MENU

All Tutorials

Java

Maven

Gradle

Servlet/Jsp

Thymeleaf

Spring

Struts2

Hibernate

Java Web Service

JavaFX

SWT

Oracle ADF

Android

iOS

Python

Swift

C#

C/C++

Ruby

Dart

Batch

Database

Oracle APEX

Report

Client

ECMAScript / Javascript

TypeScript

NodeJS

ReactJS

Flutter

AngularJS

HTML

CSS

Bootstrap

OS

Git

SAP

Amazon AWS

Others

Eclipse RCP 4 Tutorial for Beginners - e4 Workbench Application

View more Tutorials:

Eclipse Technology

Eclipse RCP

14
Shares

1- The installation requires

Install *WindowBuilder* plugin, there is a plugin that allows you to design SWT GUI applications using drag and drop convenience.
See installation instructions at:

- Install WindowBuilder for Eclipse

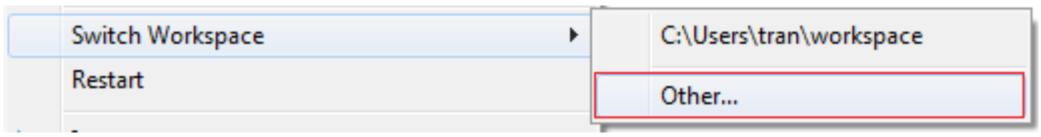
Installing e4 Tools Developer Sources:

- Install e4 Tools Developer Resources for Eclipse

2- Create new Java Workspace

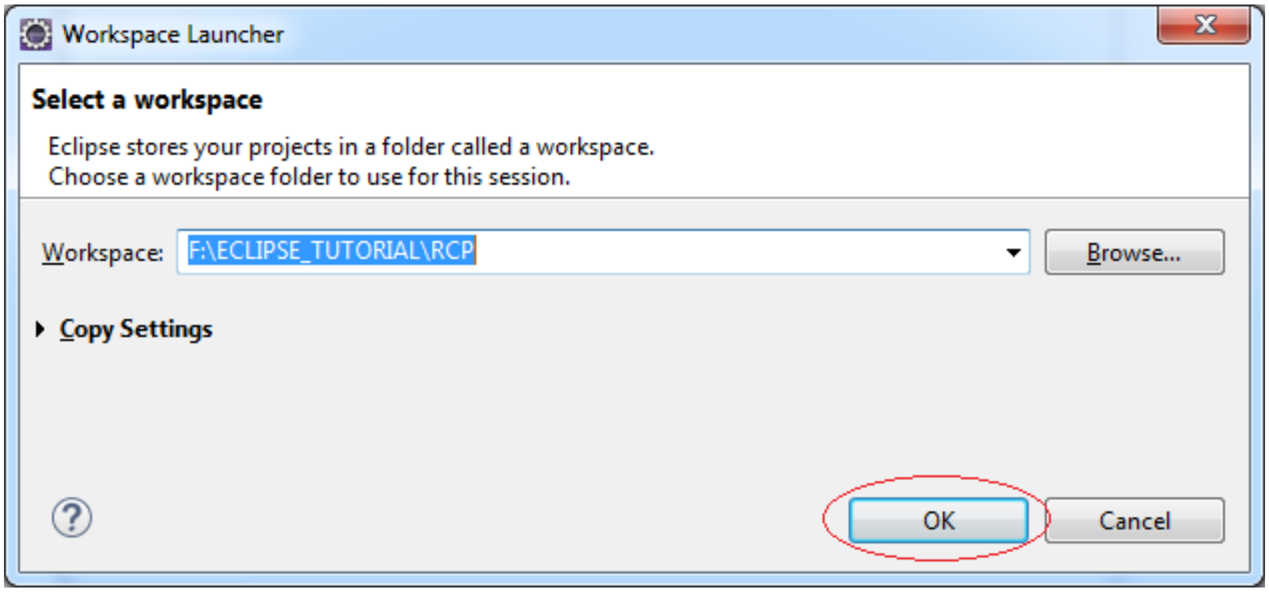
To programme RCP application you should create a new **Java Workspace**:

- File/Switch Workspace/Other..



Enter:

- F:\ECLIPSE_TUTORIAL\RCP



Select a workspace

Eclipse stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

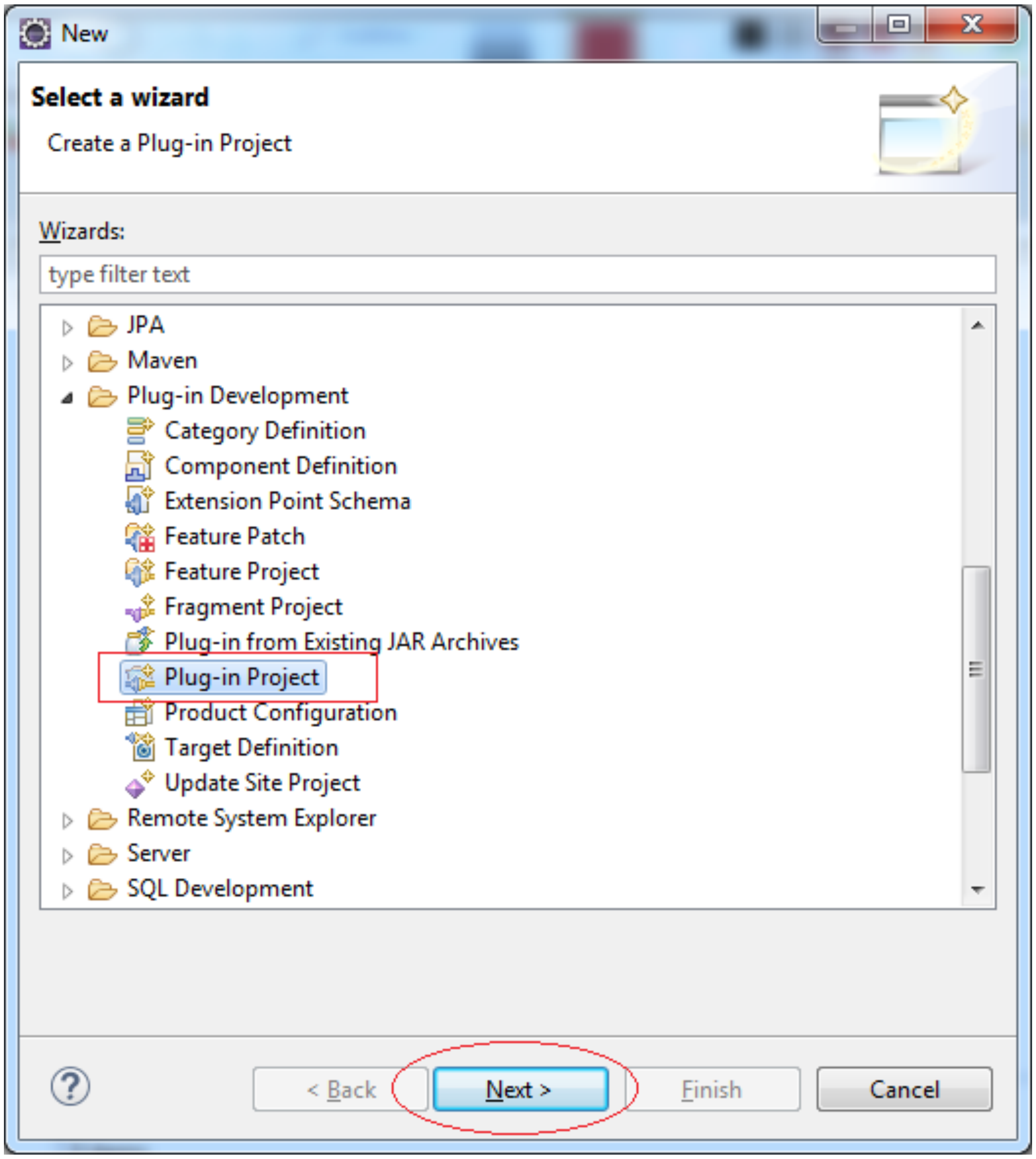
Workspace: F:\ECLIPSE_TUTORIAL\RCP

3- Create RCP Project

In this document, I will guide you to build an RCP application from the beginning (Do not follow the available template), so I'll create a blank Eclipse RCP application.

In Eclipse select:

- **File/New/Other...**


New

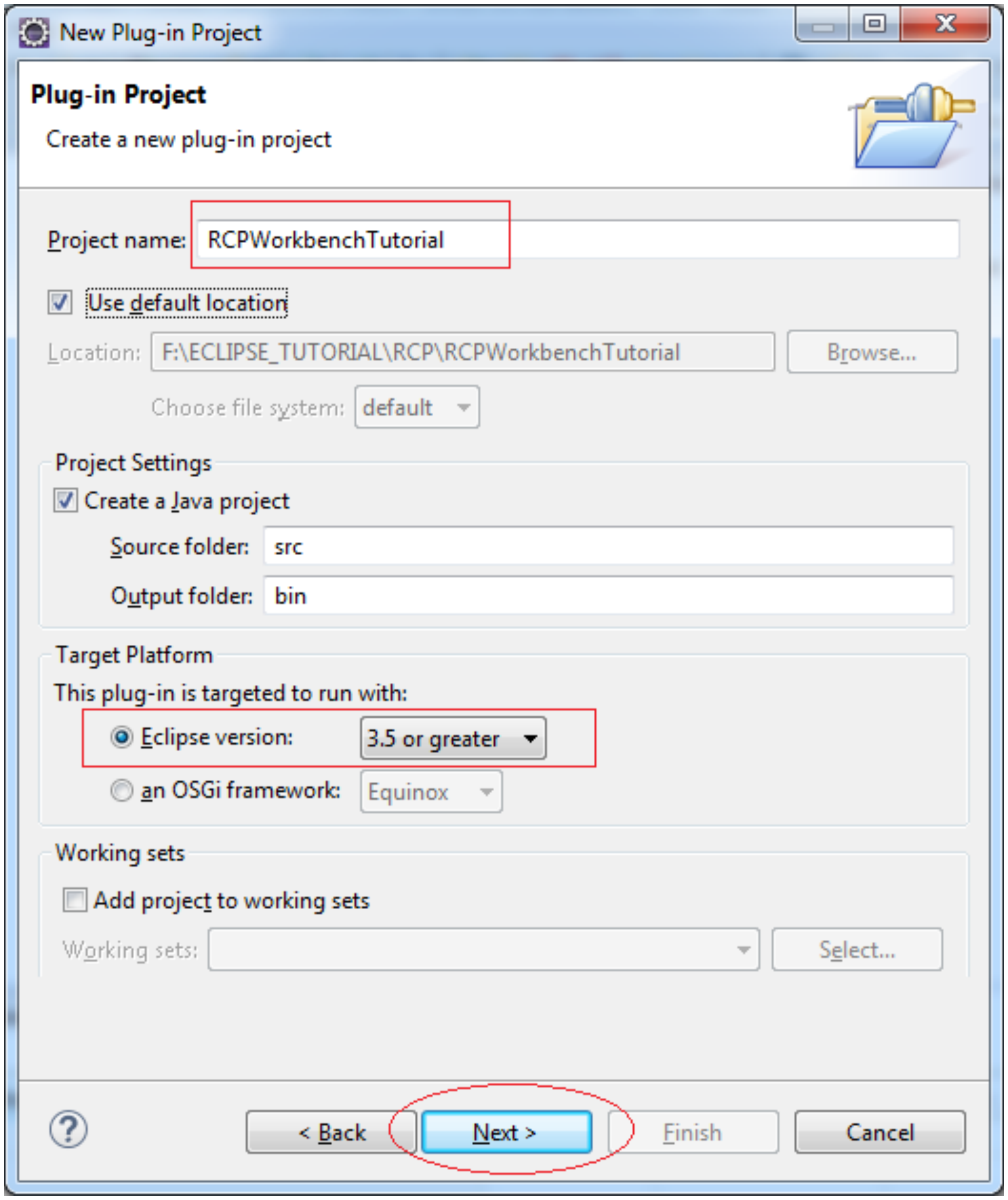
Select a wizard

Create a Plug-in Project

Wizards:

type filter text

- JPA
- Maven
- Plug-in Development
  - Category Definition
  - Component Definition
  - Extension Point Schema
  - Feature Patch
  - Feature Project
  - Fragment Project
  - Plug-in from Existing JAR Archives
  - Plug-in Project
  - Product Configuration
  - Target Definition
  - Update Site Project
- Remote System Explorer
- Server
- SQL Development



Project name: RCPWorkbenchTutorial

☑ Use default location

Location: F:\ECLIPSE_TUTORIAL\RCP\RCPWorkbenchTutorial

Choose file system: default

Project Settings

☑ Create a Java project

Source folder: src

Output folder: bin

Target Platform

This plug-in is targeted to run with:

● Eclipse version: 3.5 or greater

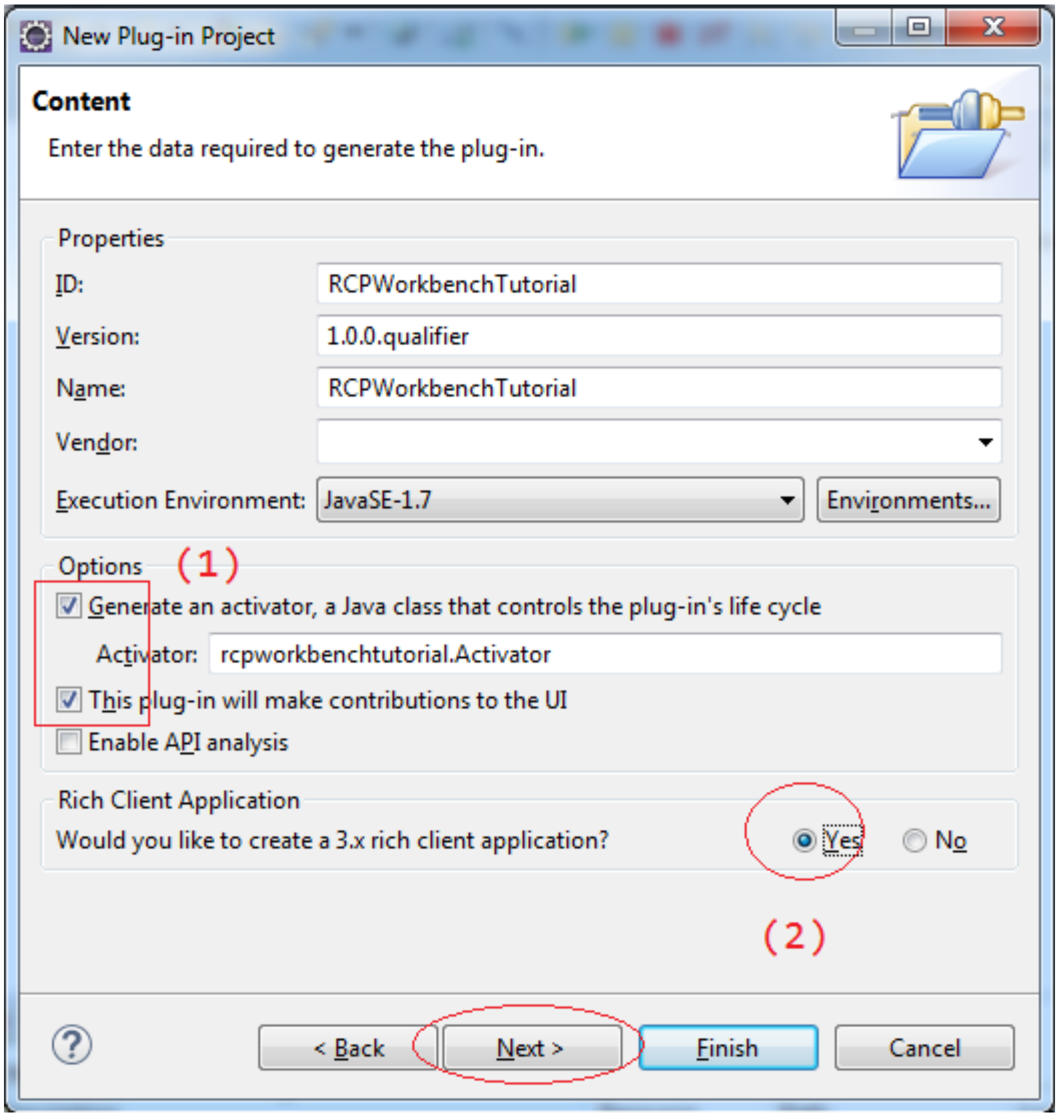
○ an OSGi framework: Equinox

Working sets

☐ Add project to working sets

Working sets:

1. Check selected on (1)
2. On the (2) select "***Yes***" to create Eclipse RCP Project (Running on Desktop), otherwise it will create RAP Project (Running on the Web).



