Eclipse Plug-in Development

Utilizing Progress Developer Studio for OpenEdge 11.x APIs

Yogesh Devatraj
Progress Software

Agenda

- Overview Eclipse
- History & Evolution of Eclipse platform
- Overview of Eclipse Architecture
- Tooling support for Plug-in development
- Getting started with PDSOE 11.x APIs
- Demo : Using PDSOE APIs to visualize OpenEdge project dependencies.
- Q&A

Overview

“An integrated development environment (IDE) for anything and nothing in particular.”

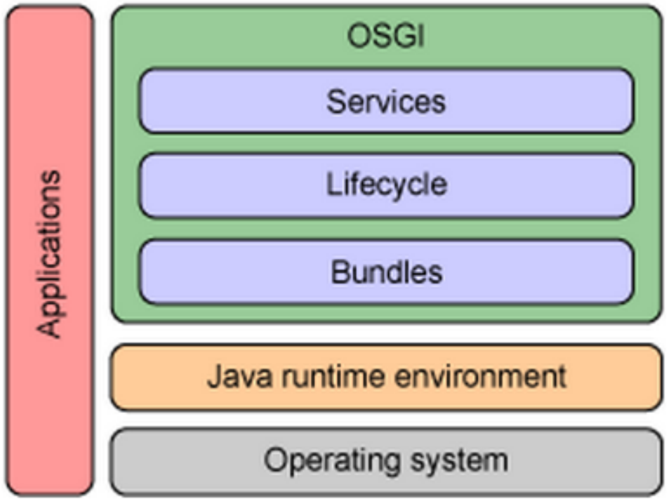
- Not just another set of tools, but a framework.
- A component-based platform that could serve as the foundation for building tools for developers.
- Let focus on building new tool, instead of dealing with infrastructure issues.

History & Evolution

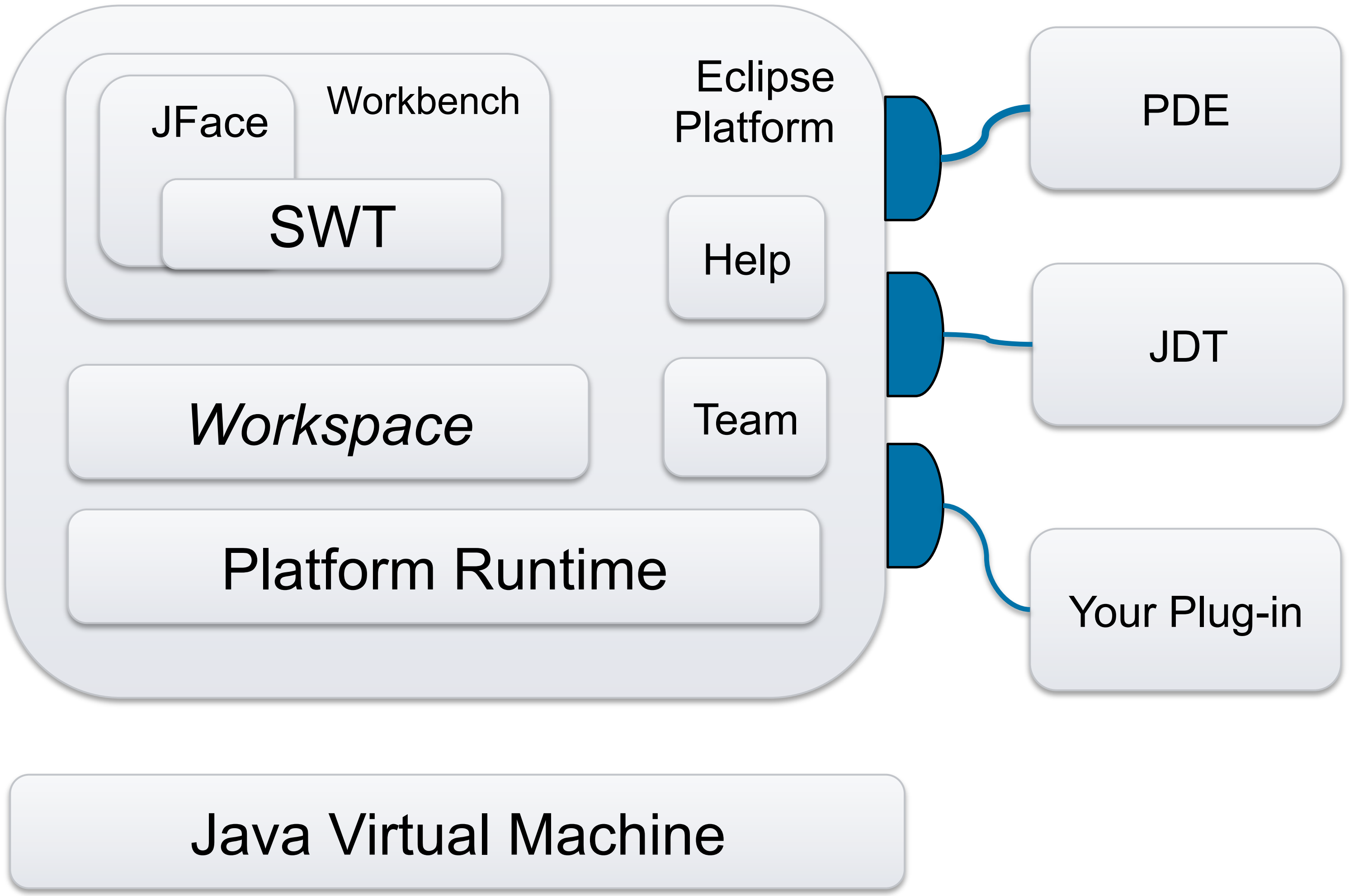
- Some of the initial code that was donated was based on VisualAge for Java, developed by IBM.
- First version (1.0) released Nov 2001.
- In early 2004, the Eclipse Foundation was formed to manage and expand the growing Eclipse community.
- Eclipse 3.X, first major release under this foundation.
- Eclipse 4.X, next generation major new version, released in July 2012.

Eclipse & Equinox

- From v3.0 , Eclipse has adapted OSGi over its proprietary plug-in system.
- Equinox, reference implementation of OSGi R4 specification, base of Eclipse plug-in system.
- Equinox is responsible for developing and delivering the OSGi framework implementation used for all of Eclipse as well as open for all
- OSGi:*bundle* = Eclipse:*plug-in*



