into the generated code frameworks for **Table_Imp** and **Philosopher_Imp** – see section 5.4.

Appendix C ECOA XML Validation with Eclipse

The Eclipse IDE supports the validation of XML against a schema. Using this feature it is possible to use Eclipse to validate the XML generated by OSETS against the ECOA XML schema.

To add the ECOA schema files to Eclipse (Neon):

1) Download and unpack the ECOA Metamodel schema files onto your computer;

   NOTE: The schema files can be downloaded from <http://www.ecoa.technology/_static/documentation/AS_I05/metamodel-1.13.1.zip>

   NOTE: You need schema 1.13.1 to be compatible with ECOA Architecture Specification v5.

2) In Eclipse create a new generic project called *Model* - **File** | **New** | **Project** | **General** | **Project** | **Next**, set the project name as *Model* and **Finish**;

   NOTE: The names are important to match names used in the ECOA catalog.xml file.

3) Create a folder *Schemas* in *Model* - **Right Click** (*Model*) | **New** | **Folder** | set folder name as *Schemas* and **Finish**;
4) Copy all the ECOA schema files (.xsd) and subfolders (sca, XML, etc) into the *Schemas* folder (drag and drop will do this);
5) Add the schema files to Eclipse's XML Catalog
   a) Select **Window** | **Preferences** | **XML** | **XML Catalog** | **User Specified Entries** | **Add**
   b) Select **Next Catalog** | **Workspace** | *Model* | *Schemas* | *catalog.xml* | **OK** | **OK** | **OK**

ECOA XML files will now be validated against the ECOA schemas.