New Plug-in Project

Content

Enter the data required to generate the plug-in.

Properties

ID:	RCPWorkbenchTutorial
Version:	1.0.0.qualifier
Name:	RCPWorkbenchTutorial
Vendor:	
Execution Environment:	JavaSE-1.7

Options (1)

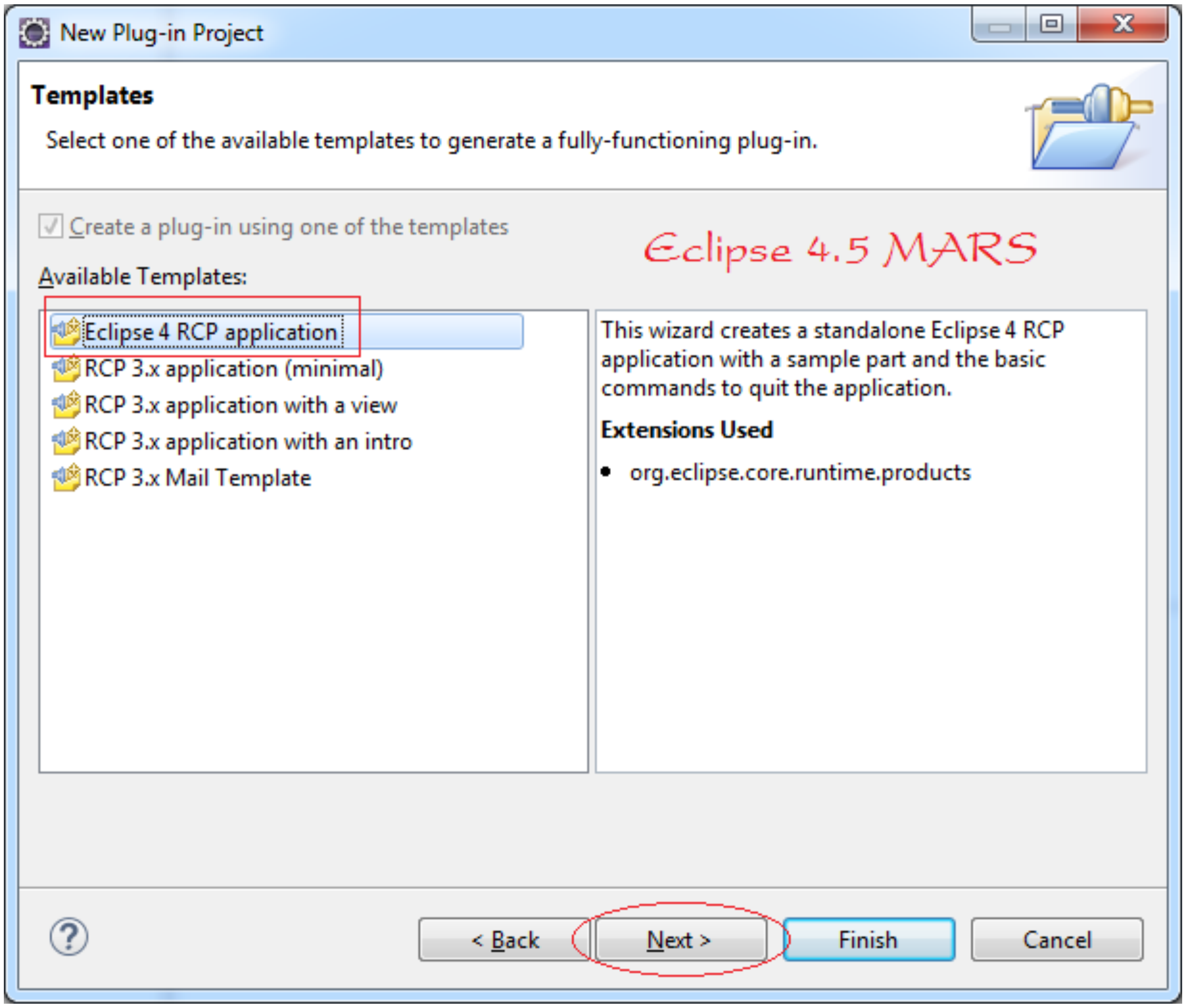
- [✓] Generate an activator, a Java class that controls the plug-in's life cycle
  - Activator: rcpworkbenchtutorial.Activator
- [✓] This plug-in will make contributions to the UI
- [ ] Enable API analysis

Rich Client Application

Would you like to create a 3.x rich client application? [✓ Yes] No

(2)

Select "*Eclipse 4 RCP Application*"


New Plug-in Project

Templates

Select one of the available templates to generate a fully-functioning plug-in.

☑ Create a plug-in using one of the templates

Available Templates:

- Eclipse 4 RCP application
- RCP 3.x application (minimal)
- RCP 3.x application with a view
- RCP 3.x application with an intro
- RCP 3.x Mail Template

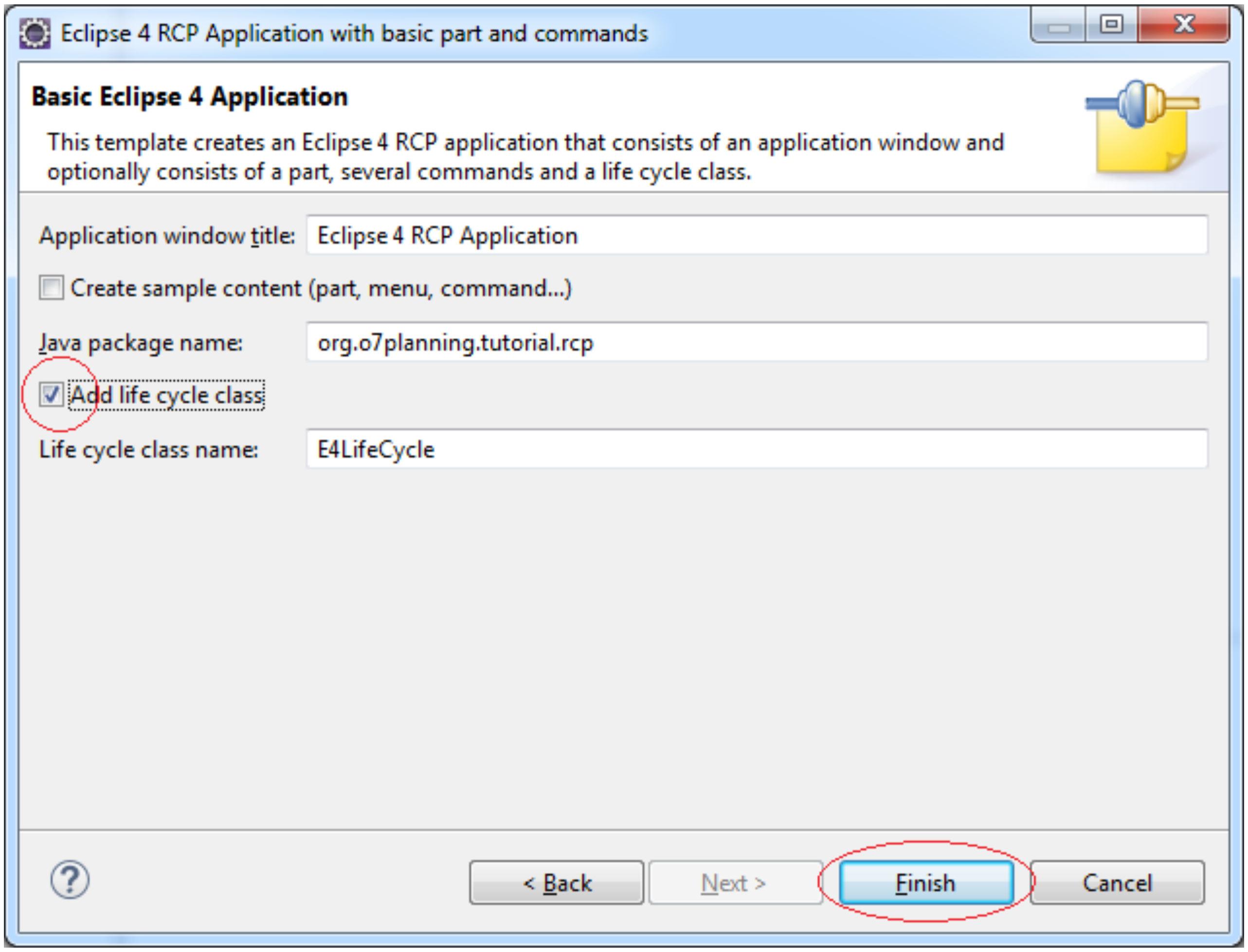
This wizard creates a standalone Eclipse 4 RCP application with a sample part and the basic commands to quit the application.

Extensions Used

- org.eclipse.core.runtime.products

Enter:

- **Java package name**: org.o7planning.tutorial.rcp



Basic Eclipse 4 Application

This template creates an Eclipse 4 RCP application that consists of an application window and optionally consists of a part, several commands and a life cycle class.

Application window title: Eclipse 4 RCP Application

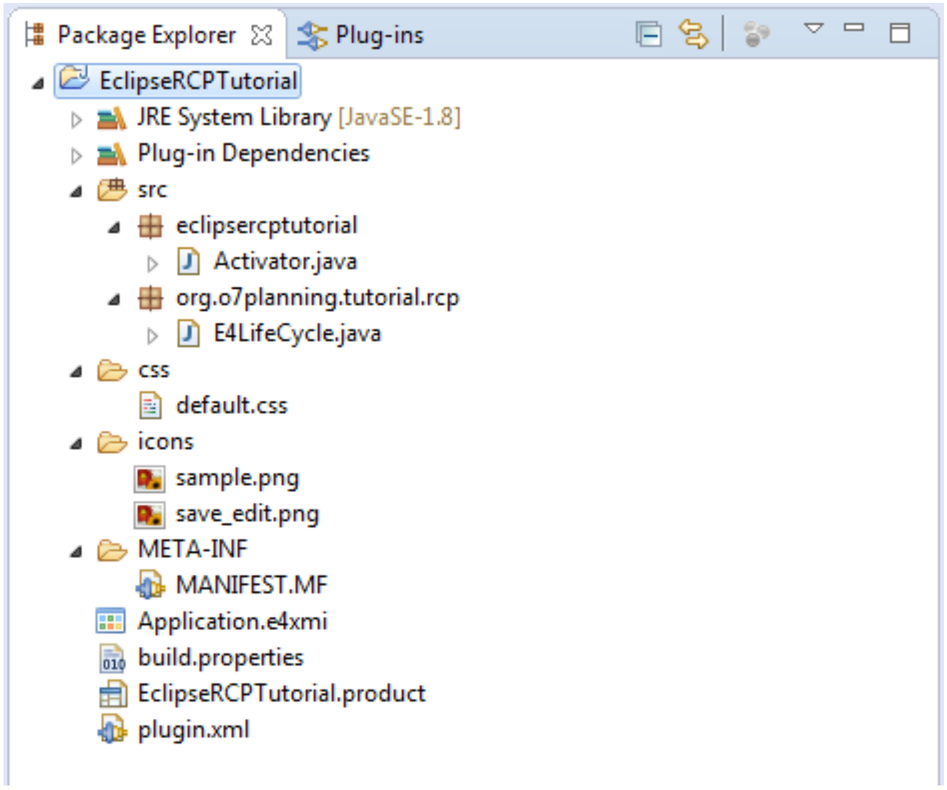
[ ] Create sample content (part, menu, command...)

Java package name: org.o7planning.tutorial.rcp

[✓] Add life cycle class

Life cycle class name: E4LifeCycle

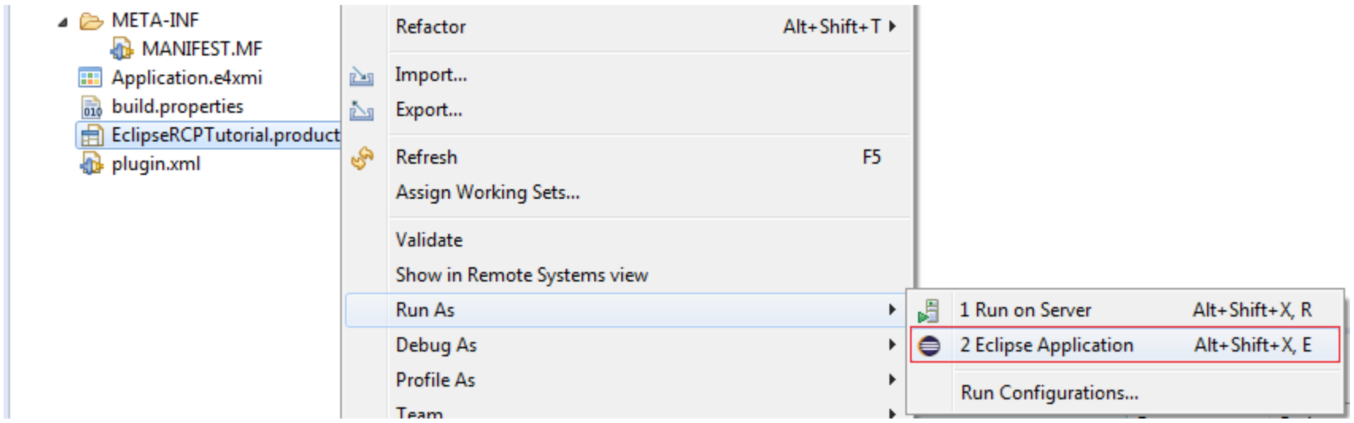
Project was created:

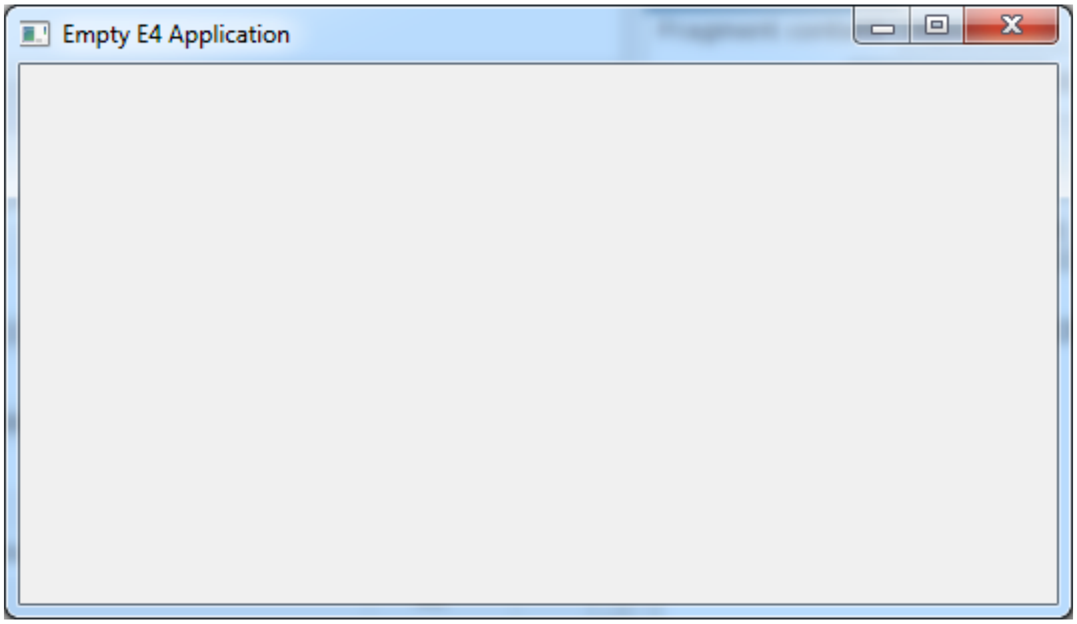


```
EclipseRCPTutorial
  JRE System Library [JavaSE-1.8]
  Plug-in Dependencies
  src
    eclipsercptutorial
      Activator.java
    org.o7planning.tutorial.rcp
      E4LifeCycle.java
  css
    default.css
  icons
    sample.png
    save_edit.png
  META-INF
    MANIFEST.MF
  Application.e4xmi
  build.properties
  EclipseRCPTutorial.product
  plugin.xml
```

Run application:

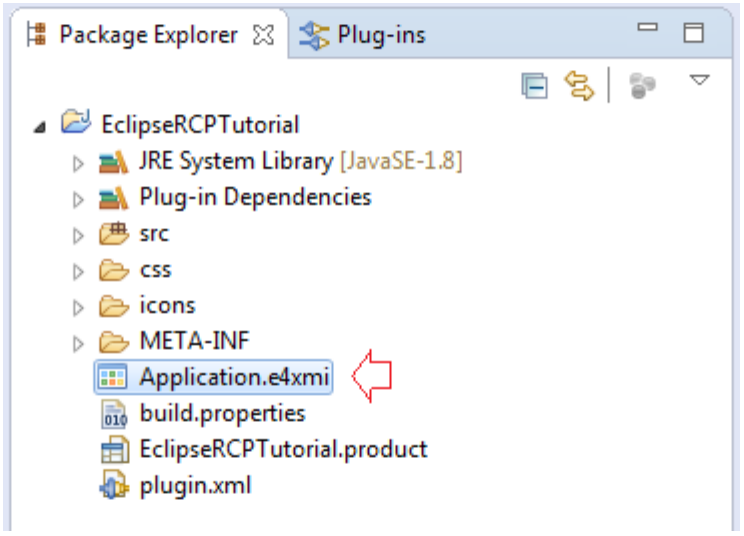
Right click *EclipseRCPTutorial.product* and select Run As/Eclipse Application

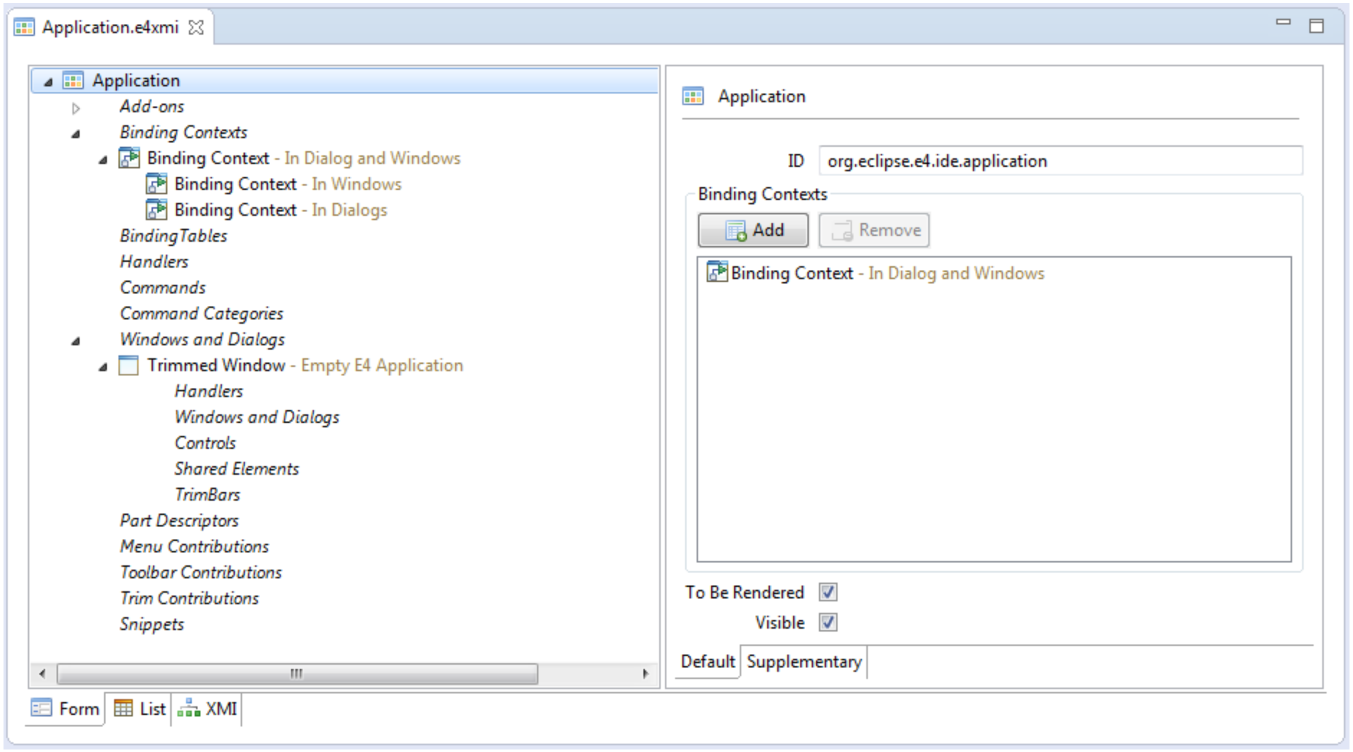




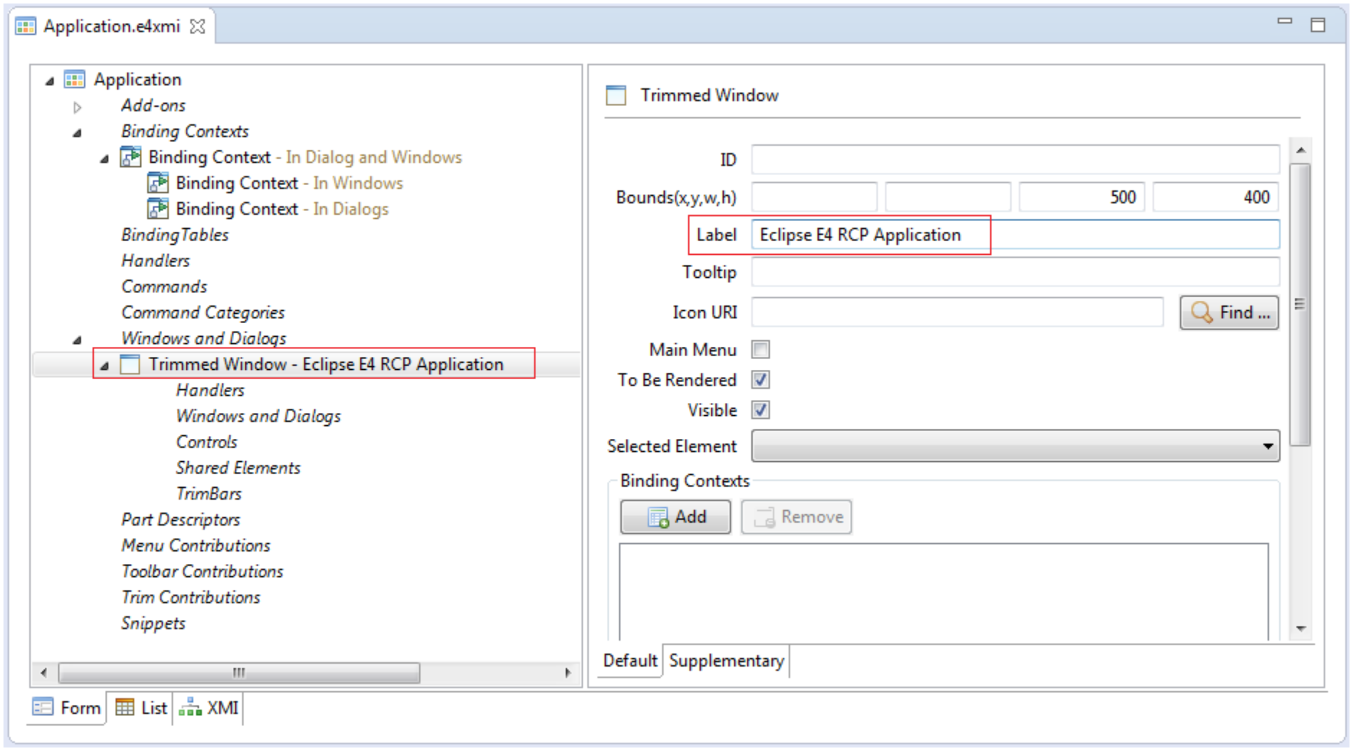
4- Structure of Empty Eclipse 4 RCP application

Empty **Eclipse 4 RCP** applications was created by **Eclipse**, see its structure. Open **Application.e4xmi** file:



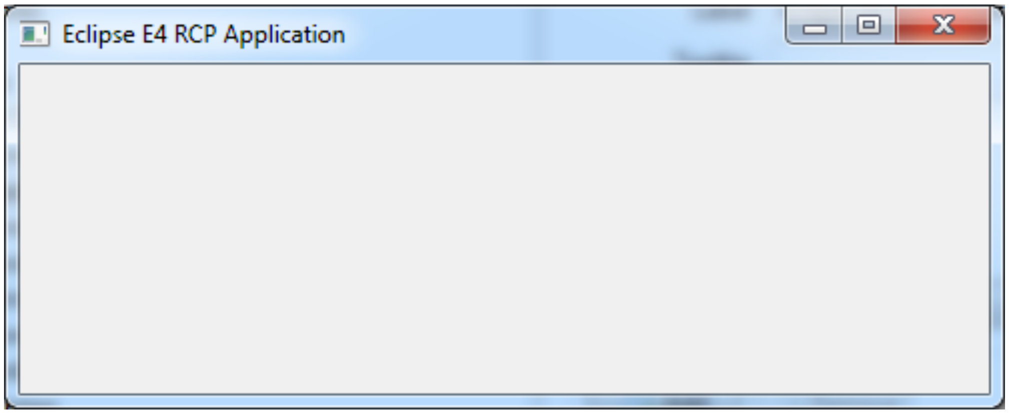


Change the title of the application into **"Eclipse E4 RCP Application"** as shown below.



ID				
Bounds(x,y,w,h)			500	400
Label	Eclipse E4 RCP Application			
Tooltip				
Icon URI				
Main Menu	☐			
To Be Rendered	✓			
Visible	✓			
Selected Element				

And rerun application:



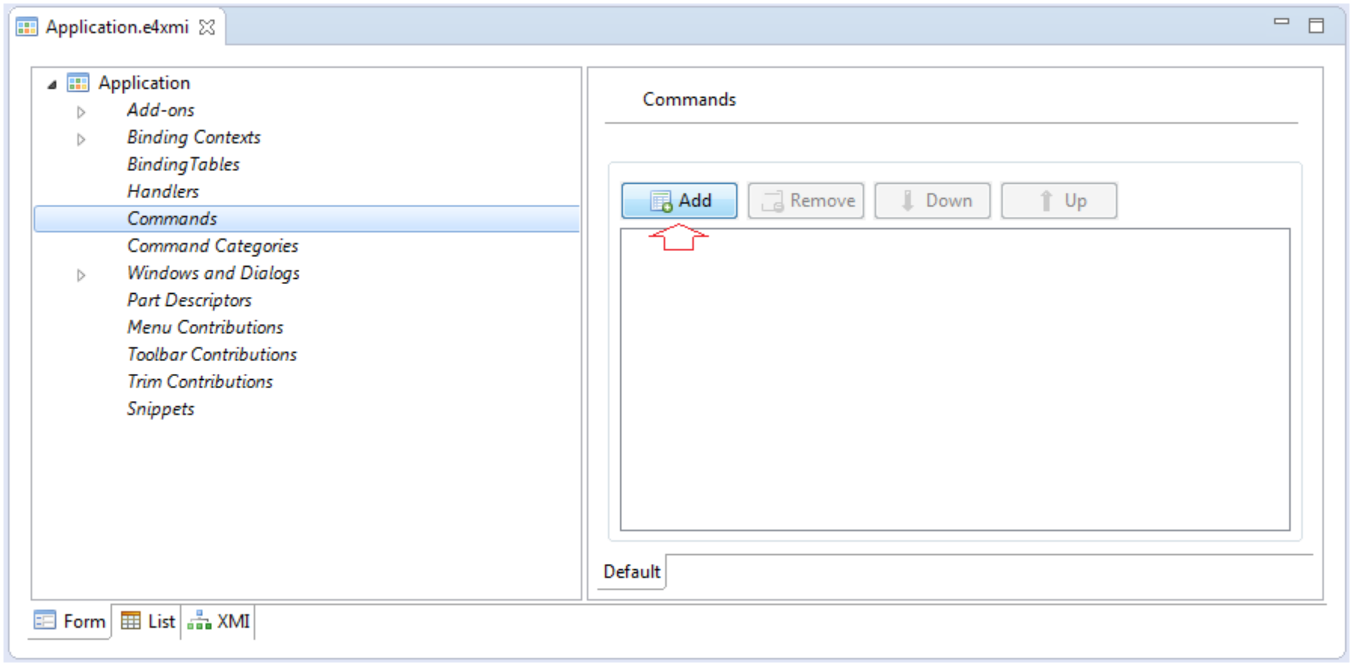
5- Handler and Command

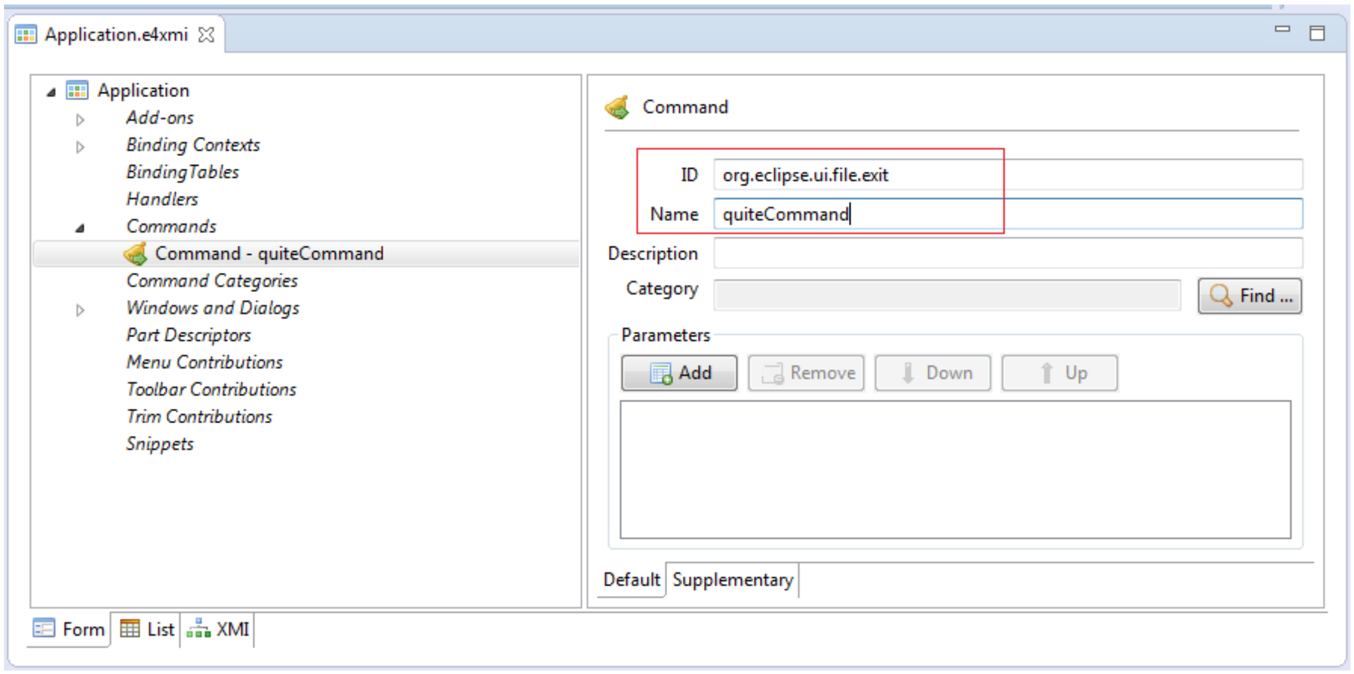
RCP Framework built a lot of Command, for example 4 commands with the following ID:

1. org.eclipse.ui.file.exit
    - Close the workbench
2. org.eclipse.ui.file.open
    - Open file
3. org.eclipse.ui.file.save
    - Save the active editor.
4. org.eclipse.ui.help.aboutAction
    - Displays the About dialog.

> **"**
>
> *You can see a list of built-in command of the RCP framework at:*
>
> - *http://help.eclipse.org/mars/topic/org.eclipse.platform.doc.isv/reference/api/org/eclipse/ui/actions/ActionFactory.html*

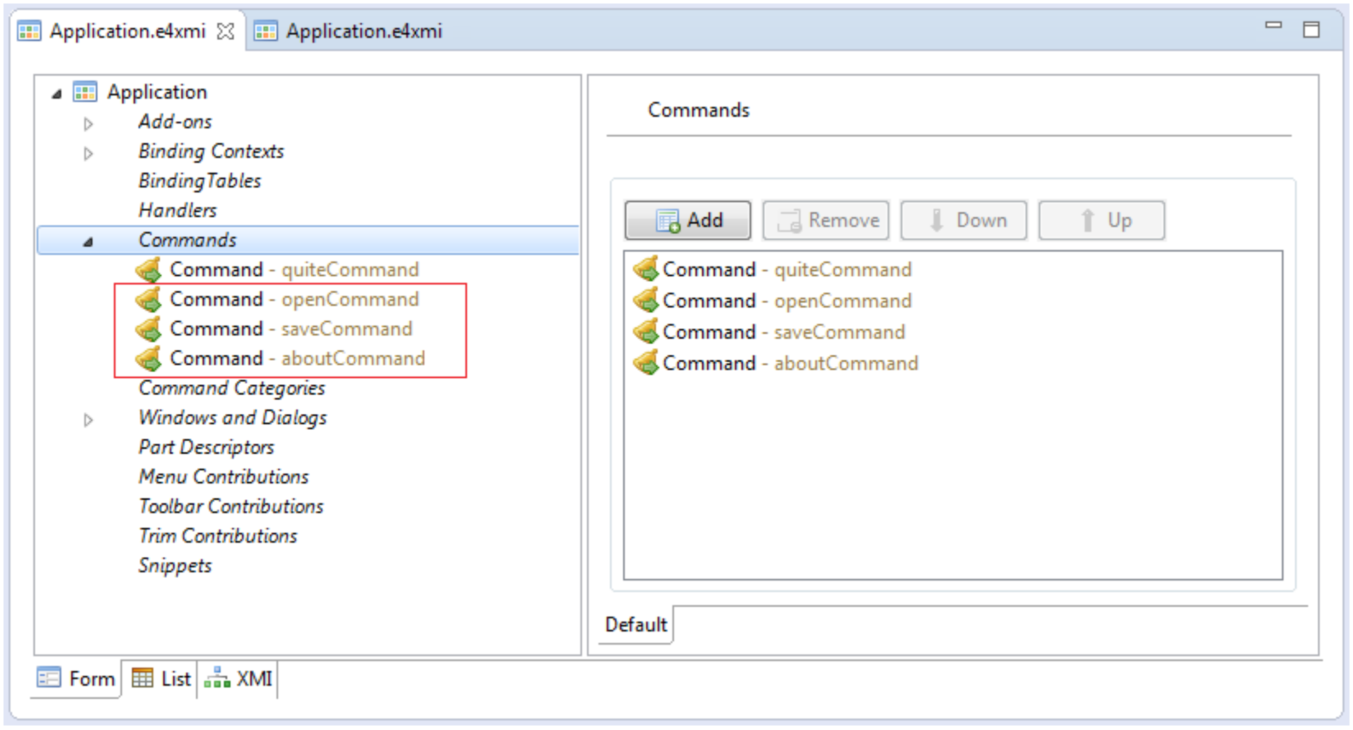
Create a Command named ***quiteCommand*** , called to exit the application command which was built by *RCP Framework*.





ID	org.eclipse.ui.file.exit
Name	quiteCommand

Similarly we create 3 other Commands:



Open selected file.

- **ID**: org.eclipse.ui.file.open
- **Name:** openCommand

Save the active editor (or part).

- **ID:** org.eclipse.ui.file.save
- **Name:** saveCommand

AboutCommand:

- **ID:** org.eclipse.ui.help.aboutAction
- **Name:** aboutCommand

Handler is the classes handling for the commands of Menu or Toolbar. When you click on MenuItem or ToolItem, it means that the call to execute a command,Handler will be executed before the Command is executed, you can cancel (cancel) the command is executed in Handler.

Handler is the classes that when executed, it will execute the task is written in method annotated by *@Execute*.

I will create 4 Handler classes:

AboutHandler.java

```
package org.o7planning.tutorial.rcp.handler;

import org.eclipse.e4.core.di.annotations.Execute;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.swt.widgets.Shell;

public class AboutHandler {

  @Execute
  public void execute(Shell shell) {
     MessageDialog.openInformation(shell, "About", "Eclipse 4 RCP Application");
  }
}
```

OpenHandler.java

```
package org.o7planning.tutorial.rcp.handler;

import org.eclipse.e4.core.di.annotations.Execute;
import org.eclipse.swt.widgets.FileDialog;
import org.eclipse.swt.widgets.Shell;

public class OpenHandler {

  @Execute
  public void execute(Shell shell){
     FileDialog dialog = new FileDialog(shell);
     dialog.open();
  }
}
```

QuitHandler.java

```
package org.o7planning.tutorial.rcp.handler;

import org.eclipse.e4.core.di.annotations.Execute;
import org.eclipse.e4.ui.workbench.IWorkbench;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.swt.widgets.Shell;
```

```
public class QuitHandler {
  @Execute
  public void execute(IWorkbench workbench, Shell shell){
     if (MessageDialog.openConfirm(shell, "Confirmation",
           "Do you want to exit?")) {
        workbench.close();
     }
  }
}
```

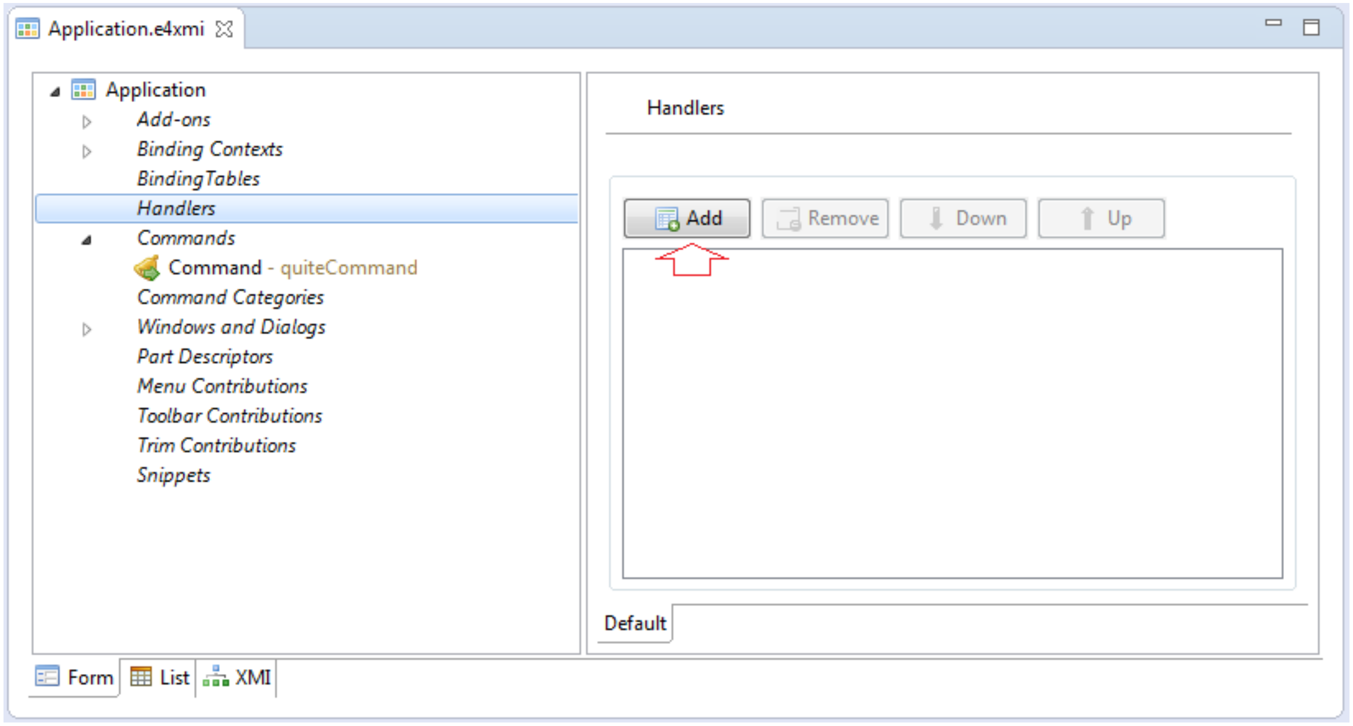
SaveHandler.java

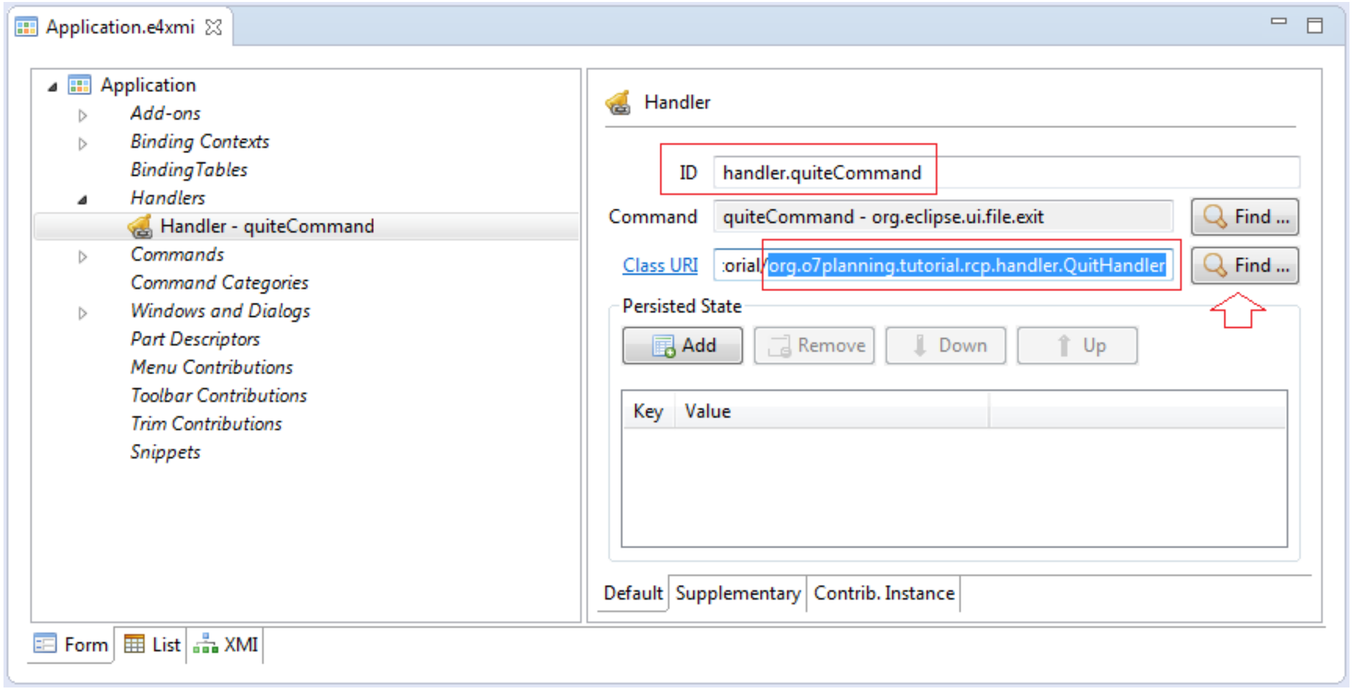
```
package org.o7planning.tutorial.rcp.handler;

import org.eclipse.e4.core.di.annotations.CanExecute;
import org.eclipse.e4.core.di.annotations.Execute;
import org.eclipse.e4.ui.workbench.modeling.EPartService;

public class SaveHandler {

  @CanExecute
  public boolean canExecute(EPartService partService) {
     if (partService != null) {
        return !partService.getDirtyParts().isEmpty();
     }
     return false;
  }

  @Execute
  public void execute(EPartService partService) {
     partService.saveAll(false);
  }
}
```

Declaring Handler classes with applications.

Declaring a Handler with Application, you must declare the ID, Handler class and Command.