Basic Architecture : Eclipse Platform

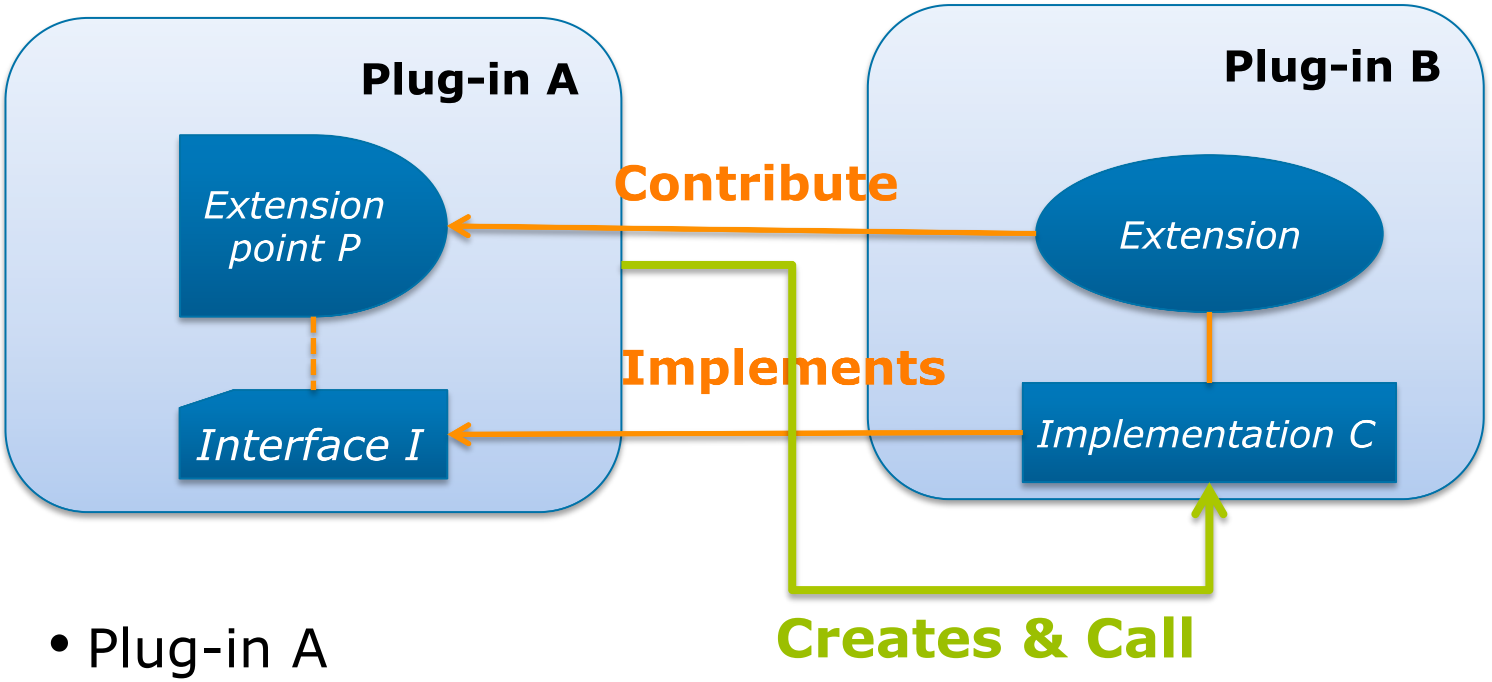


What is a Plug-in ?

“A plugin is essentially a JAR file with a manifest which describes itself, its dependencies, and how it can be utilized, or extended”

- In Eclipse, everything is a plug-in.
- Describes itself to the system using an OSGi manifest (MANIFEST.MF) file and a plug-in manifest (plugin.xml) file.
- Uses extension point to interact with each other.
- Can expose functionality as contributions to other extensions or define their own extension points, to which other bundles may contribute

Basic Architecture : Eclipse Plug-in



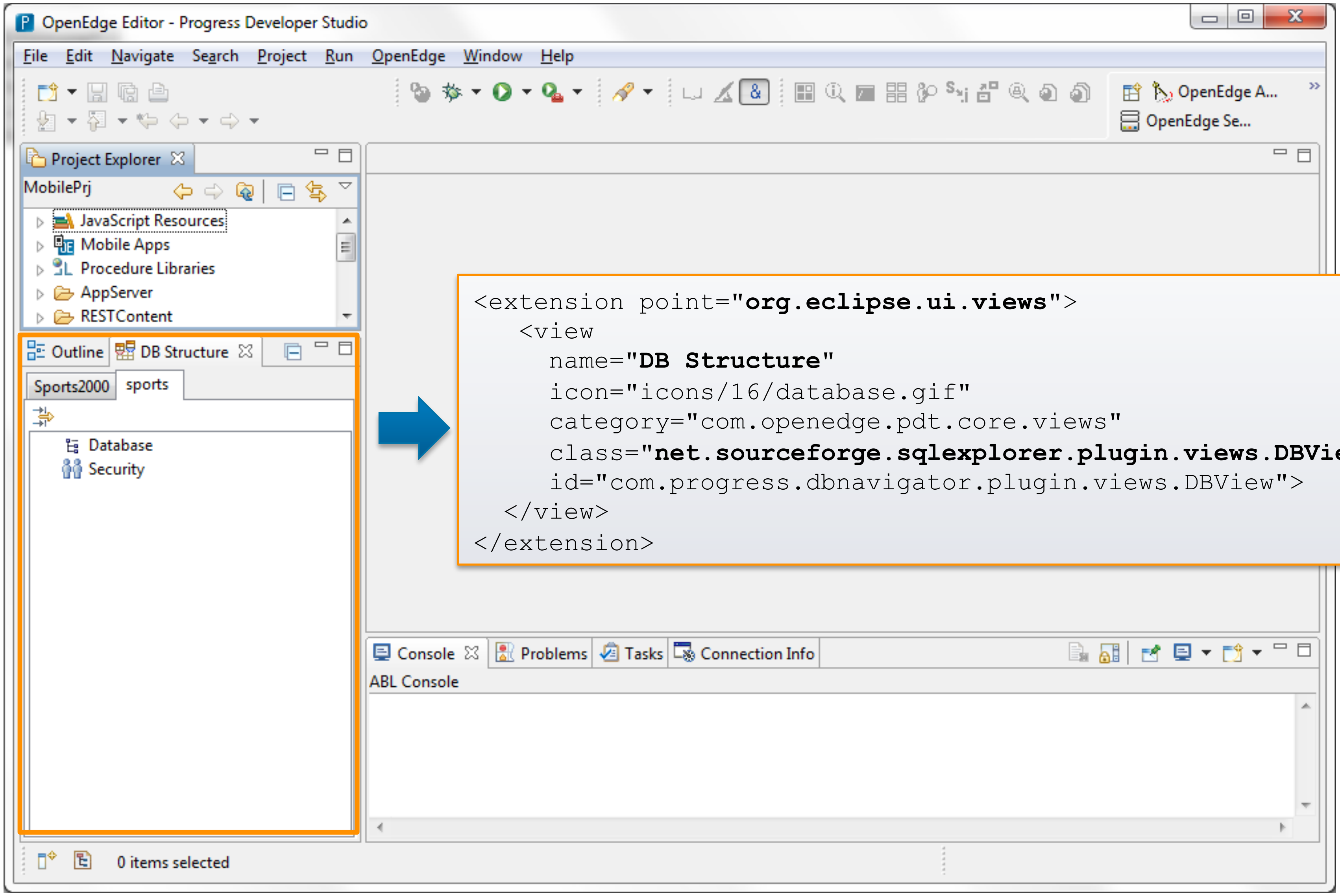
- Plug-in A
  - Declares extenuation point A
  - Declares Interface to implement
- Plug-in B
  - Contribute to extension point by providing implementation C for I.
- Plug-in A instantiate C and call interface methods.

Extension point & extension

```
<schema
    targetNamespace="com.eclipse.plugin.sam
    ple.extension" xmlns="http://
    www.w3.org/2001/XMLSchema">
<annotation>
      <appinfo>
         <meta.schema
    plugin="com.eclipse.plugin.sample.exten
    sion"
    id="com.eclipse.plugin.sample.exte
    nsion.greet" name="Greet"/>
      </appinfo>
      <documentation/>
   </annotation>
   <element name="extension">
     ....
      <complexType>
         <sequence minOccurs="1"
    maxOccurs="unbounded">
            <element ref="client"/>
         </sequence>
         <attribute name="point"
    type="string" use="required">
            <annotation>
               <documentation/>
            </annotation>
         </attribute>
```

```
 <attribute name="id" type="string">
            <annotation>
               <documentation/>
            </annotation>
         </attribute>
<attribute name="name" type="string">
            <annotation>
               <documentation/>
               <appinfo>
                  <meta.attribute
    translatable="true"/>
               </appinfo>
            </annotation>
         </attribute>
      </complexType>
   </element>
   <element name="client">
      <complexType>
         <attribute name="class"
    type="string" use="required">
            <annotation>
               <documentation/>
               <appinfo>
                  <meta.attribute
    kind="java"/>
               </appinfo>
            </annotation>
         </attribute>
      </complexType>
   </element>
  ....
</schema>
```

Extensions in action : PDSOE DB Structure View



```
<extension point="org.eclipse.ui.views">
   <view
      name="DB Structure"
      icon="icons/16/database.gif"
      category="com.openedge.pdt.core.views"
      class="net.sourceforge.sqlexplorer.plugin.views.DBView"
      id="com.progress.dbnavigator.plugin.views.DBView">
  </view>
</extension>
```

Plug-in loading & activation

- Each eclipse plug-in has its own class loader.
  - Starting/stopping any plug-in independently
  - Having multiple version of same plug-in
  - Restrict class visibility to only exported
- Lazy activation
  - Gets activated only on when needed.
  - Scalable for larger set of installed plug-ins
  - Helps to decrease start-up time.
- Lazy activation doesn't stop plug-in UI contribution disappear till plug-in load.