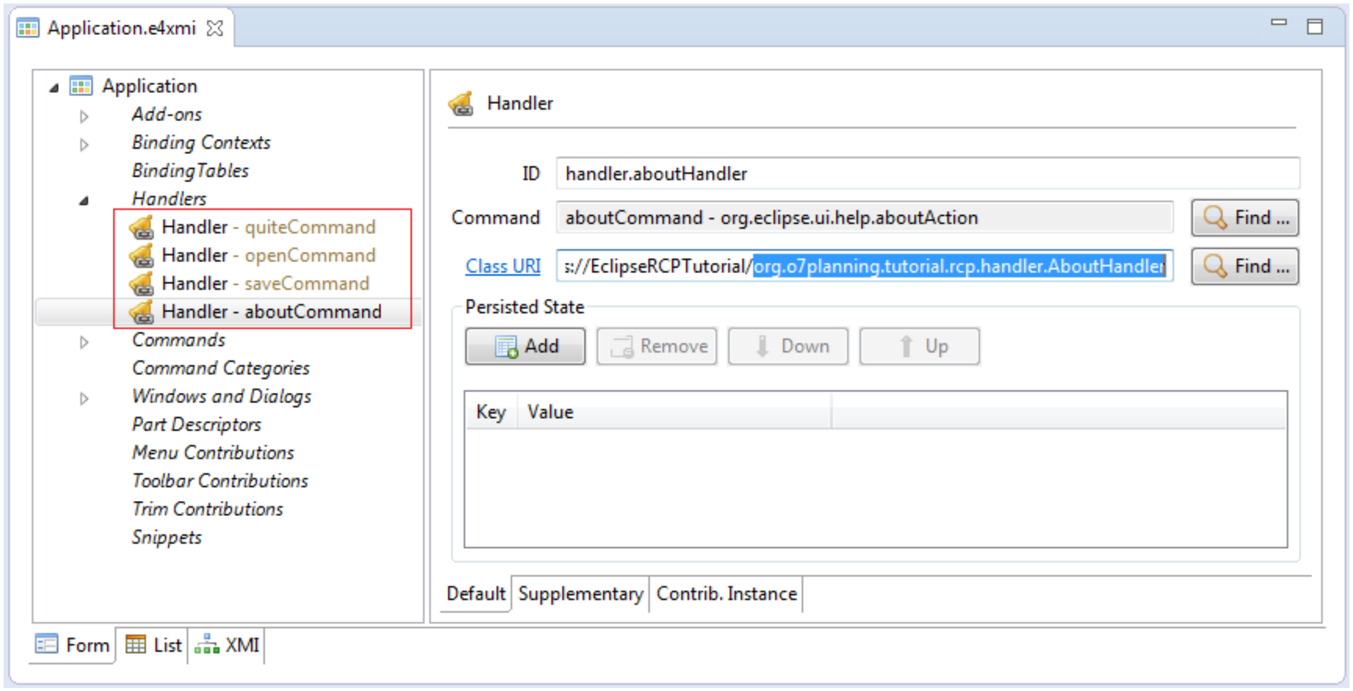
For example, the user closes the application by clicking the menuItem Exit, the MenuItem associated with ***quiteComman**d*, QuiteHandler is class handling for this command (as declared above), class ***QuiteHandler** will* ask users really want to close the application or not, if yes, q***uiteCommand*** Command will be executed.





ID: handler.quiteCommand
Command: quiteCommand - org.eclipse.ui.file.exit
Class URI: :orial/org.o7planning.tutorial.rcp.handler.QuitHandler

Similarly declare other Handler.



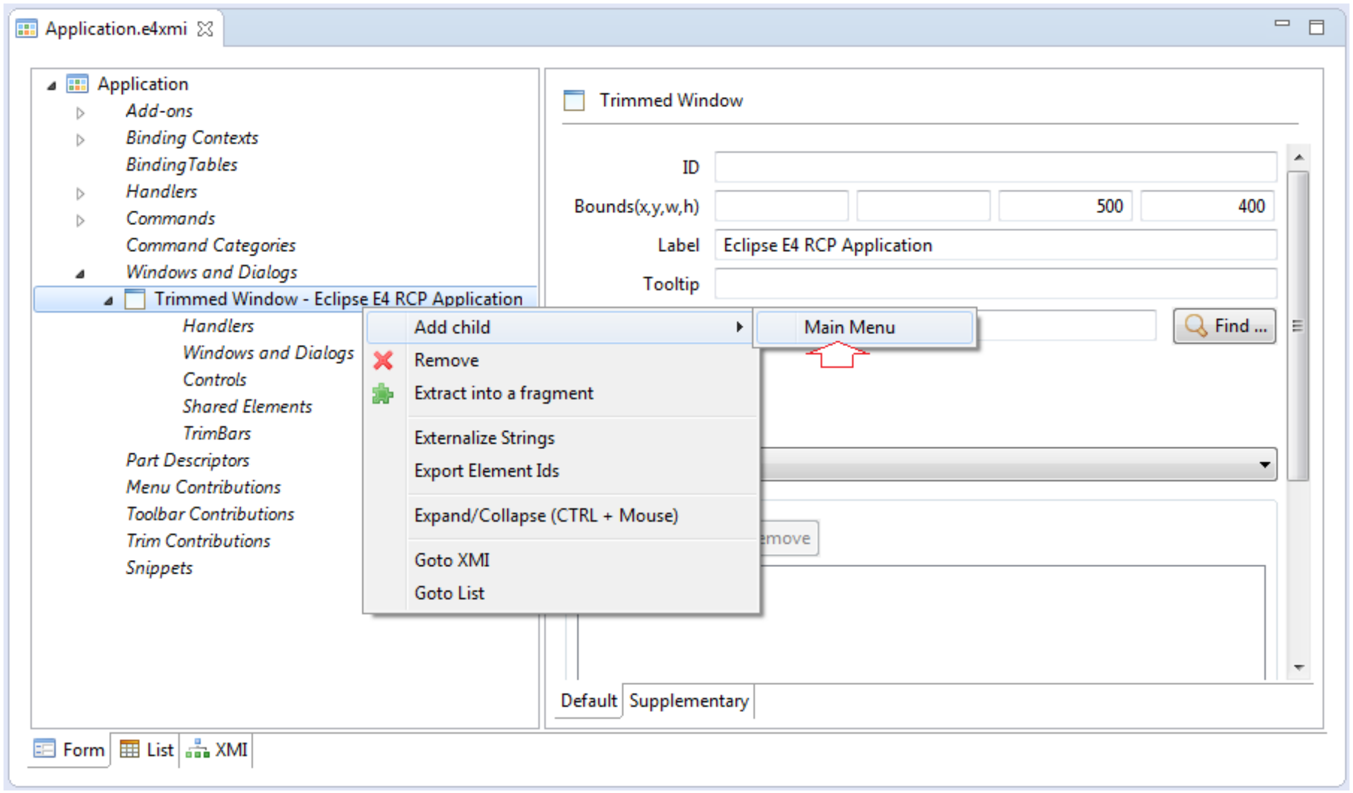
ID: handler.aboutHandler

Command: aboutCommand - org.eclipse.ui.help.aboutAction

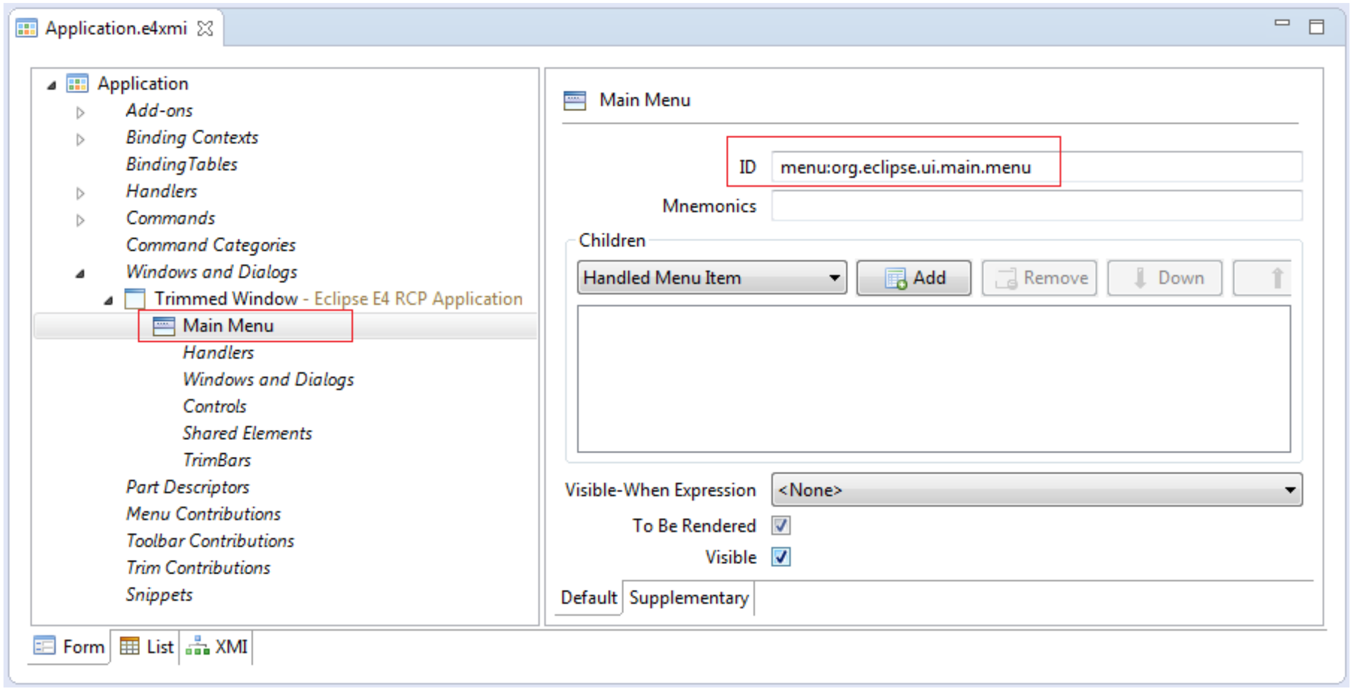
Class URI: :://EclipseRCPTutorial/org.o7planning.tutorial.rcp.handler.AboutHandler

6- Create Menus

Create Main Menu:



Enter the ID of Main Menu, It must be named: ***menu:org.eclipse.ui.main.menu***



ID: menu:org.eclipse.ui.main.menu

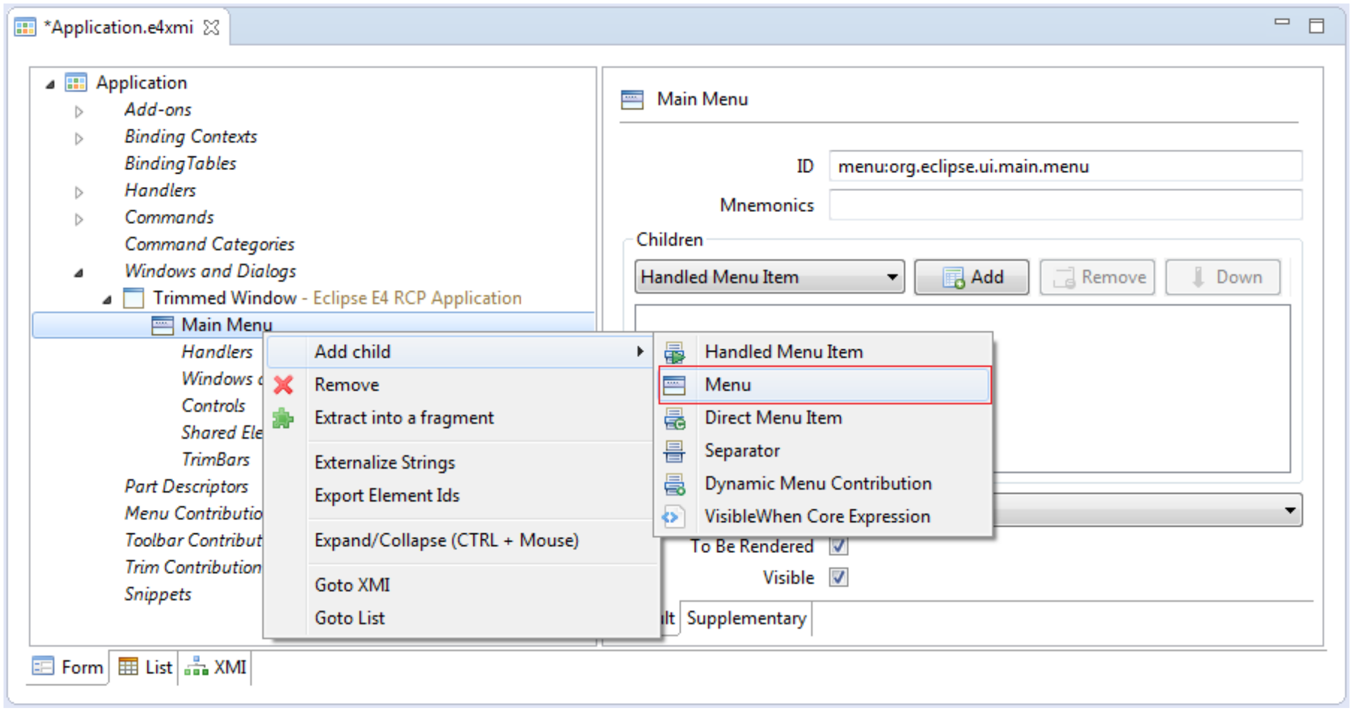
Visible-When Expression: <None>

To Be Rendered: ✓

Visible: ✓

Create 3 submenus:

- *File, Function, Help*



Main Menu

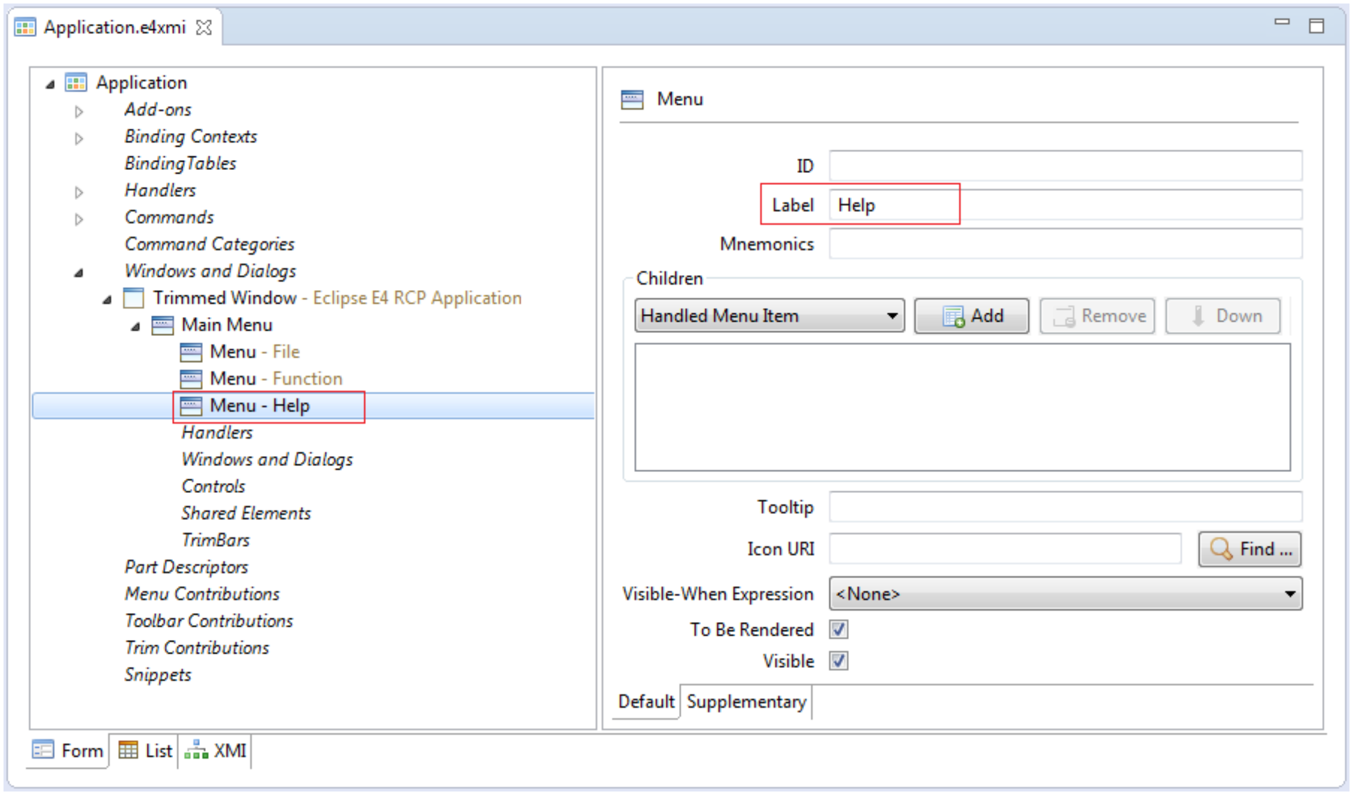
ID: menu:org.eclipse.ui.main.menu

Mnemonics:

Children: Handled Menu Item | Add | Remove | Down

To Be Rendered ✓

Visible ✓



Menu

ID:

Label: Help

Mnemonics:

Children: Handled Menu Item | Add | Remove | Down

Tooltip:

Icon URI: | Find ...

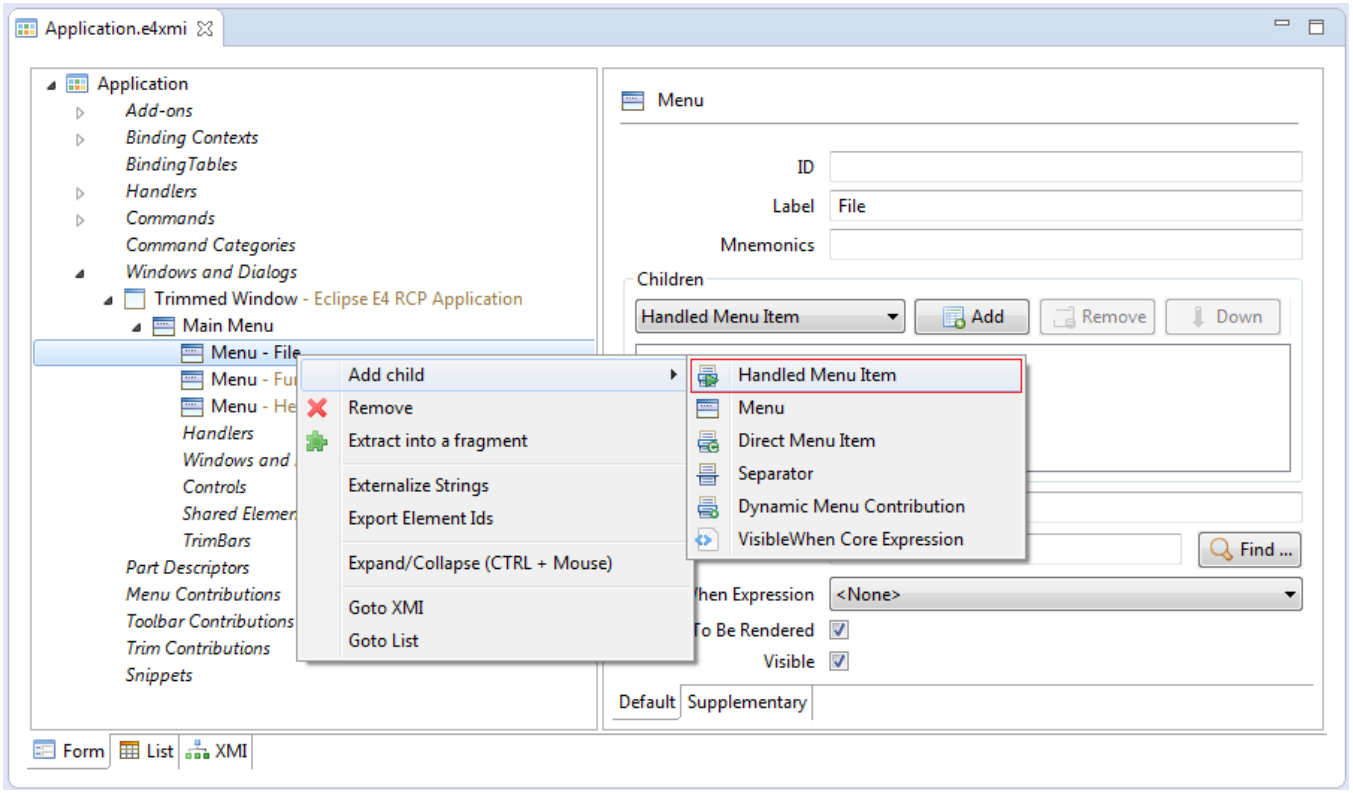
Visible-When Expression: <None>

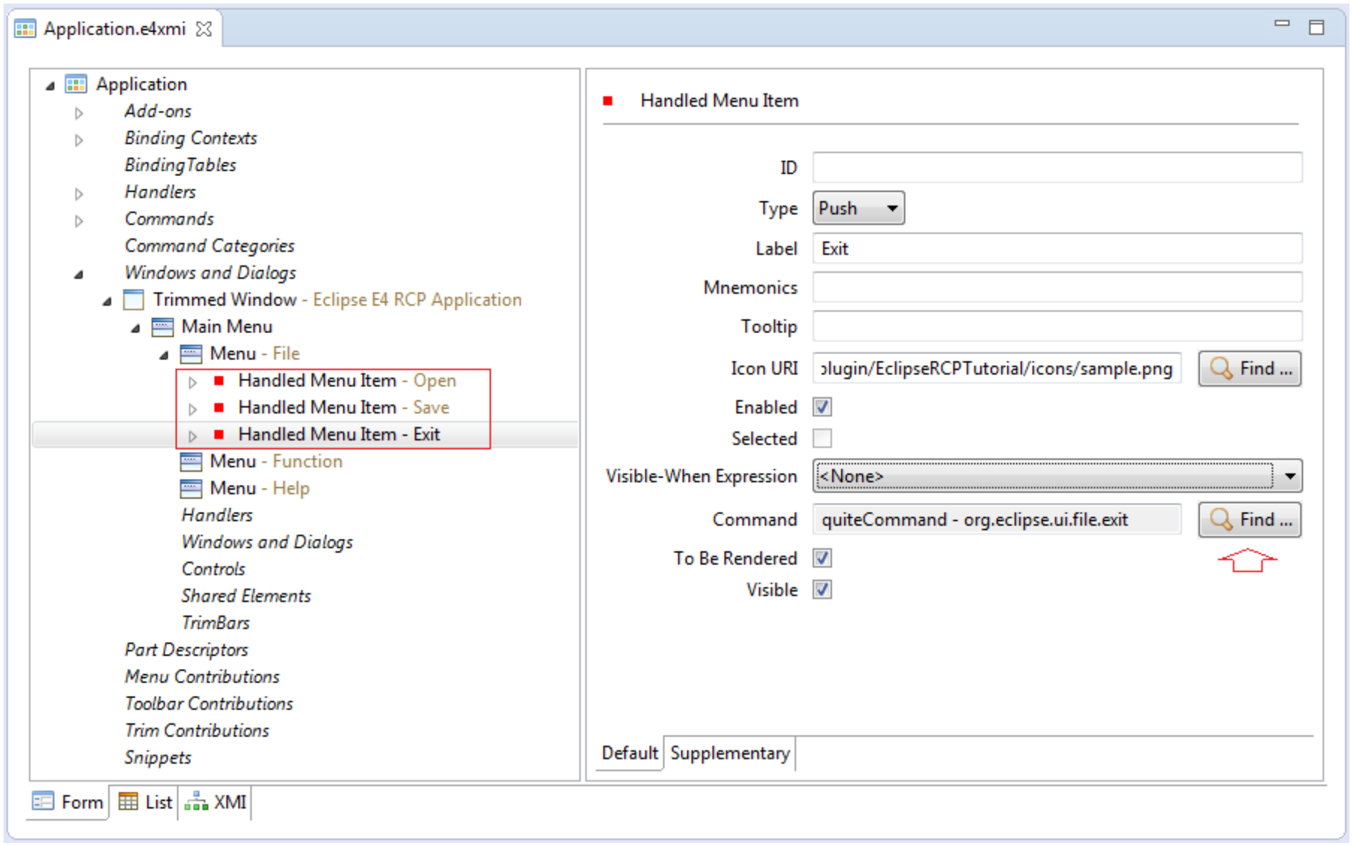
To Be Rendered ✓

Visible ✓

Create submenus of **File** menu.