Plug-in : OSGi Manifests

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Plugin
Bundle-SymbolicName: test.sample.plugin;
singleton:=true
Bundle-Version: 1.0.0.qualifier
Bundle-Activator:
test.sample.plugin.Activator
Require-Bundle: org.eclipse.ui,
 org.eclipse.core.runtime
Bundle-ActivationPolicy: lazy
Bundle-RequiredExecutionEnvironment:
JavaSE-1.6
Export-Package: test.sample.plugin
```

Plug-in on which it depends

List of packages available for others

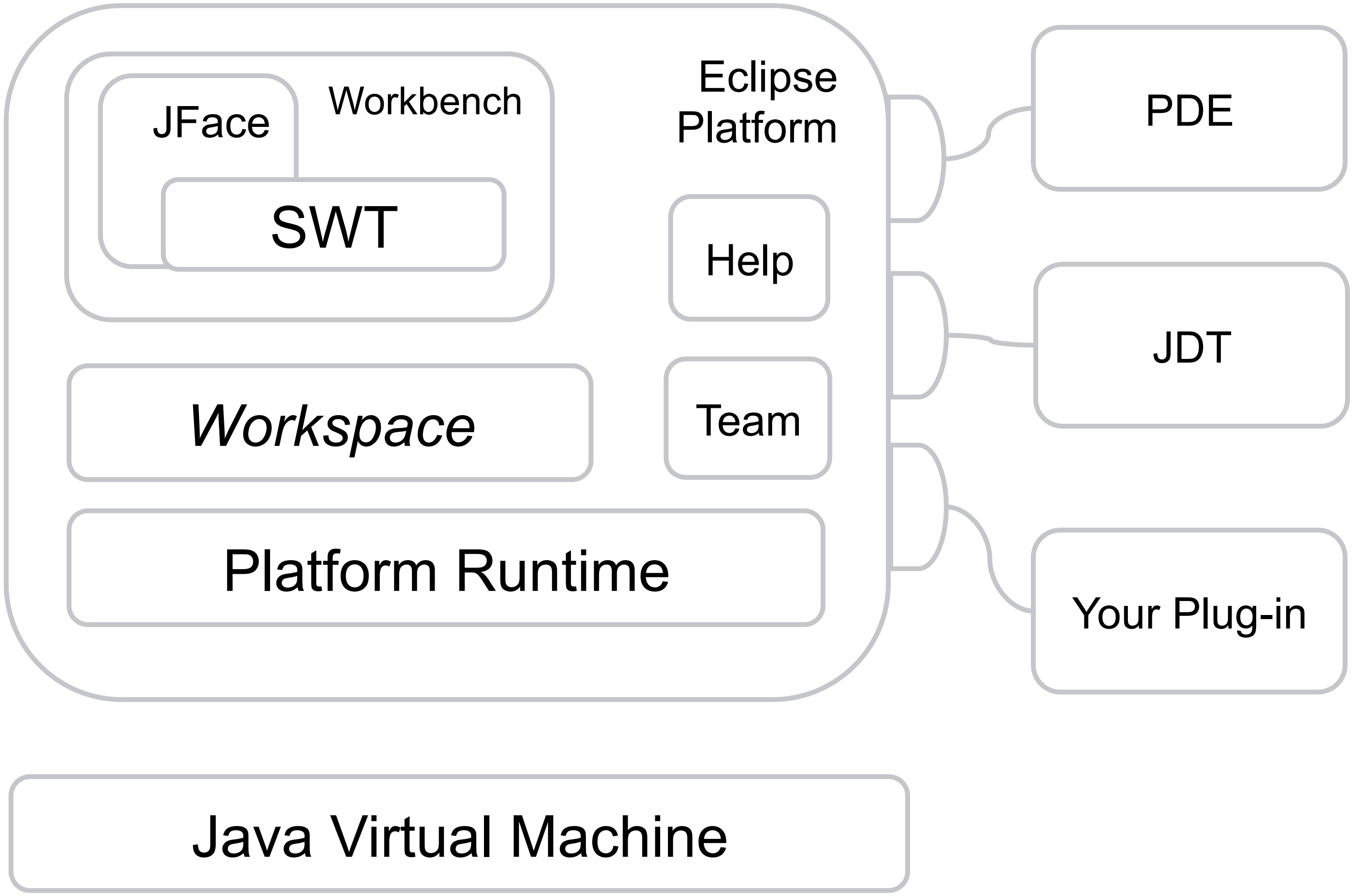
Plug-in : plugin.xml structure

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.4"?>
<plugin>
   <extension
         point="org.eclipse.ui.commands">
      <category
            name="Sample Category"
            id="test.sample.plugin.commands.category">
      </category>
      <command
            name="Sample Command"
            categoryId="test.sample.plugin.commands.category
            id="test.sample.plugin.commands.sampleCommand">
      </command>
   </extension>
.....
 <extension-point id="test.sample.plugin.myextension"
   name="test.sample.plugin.myextension"
   schema="schema/test.sample.plugin.myextension.exsd">
 </extension-point>
```

Introducing new Command by contributing to org.eclipse.ui.commands extension point.

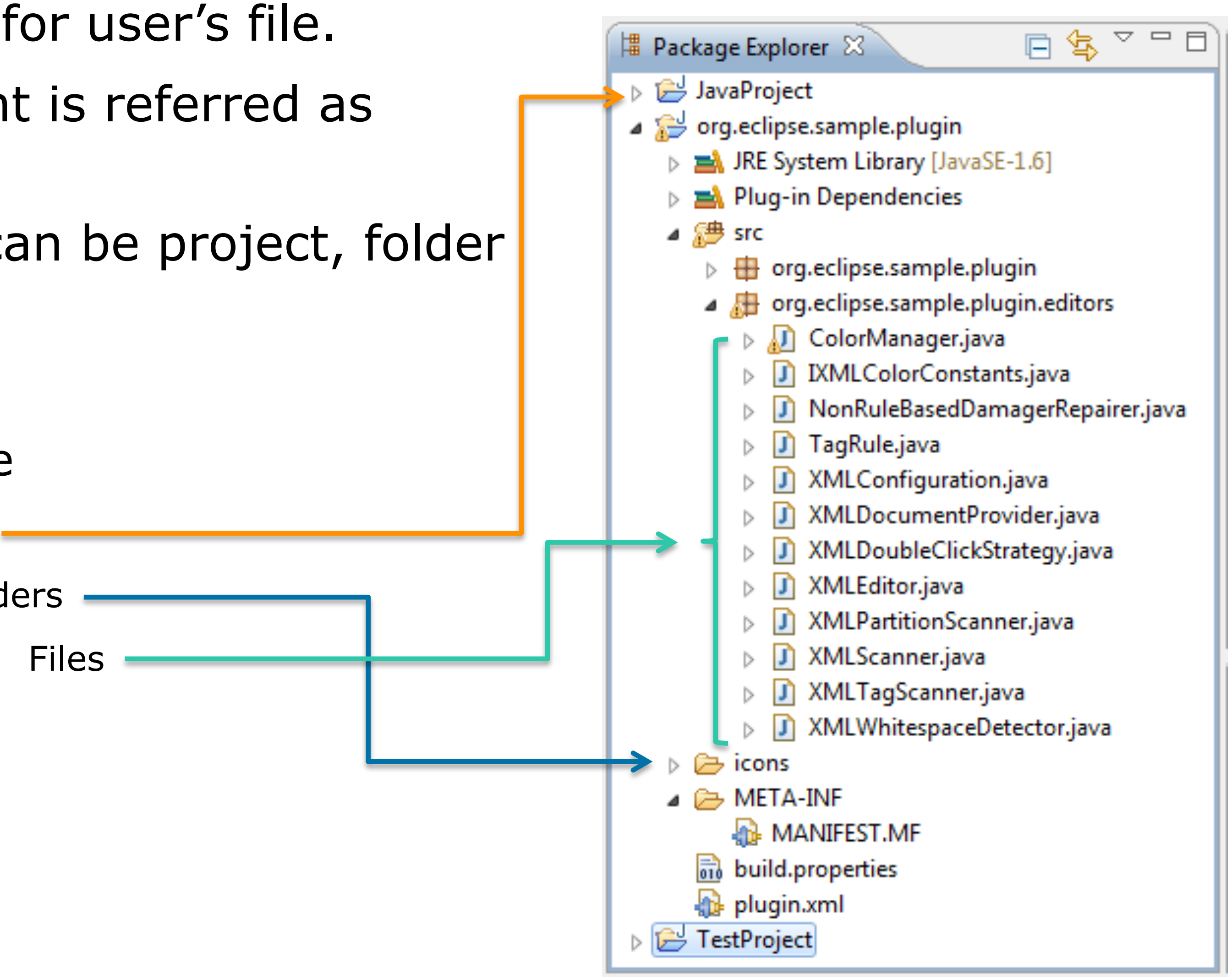
Exposing extension for other to contribute