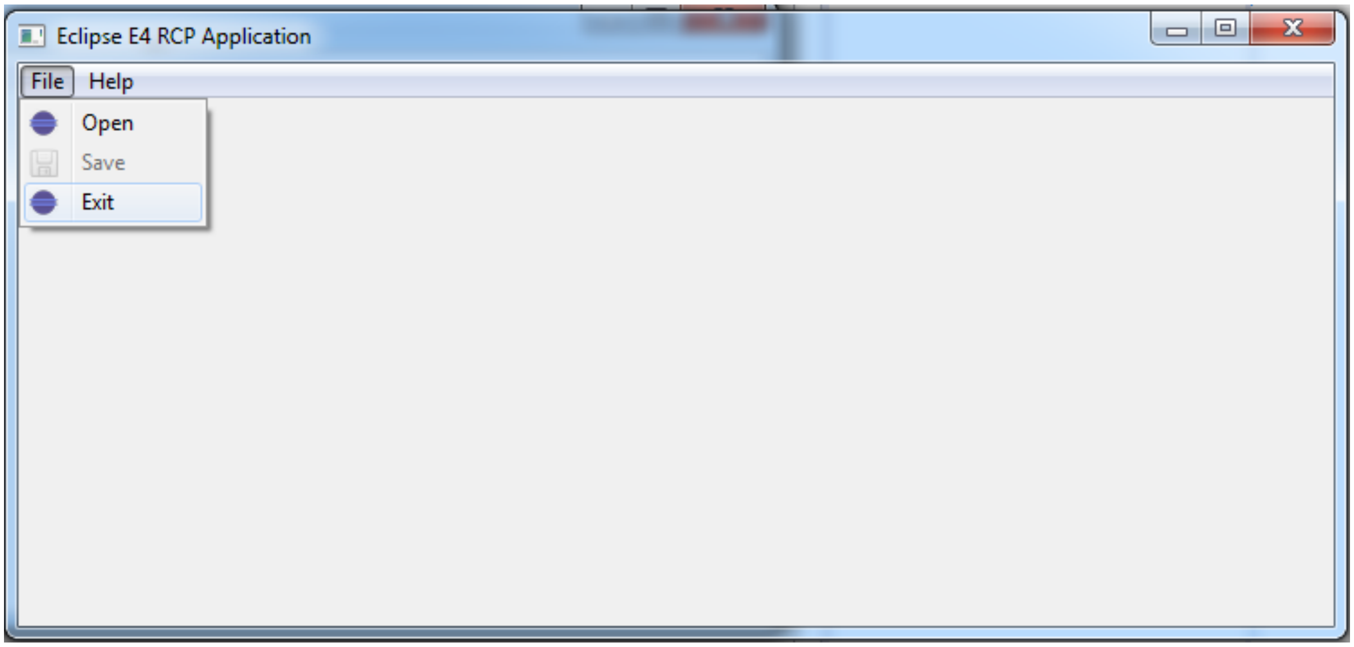
- Open
- Save
- Exit

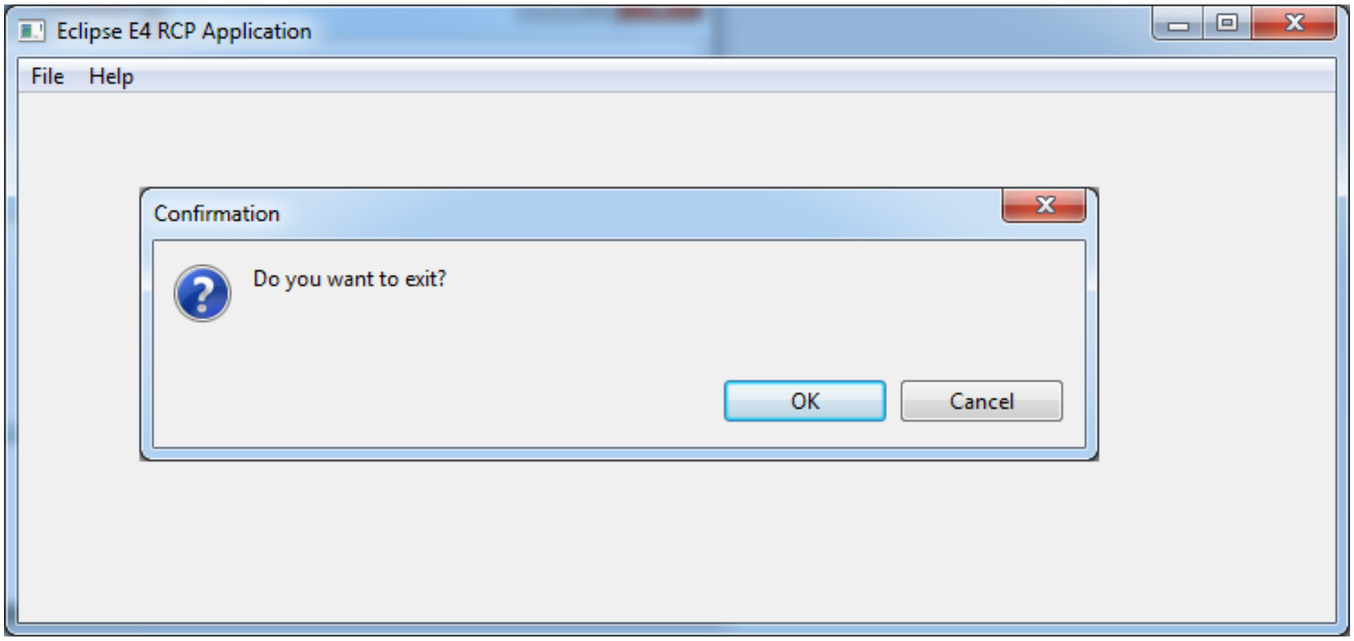




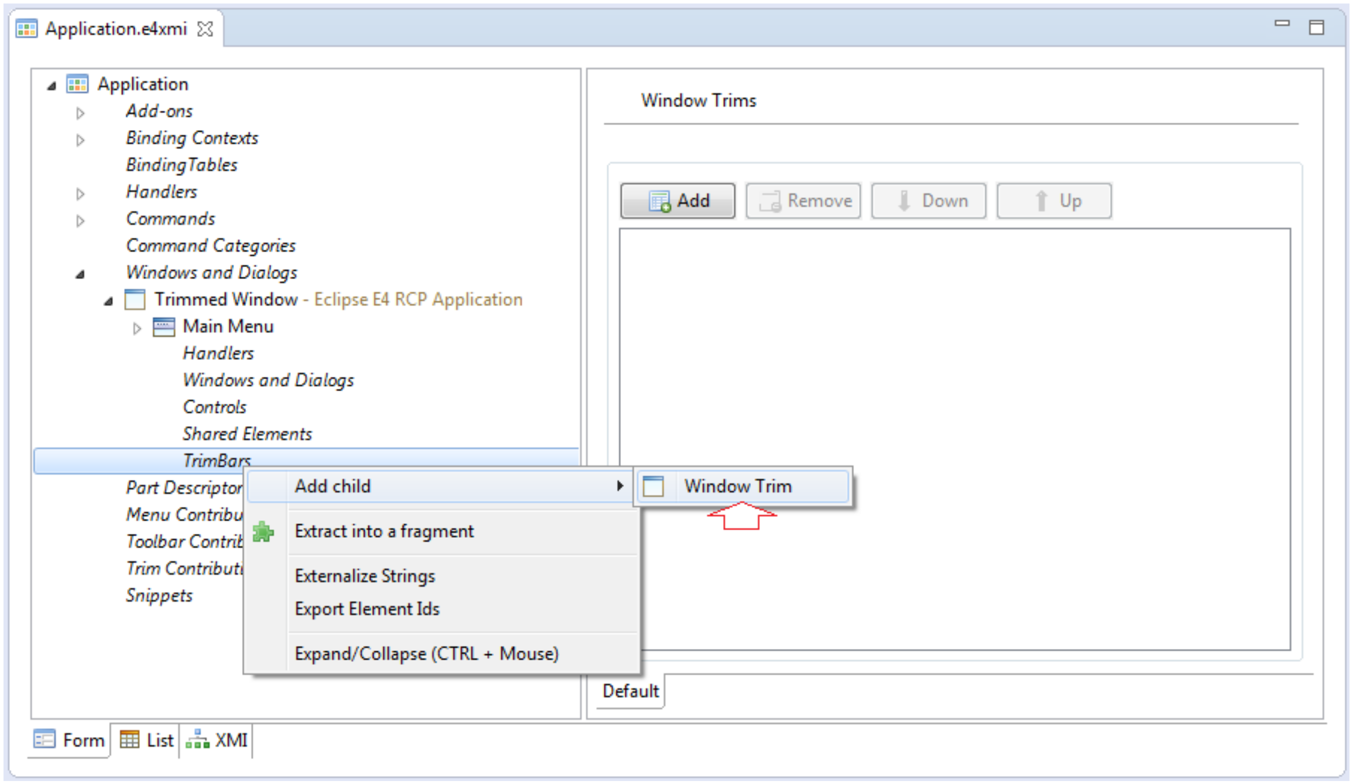
ID	
Type	Push
Label	Exit
Mnemonics	
Tooltip	
Icon URI	plugin/EclipseRCPTutorial/icons/sample.png
Enabled	✓
Selected	
Visible-When Expression	<None>
Command	quiteCommand - org.eclipse.ui.file.exit
To Be Rendered	✓
Visible	✓

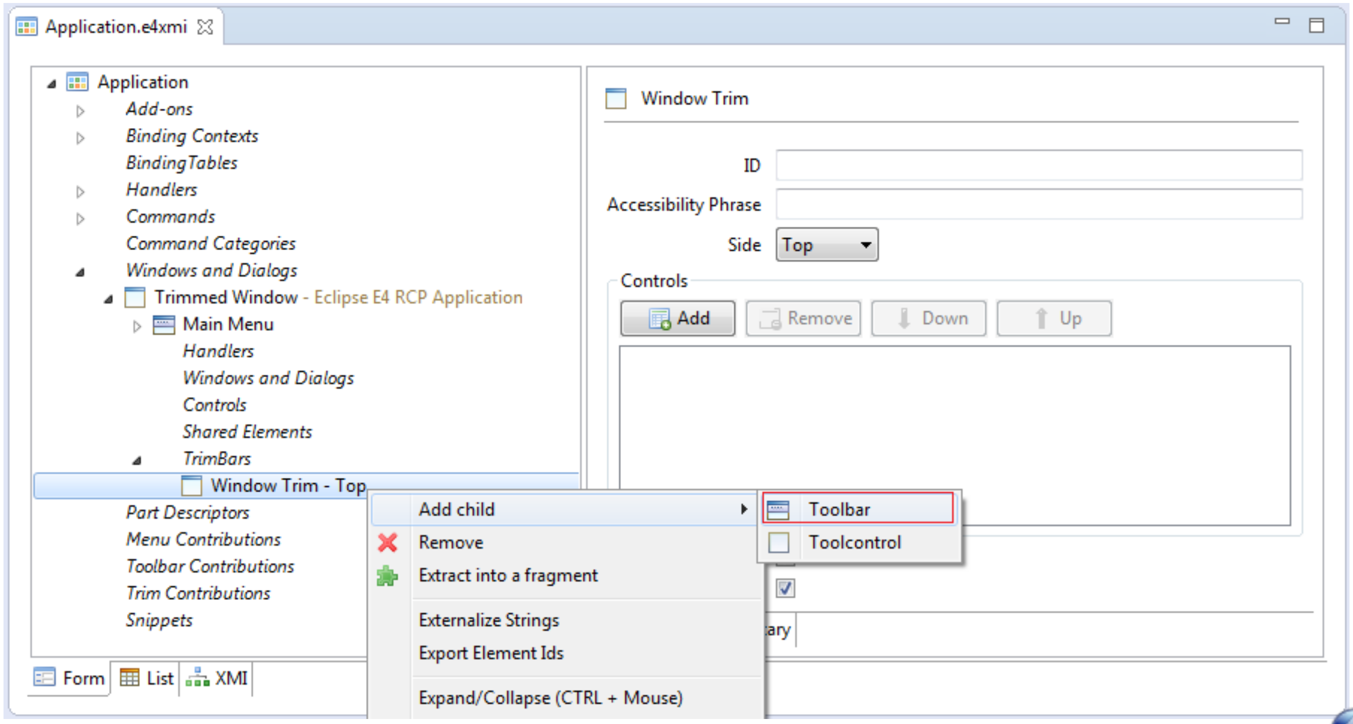
Rerun your application:





7- Create Toolbar

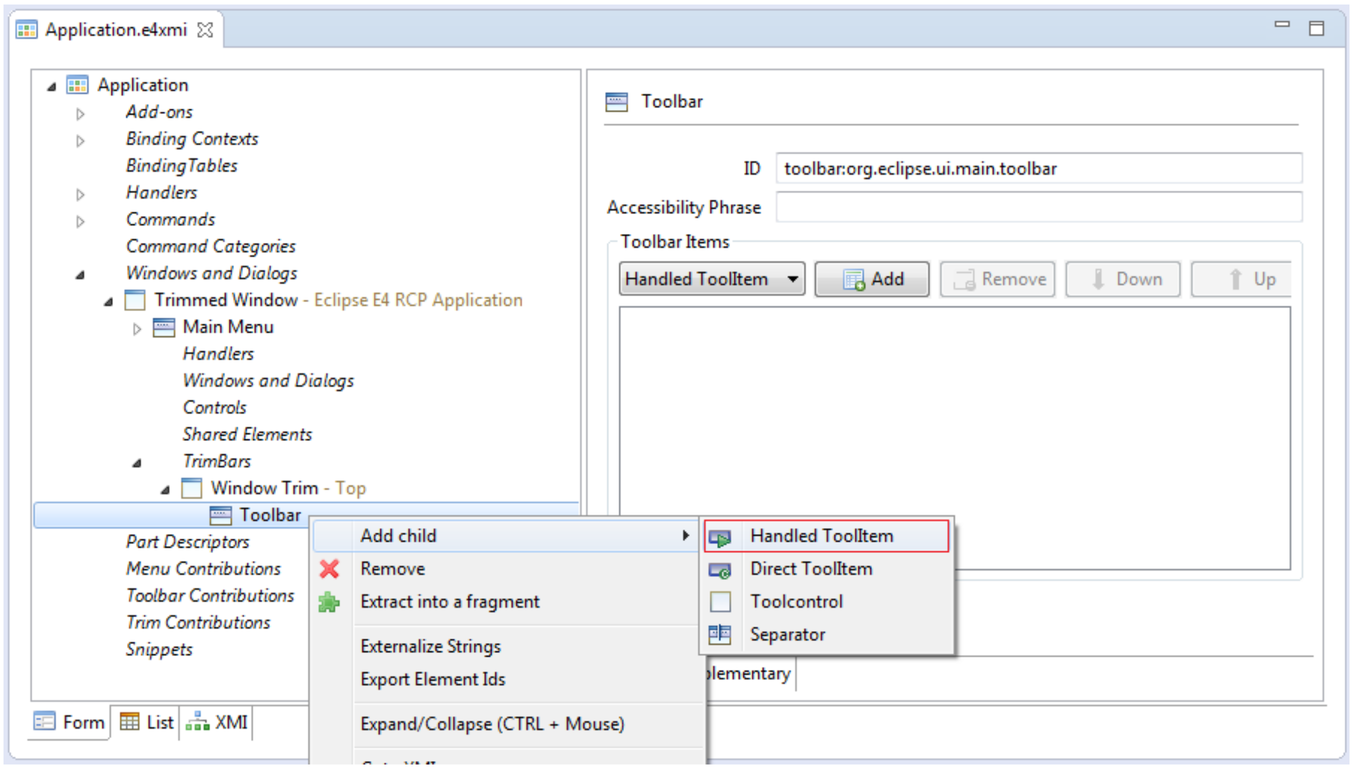




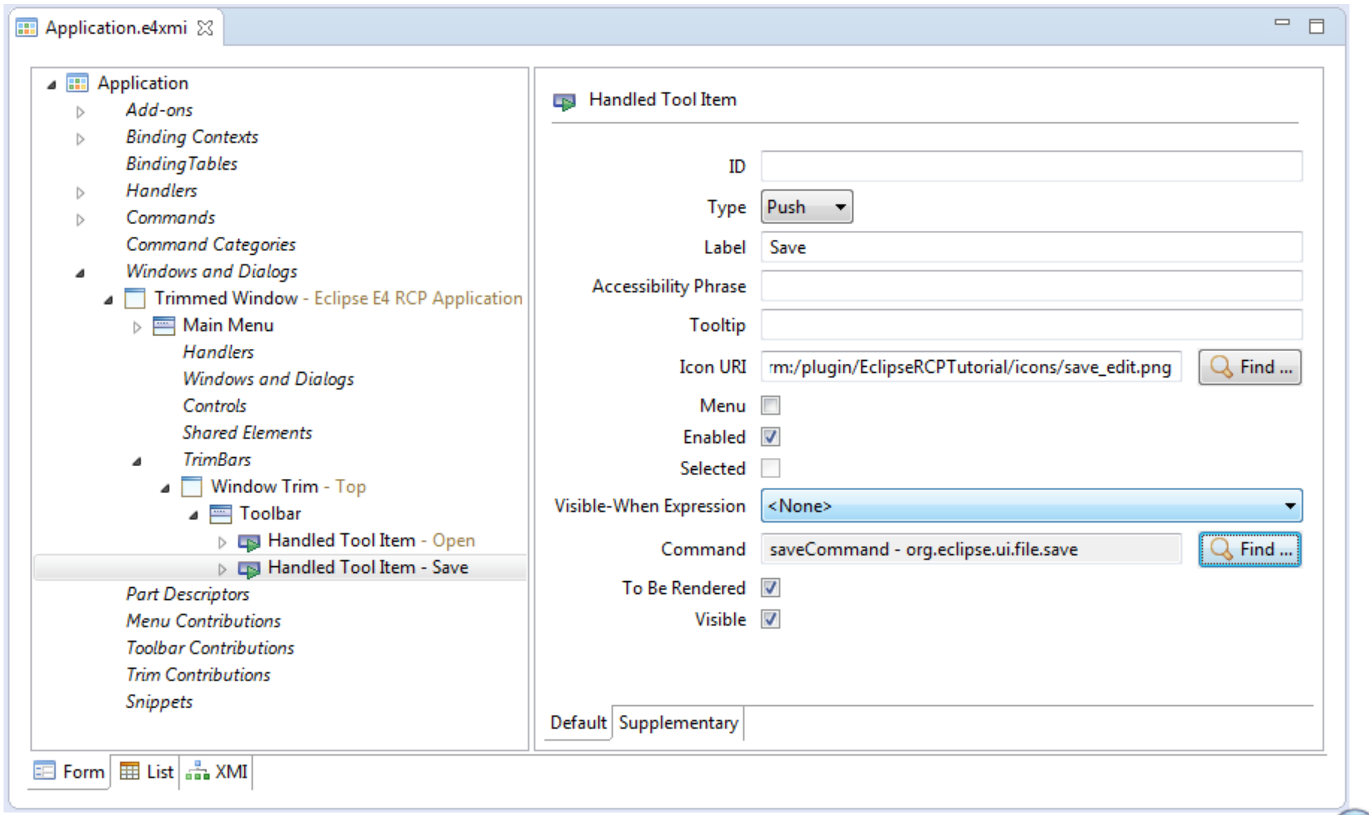
Create Main Toolbar, with ID:

- ID: toolbar:org.eclipse.ui.main.toolbar

Add 2 ToolItem: open & save

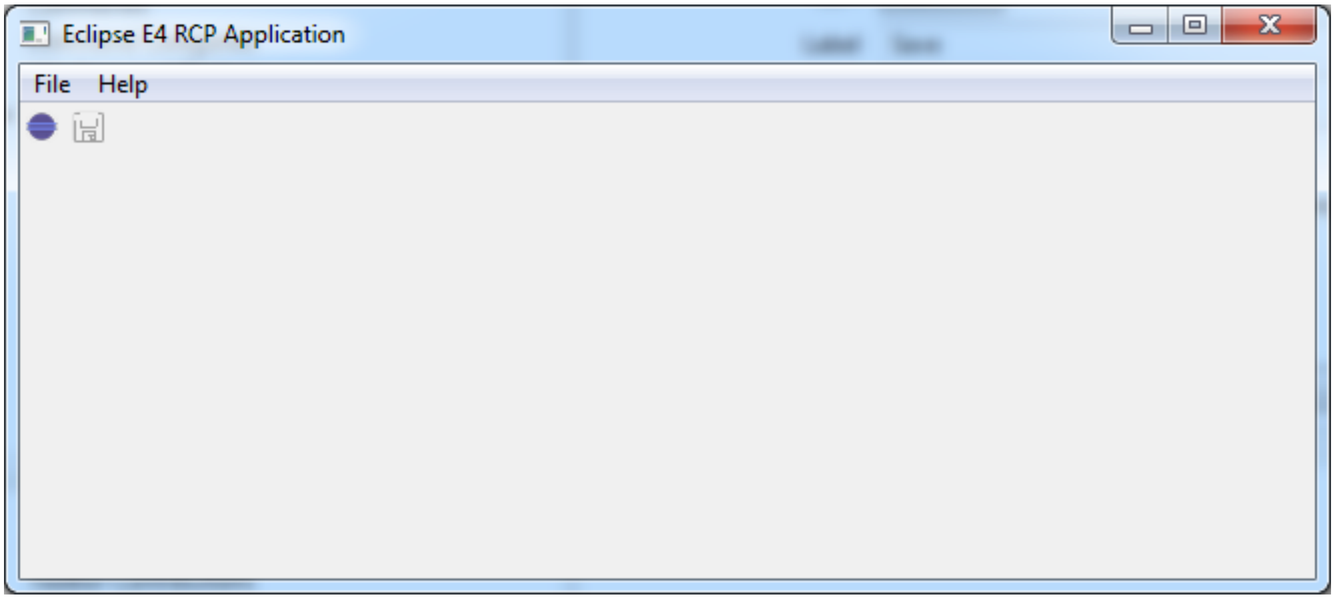


ID: toolbar:org.eclipse.ui.main.toolbar
Accessibility Phrase:



ID:
Type: Push
Label: Save
Accessibility Phrase:
Tooltip:
Icon URI: rm:/plugin/EclipseRCPTutorial/icons/save_edit.png
Menu: ☐
Enabled: ☑
Selected: ☐
Visible-When Expression: <None>
Command: saveCommand - org.eclipse.ui.file.save
To Be Rendered: ☑
Visible: ☑

Rerun your application:

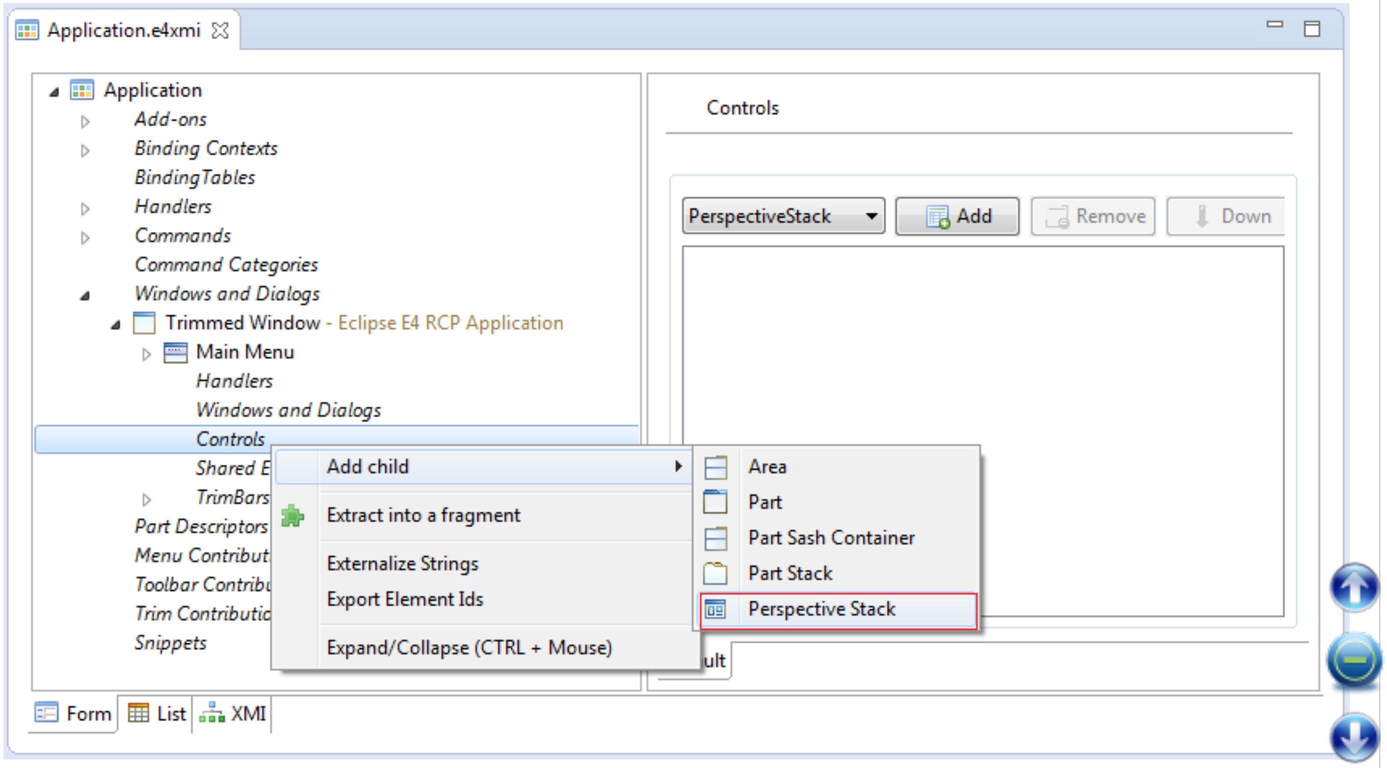


8- Part

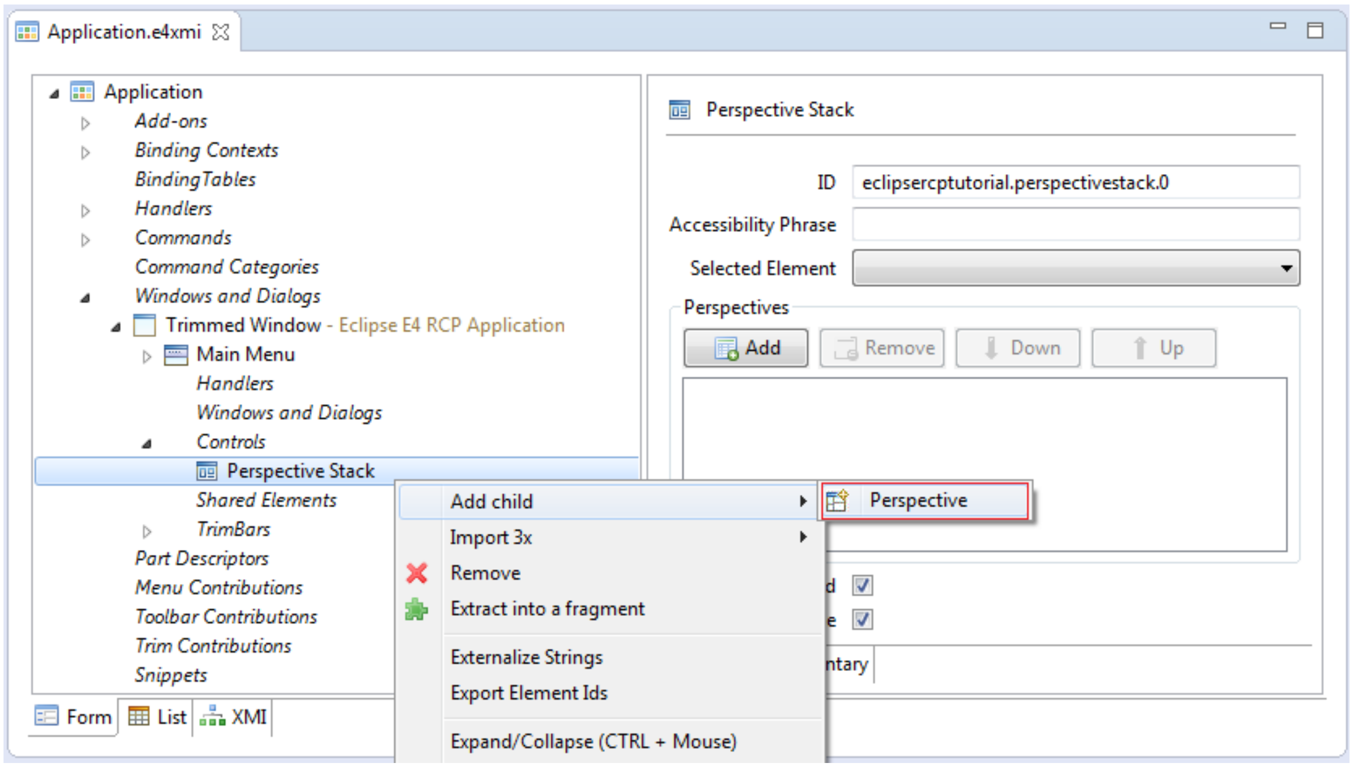
SamplePart.java

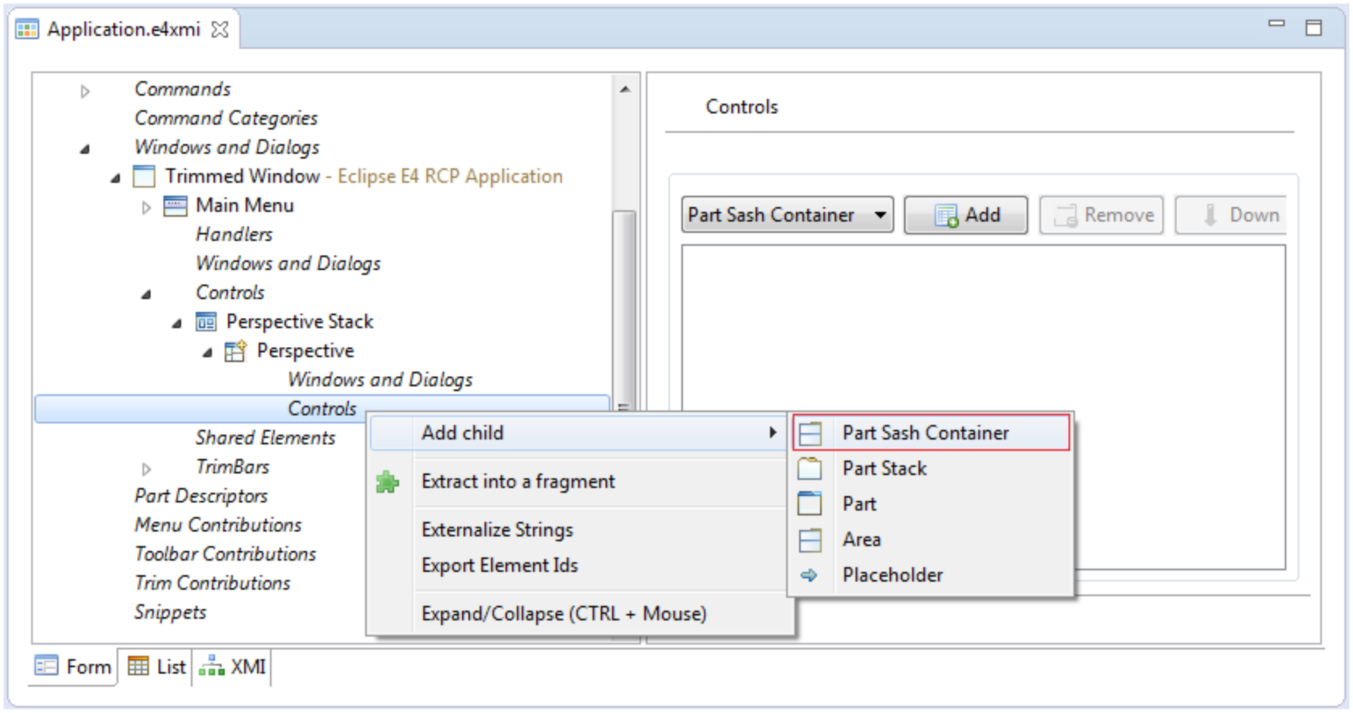
```
/*******************************************************************************
 * Copyright (c) 2010 - 2013 IBM Corporation and others.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 *
 * Contributors:
 *     IBM Corporation - initial API and implementation
 *     Lars Vogel <lars.Vogel@gmail.com> - Bug 419770
 *******************************************************************************/
package org.o7planning.tutorial.rcp.part;

import javax.annotation.PostConstruct;
import javax.inject.Inject;

import org.eclipse.e4.ui.di.Focus;
import org.eclipse.e4.ui.di.Persist;
import org.eclipse.e4.ui.model.application.ui.MDirtyable;
import org.eclipse.jface.viewers.TableViewer;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.ModifyEvent;
import org.eclipse.swt.events.ModifyListener;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Text;

public class SamplePart {

    private Text txtInput;
    private TableViewer tableViewer;

    @Inject
    private MDirtyable dirty;

    @PostConstruct
    public void createComposite(Composite parent) {
        parent.setLayout(new GridLayout(1, false));

        txtInput = new Text(parent, SWT.BORDER);
        txtInput.setMessage("Enter text to mark part as dirty");
        txtInput.addModifyListener(new ModifyListener() {
            @Override
            public void modifyText(ModifyEvent e) {
                dirty.setDirty(true);
            }
        });
        txtInput.setLayoutData(new GridData(GridData.FILL_HORIZONTAL));
        tableViewer = new TableViewer(parent);

        tableViewer.add("Sample item 1");
        tableViewer.add("Sample item 2");
        tableViewer.add("Sample item 3");
        tableViewer.add("Sample item 4");
        tableViewer.add("Sample item 5");
        tableViewer.getTable().setLayoutData(new GridData(GridData.FILL_BOTH));
    }

    @Focus
    public void setFocus() {
        tableViewer.getTable().setFocus();
    }

    @Persist
    public void save() {
        dirty.setDirty(false);
    }
}
```

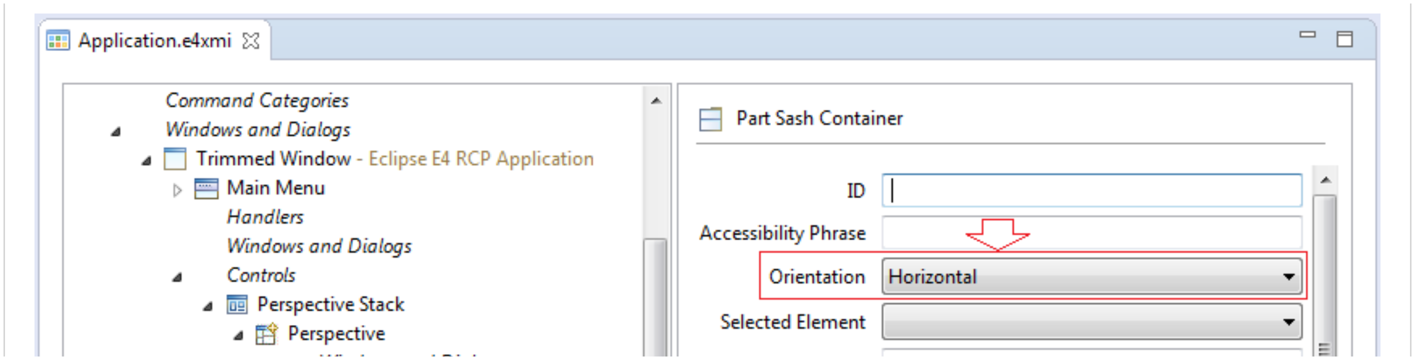
Add new **PerspectiveStack**. This is a stack, it can contain the perspective.



Add new Perspective:






Application.e4xmi
Command Categories
Windows and Dialogs
Trimmed Window - Eclipse E4 RCP Application
Main Menu
Handlers
Windows and Dialogs
Controls
Perspective Stack
Perspective
Part Sash Container
ID
Accessibility Phrase
Orientation
Horizontal
Selected Element