Eclipse platform : Workspace



Eclipse platform : Workspace

- Central hub for user's file.
- Each element is referred as Resource.
- A resource can be project, folder or file.
- Hierarchy
  - Workspace
    - Project
      - Folders
        - Files
    - Project
    - Files



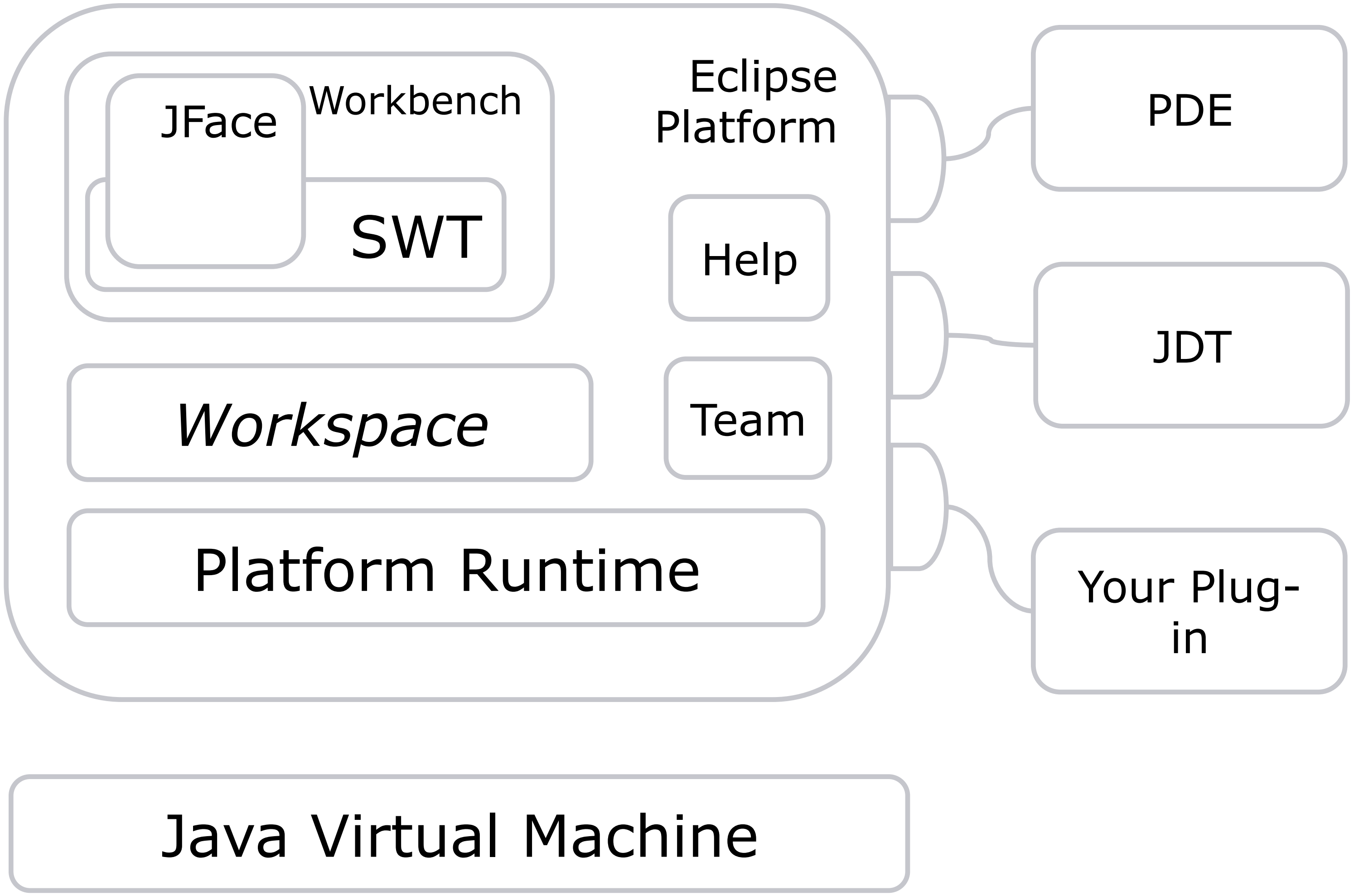
Workspace : Resource

- Workspace organize resource as tree for faster traversing.
- Resource supports several kind of extendable metadata
  - Persistent properties
  - Session properties
  - Markers
  - Project Natures
- Supports change listener to monitor state/lifecycle.
- You can even have derived or linked resources.
- APIs are available to manipulate resource state or properties.

Resource APIs

```
FileInputStream fileStream = null;
try {
/* Get reference of current workspace */
    IWorkspaceRoot myWorkspaceRoot = ResourcesPlugin.getWorkspace
    ().getRoot();
    /* Get/find Project reference from the current workspace */
    IProject myWebProject = myWorkspaceRoot.getProject("MyWeb");
    /* open the project if necessary */
    if (myWebProject.exists() && !myWebProject.isOpen()) {
          myWebProject.open(null);
    }
    IFolder imagesFolder = myWebProject.getFolder("images");
    if (imagesFolder.exists()) {
          /* create a new file */
          IFile newLogo = imagesFolder.getFile("newLogo.png");
          fileStream = new FileInputStream("newLogo.png");
          newLogo.create(fileStream, false, null);
    }
} catch (Exception e) {
    e.printStackTrace();
} finally{
    if(fileStream != null){
    fileStream.close();
    }
}
```

Eclipse Workbench



Eclipse Workbench : SWT

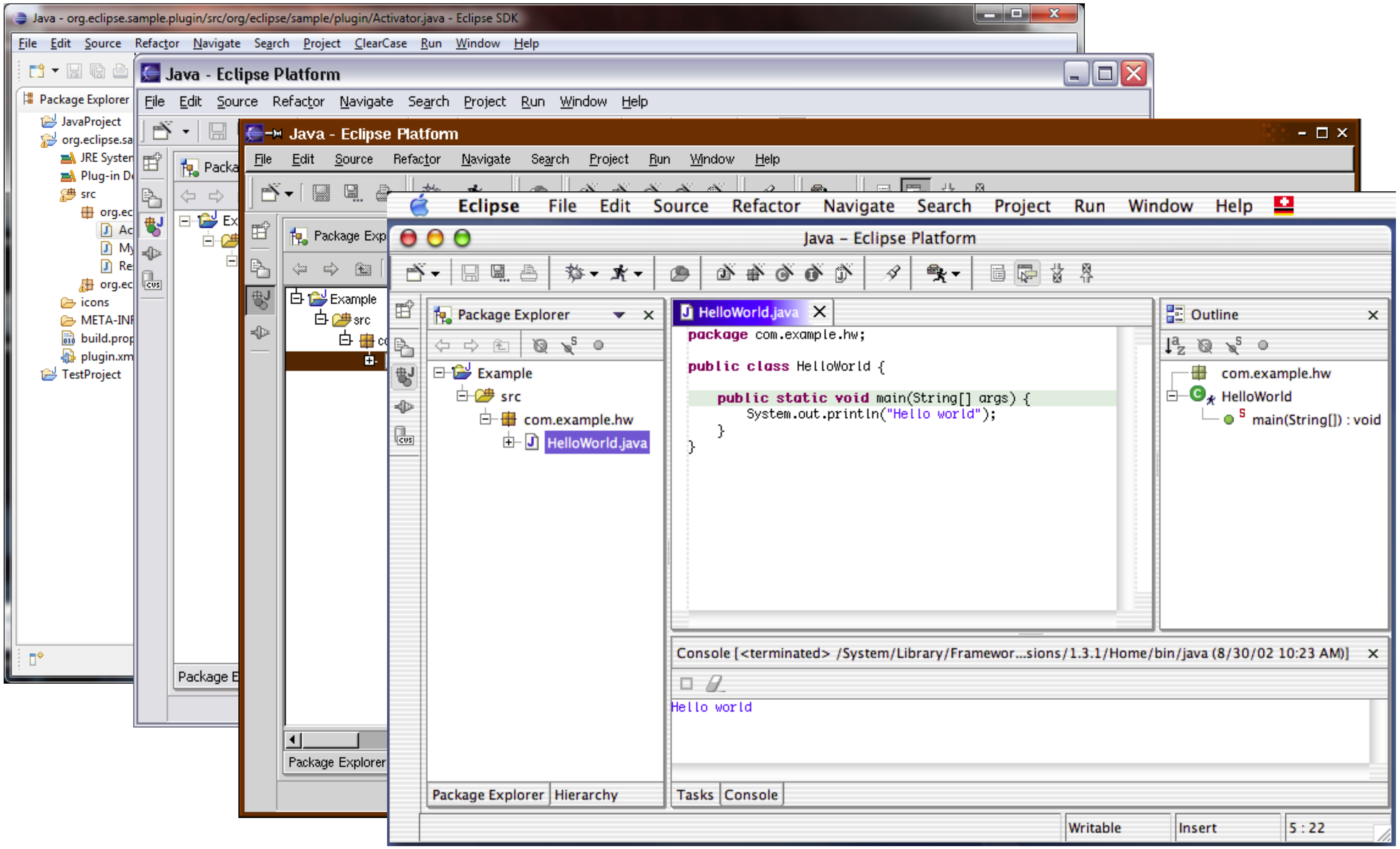
PROGRESS software

"Standard Widget Toolkit"

- Generic graphics and GUI widget set
  - buttons, lists, text, menus, trees, styled text...
- SWT overcomes problems faced by AWT as well as Swing.
- Simple, Small, Fast & native
- OS-independent API
- Uses native widgets where available
- Emulates widgets where unavailable

Eclipse Workbench : SWT

Windows 7 : Windows XP : Linux : Mac OS



Java - org.eclipse.sample.plugin/src/org/eclipse/sample/plugin/Activator.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project ClearCase Run Window Help
Package Explorer
JavaProject
TestProject
Java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Eclipse File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
Example
src
com.example.hw
HelloWorld.java
Outline
HelloWorld
main(String[]) : void
Package Explorer Hierarchy
Tasks Console
Writable Insert 5 : 22

```
package com.example.hw;

public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello world");
    }
}
```

Console [<terminated> /System/Library/Framewor...sions/1.3.1/Home/bin/java (8/30/02 10:23 AM)]

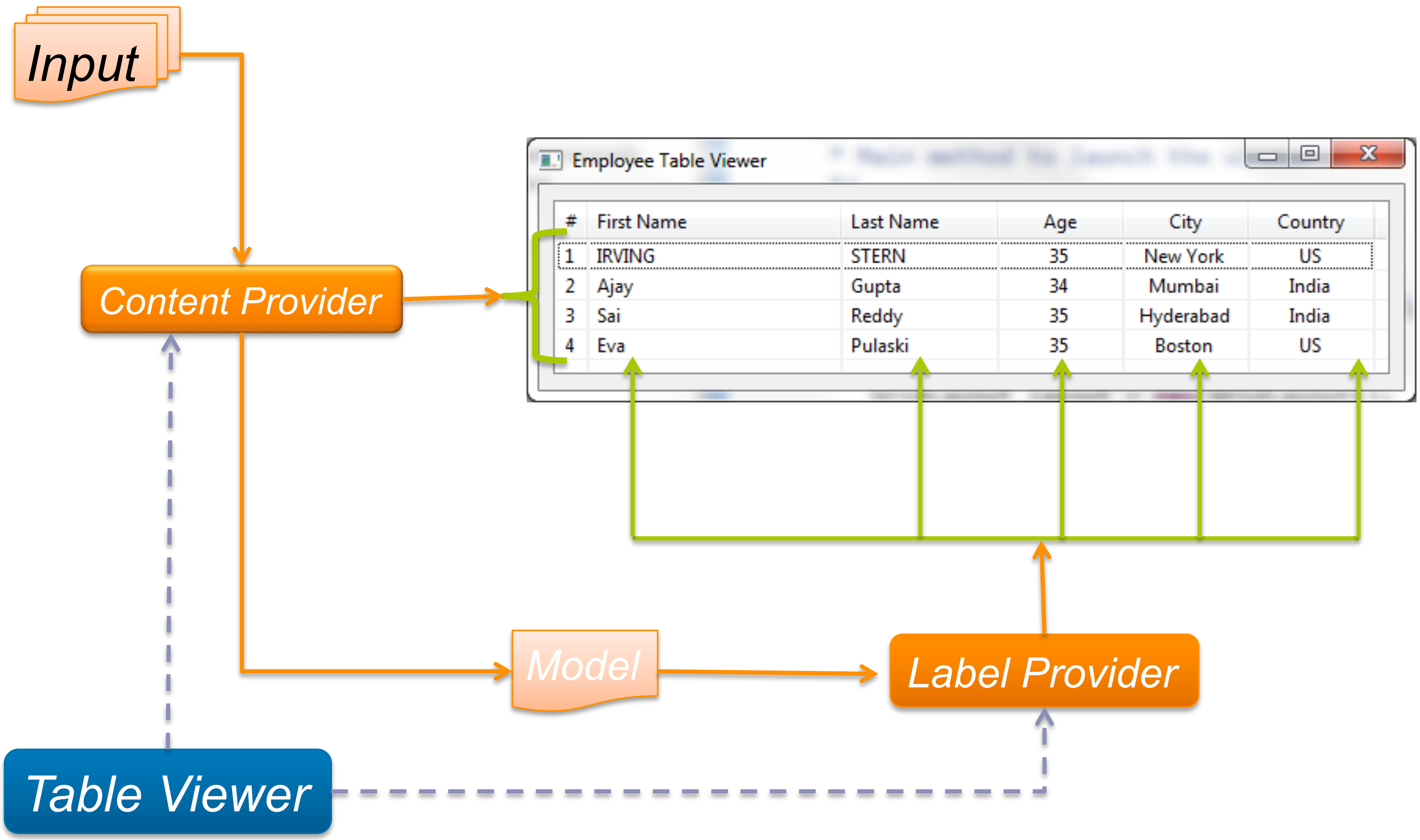
```
Hello world
```

Eclipse Workbench : JFace

"JFace is set of UI frameworks for common UI tasks"

- Provides utility object to use or classes to extend to achieve common functionality.
- Handler common UI programming tasks like
  - Viewer (List, Table & Tree)
  - Image & font registries
  - Dialog & Wizards
  - Field assist
- Unlike SWT, JFace allows to work directly on your domain model.

JFace : Table Viewer Example

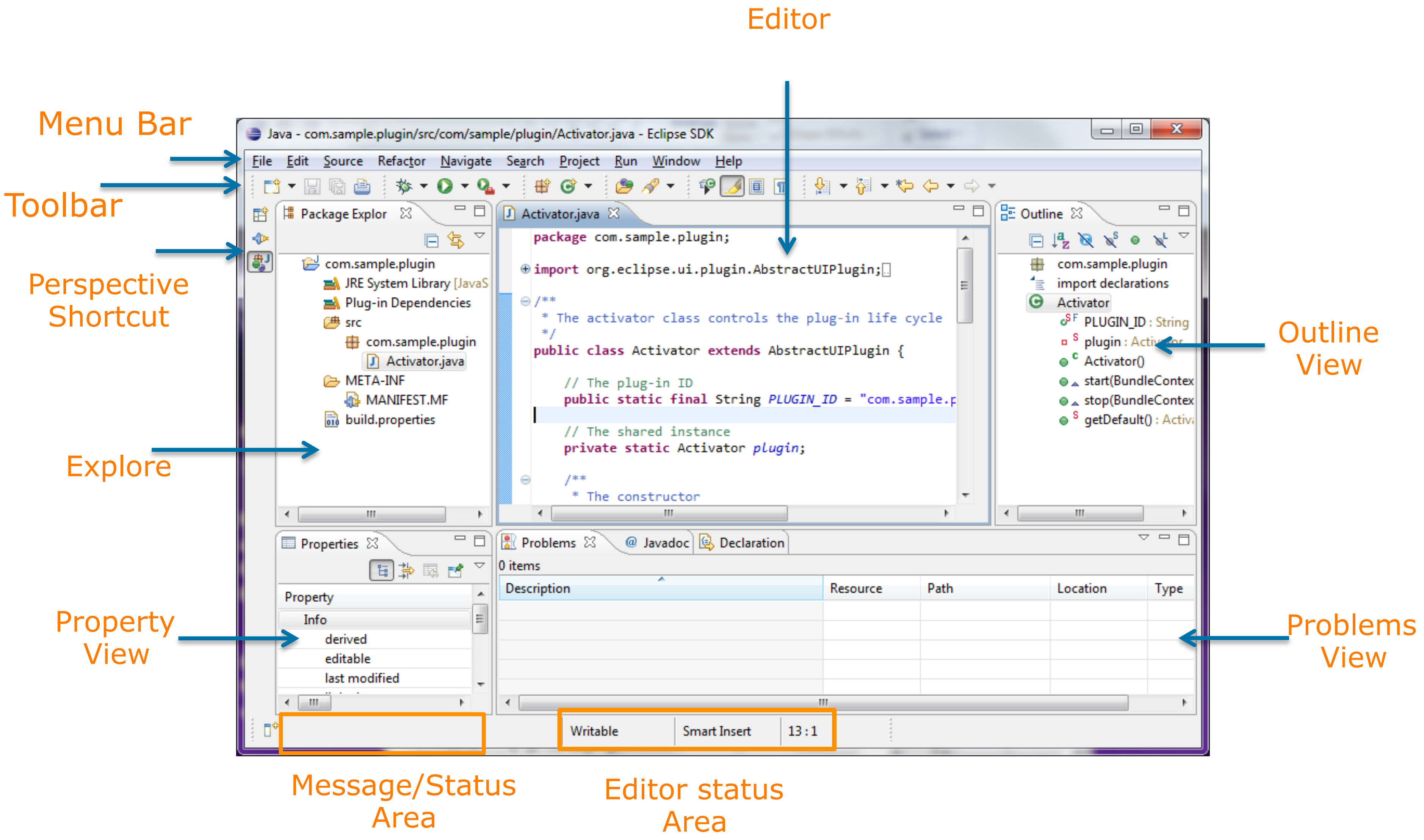


#	First Name	Last Name	Age	City	Country
1	IRVING	STERN	35	New York	US
2	Ajay	Gupta	34	Mumbai	India
3	Sai	Reddy	35	Hyderabad	India
4	Eva	Pulaski	35	Boston	US

Eclipse Workbench

- Workbench brings together all UI components.
- Centred around
  - Editor
  - Views
  - Menus
  - Toolbars
  - Status bar

Eclipse Workbench



Package Explorer:

```
com.sample.plugin
  JRE System Library [JavaS
  Plug-in Dependencies
  src
    com.sample.plugin
      Activator.java
  META-INF
    MANIFEST.MF
  build.properties
```

Activator.java:

```
package com.sample.plugin;

import org.eclipse.ui.plugin.AbstractUIPlugin;

/**
 * The activator class controls the plug-in life cycle
 */
public class Activator extends AbstractUIPlugin {

    // The plug-in ID
    public static final String PLUGIN_ID = "com.sample.p

    // The shared instance
    private static Activator plugin;

    /**
     * The constructor
```

Outline:

```
com.sample.plugin
import declarations
Activator
  PLUGIN_ID : String
  plugin : Activator
  Activator()
  start(BundleContex
  stop(BundleContex
  getDefault() : Activ
```

Properties:

Property
Info
derived
editable
last modified

Problems: 0 items

Description	Resource	Path	Location	Type

Writable | Smart Insert | 13 : 1

Editor ,Views & Perspective

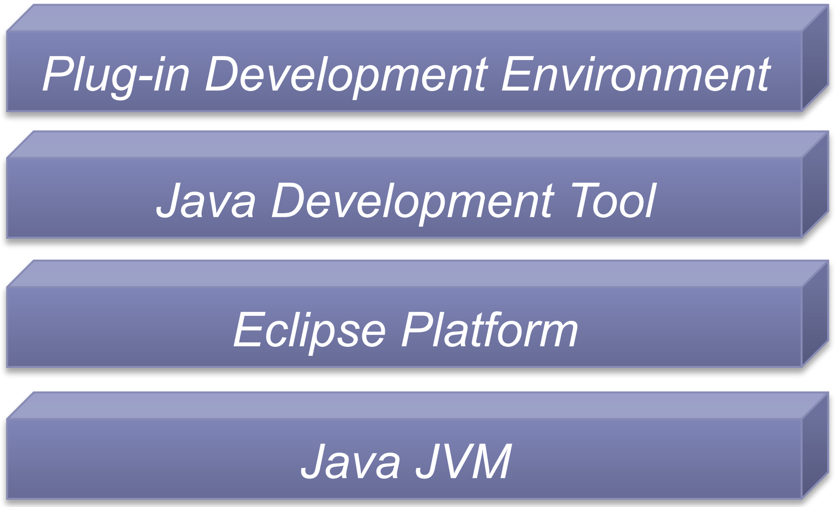
- Editor
  - Editors appear in workbench editor area
  - Contribute actions to workbench menu and tool bars
  - Open editors are stacked
- View
  - Views provide information on some object
  - Views augment editors & other view
    - Example: Outline view summarizes content or Properties view describes selection
- Perspective
  - A perspective defines the initial set and layout of views menu, toolbar and editors in the Workbench window.
  - Workbench supports customizing any perspective

Tooling Support for Plug-in Development

“Plug-in Development Environment (PDE) provides tools to create, develop, test, debug, build and deploy Eclipse plug-ins, fragments, features, update sites and RCP products.”

PDE main components

- **UI** : A rich set of models, tools and editors to develop plug-ins and OSGi bundles
  - Form-Based Manifest Editor
  - New Project Creation Wizards
  - Import/Export Wizards
  - Launcher
  - Views
    - Plug-in Registry
    - Plug-in Dependency



Tooling Support for Plug-in Development

- **API Tools** : Tooling to assist API documentation and maintenance
  - Compatibility Analysis
  - API Restriction Tags
  - Version Number Validation
- **PDE Build** : Ant based tools and scripts to automate build processes.

DISCLAIMER

- Further part of session discusses internal API's for Progress Developer Studio for OpenEdge (PDSOE) 11.X and are subject to change at any time without notice. Use at your own risk.
- PDSOE APIs are neither public nor supported.
- Please contact product management to get more details about these APIs.

DISCLAIMER

PDSOE 11.X API : Overview

- To extends eclipse platform to add ABL application development support
  - Ex: Create and configure OpenEdge project from your own plug-in
- APIs organized as
  - OE project creation and configurations
  - Handling OE project PROPATH & other properties.
  - Database connection configuration
  - Creating launch configuration
- Need to PDSOE plug-ins to use these APIs.

PDSOE 11.X API : Getting Started

- Have OpenEdge 11.0 installation.
- Eclipse Plug-in Development Environment 3.6
- Use Eclipse update manager to install PDSOE plug-ins
  - Help->Software Update ->Available Software
  - Select "Add Site"->"Local"
  - Locate P2 repos at ***DLC/oeide/Architect_repo***
  - Select newly added site from list.
  - It will take care all dependencies.
- Use integrateArchitect.bat to provision PDSOE repos to target development eclipse environment.

Creating and Configuring OpenEdge Project

- OpenEdge projected adapted Faceted framework in 11.0 release.
- OpenEdge project supported facets
  - OpenEdge
  - AppServer
  - ChUI
  - Dynamics
  - WebSpeed
  - GUI for .Net
- Facet will take care of configuring & resolving dependencies.

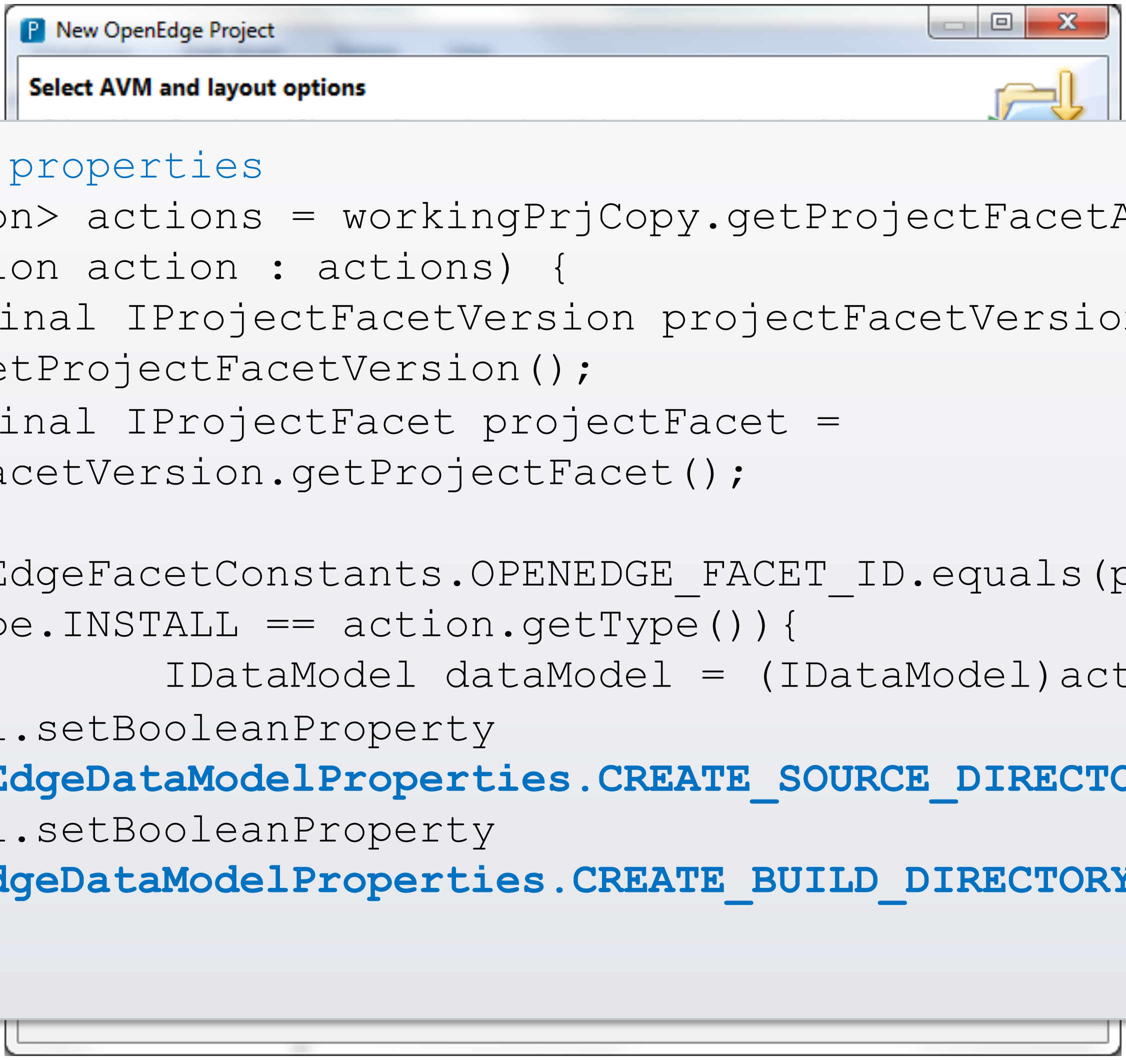
Installing OpenEdge Facet

```
//Create project working copy
IFacetedProjectWorkingCopy  workingPrjCopy =
FacetedProjectFramework.createNewProject();
workingPrjCopy.setProjectName(getProjectName());

//Get OpenEdge Project Facet
IProjectFacet oeProjectFacet = ProjectFacetsManager.getProjectFacet
(IOpenEdgeFacetConstants.OPENEDGE_FACET_ID);
if (oeProjectFacet == null) {
        throw new RuntimeException("Unable to retrieve OpenEdge facet.");
}
IProjectFacetVersion  openedgeFacetVersion = oeProjectFacet.getDefaultVersion();

//Install OE Facet to newly created project.
workingPrjCopy.addProjectFacet(openedgeFacetVersion);
try {
        workingPrjCopy.commitChanges(monitor);
} catch (CoreException e) {
        throw new RuntimeException("Unable to install OpenEdge Facet");
}
```

Update project properties while installing Facet.


New OpenEdge Project

Select AVM and layout options

```
//update properties
Set<Action> actions = workingPrjCopy.getProjectFacetActions();
for (Action action : actions) {
        final IProjectFacetVersion projectFacetVersion =
action.getProjectFacetVersion();
        final IProjectFacet projectFacet =
projectFacetVersion.getProjectFacet();

if(IOpenEdgeFacetConstants.OPENEDGE_FACET_ID.equals(projectFacet.getId
())&& Type.INSTALL == action.getType()){
                IDataModel dataModel = (IDataModel)action.getConfig();
dataModel.setBooleanProperty
(  IOpenEdgeDataModelProperties.CREATE_SOURCE_DIRECTORY, true);
dataModel.setBooleanProperty
( IOpenEdgeDataModelProperties.CREATE_BUILD_DIRECTORY, true);
        }
}
```

Update project properties while installing Facet.

Property Name	Allowed Values	Default Value	Description
`IOEProjectDataModelProperties.CREATE_SOURCE_DIRECTORY`	TRUE/FALSE	False	Set this property to create “src" directory under project.
`IOEProjectDataModelProperties.SOURCE_DIRECTORY_PATH`	Valid path		Custom source directory path for the project
`IOEProjectDataModelProperties.CREATE_BUILD_DIRECTORY`	TRUE/FALSE	False	Set this property to create “r-code" directory under project.
`IOEProjectDataModelProperties.BUILD_DIRECTORY_PATH`	Valid path		Custom build directory for the project.
`IOEProjectDataModelProperties.USE_PROJECT_ROOT`	True/false	true	Set this property to “true” to use project root directory for source and r-code.

IOpenEdgeProject

- PDSOE maintains wrapper reference (*IOpenEdgeProject)* for every eclipse project (*org.eclipse.core.resources.IProject*) with OpenEdge facet installed.
- *IOpenEdgeProject* maintains additional information specific to OpenEdge project like PROPATH, database connection, runtime etc
- Get *IOpenEdgeProject* reference from *OpenEdgeProjectManager*

```
// Returns null if passed project is not OpenEdge natured
IOpenEdgeProject oeproject = OEProjectPlugin.getDefault()
.getOpenEdgeModel().getOpenEdgeProject(project);
```

PROPATH

- PROPATH maintains list of entries where the AVM searches for files and procedures.
- Every OpenEdge project has own PROPATH.
- OpenEdge projects use an xml file (.propath) in the project’s root directory to store PROPATH information.
- Closely monitored for any changes and changes picked up by OE project environment.

PROPATH Entry Type

PROPATH Entry Type	Description
PROPATH_DIRECTORY	Refers to a file on the local file system by absolute path
CONTAINER	A container is a PropathEntry that resolves to multiple values. Extend `com.openedge.pdt.core` to introduce your own.
SOURCE_DIRECTORY	A source directory points to a folder within the project.
PROCEDURE_LIBRARY	A procedure points to the location of an OpenEdge .pl file.

Adding/Updating Project PROPATH

- Create PROPATH Entry

```
//Adding PROPATH enty for source
//source folder under project directory
IFolder sourceFolder = project.getFolder(new Path("src"));
sourceFolder.create(false, true, monitor);
//PROPATH Entry
PropathEntry srcEntry = new                PropathEntry
(PropathConstants.SOURCE_DIRECTORY);
srcEntry.setPath(PropathConstants.ROOT_VARIABLE + "/" +
        sourceFolder.getName());
srcEntry.setEnvironment(new PropathEnvironmentEntry
(PropathConstants.ALL_ENVIRONMENTS));
```

- Persisting to .propath

```
IFile propathFile = project.getFile(PropathConstants.PROPATH_NAME);
PropathWriter propathWriter = new PropathWriter(propathFile);
PropathEntry propathEntries[] = new PropathEntry[] {rootEntry, srcEntry,
stdLibs};
propathWriter.savePropathDefinition(propathEntries, monitor);
```

Database Connection

- OpenEdge project need connection to handle database related operations.
- PDSOE manage set of database connections profiles at workspace level.
- OpenEdge project can be associated with one or more database connection profile.
- This information is persisted in .dbconnection under project's root.
- Connection profiles are shared among workspace project, but each AVM has its own connection to database.

Creating & associating DB connection

- Create DB connection profile

```
//Create database connection profile
String databaseName = new Path(getDatabaseConnection())
         .removeFileExtension().lastSegment();
DatabaseConnectionProfile dbProfile = new DatabaseConnectionProfile
(databaseName);
dbProfile.setPhysicalName(getDatabaseConnection());
UUID uuid = UUID.randomUUID();
dbProfile.setIdentifier(uuid.toString());
```

- Persist & associate to project

```
DatabaseConnectionManager dbMgr = OEProjectPlugin.getDefault
().getDatabaseConnectionManager();
//Persist DB connection profile
dbMgr.addDatabaseConnectionProfile(new DatabaseConnectionProfile[]
{dbProfile});
//Associate to OpenEdge project
dbMgr.assignDatabaseConnectionProfile(new DatabaseConnectionProfile[]
{dbProfile}, project);
```

Accessing & setting project properties

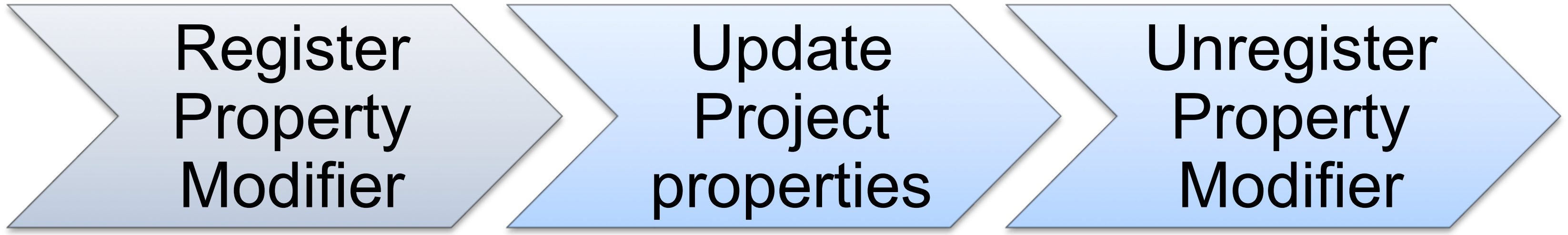
- IOpenEdgeProjectConfiguration interface provides a read-only view of the properties associated with the project.

```
IOpenEdgeProjectConfiguration oeProjectConfig = oeproject.getConfiguration;
Ipath rCodePath = oeProjectConfig.getRcodePath();
```

- Monitor project properties

```
//Creating Listener
IProjectPropertyUpdateListener listener = new
IProjectPropertyUpdateListener() {
    public void propertiesUpdated(IOpenEdgeProject oeproject) {
        //un-register listerner
        configuration.removePropertyUpdateListener(this);
        // Do you job ...
    }
};
//Registering listener
configuration.addPropertyUpdateListener(listener);
```

Updating OpenEdge project properties


Register Property Modifier
Update Project properties
Unregister Property Modifier

```
//Property Modifier
IProjectPropertyModifier modifier = new IProjectPropertyModifier() {
        //..     …
}
//Register Property Modifier
oeproject.getConfiguration().addPropertyModifier(modifier);
//Update Project properties
OEProperties.setProperty(oeprojec,
IOEProjectProperties.P_PRO_VERSION, proversion);
// Unregister modifier to rebuild project configuration
oeproject.getConfiguration().removePropertyModifier(modifier)
```

Access OpenEdge project runtime

- Each OpenEdge project is associated with either its own AVM runtime or shared AVM runtime.
- Each OE project has reference to its associated runtime.

```
IAVMClient runtime = oeProject.getRuntime();
boolean connected = runtime.isConnected();
if(connected){
    boolean available = runtime.isAvailable();
    if(available){
        //your stuff here..
    } else {
    `   // register listener to get notified on AVM runtime state
        runtime.addAVMRuntimeListener(new AVMRuntimeListenerAdapter
        () {
            @Override
            public void runtimeAvailable(IAVMClient runtime) {
            //your stuff here..
            }
        });
    }
}
```

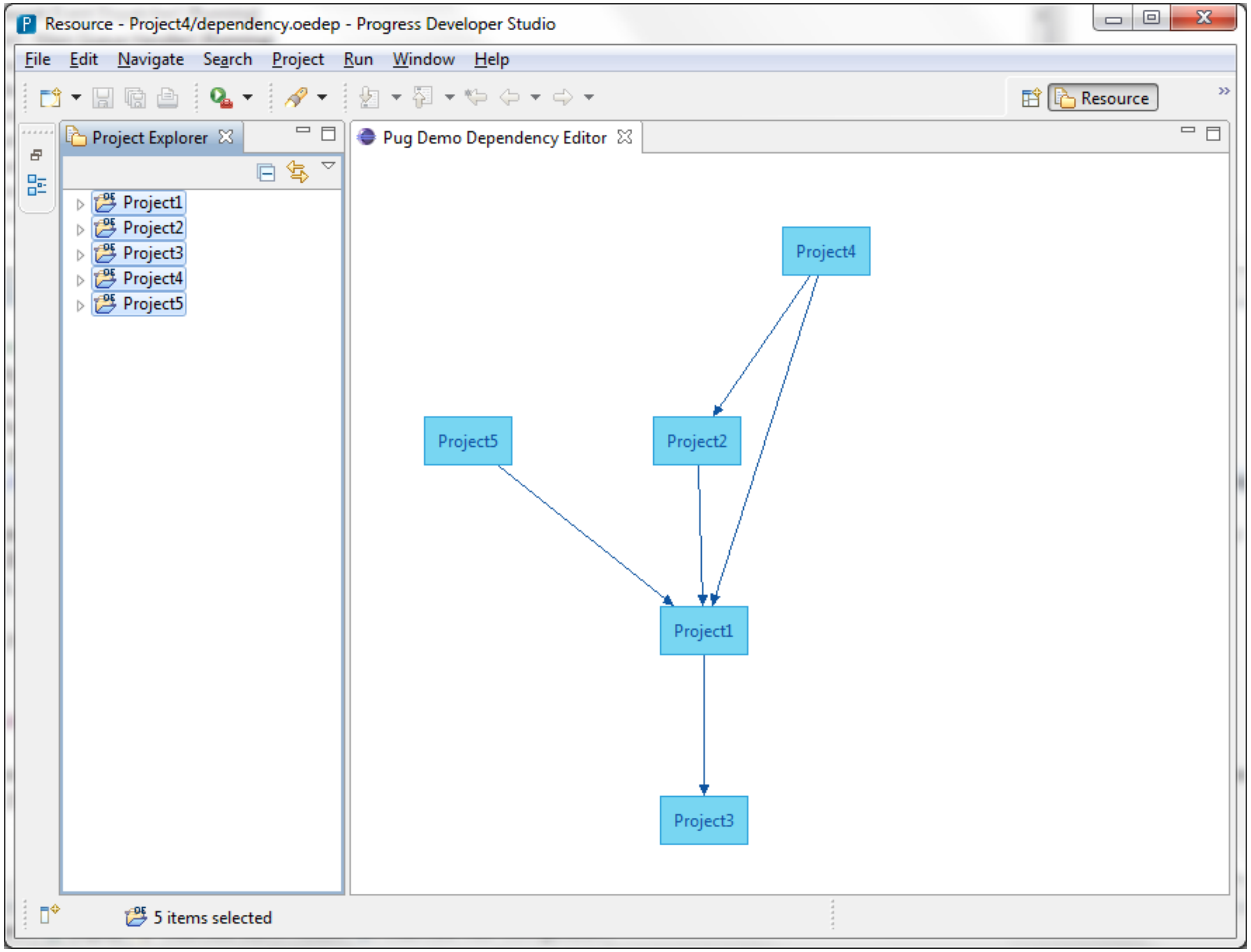
Custom launch configuration

- Extend `org.eclipse.debug.ui.launchShortcuts` extension point.
- Provide implementation for `ILaunchShortcut2.launch (Iselection, String)` or `ILaunchShortcut2.launch (IEditorPart, String)`
- Configure launch configuration with
  - ABL file to be execute
  - Launch mode Run or Debug.
  - PROPATH & ABL runtime
  - Temporary working directory.

Demo

Demo

OpenEdge Projects Dependency Graph


Resource - Project4/dependency.oedep - Progress Developer Studio
File Edit Navigate Search Project Run Window Help
Resource
Project Explorer
Project1
Project2
Project3
Project4
Project5
Pug Demo Dependency Editor
Project4
Project5
Project2
Project1
Project3
5 items selected

PROGRESS EXCHANGE 2013

DISCOVER. DEVELOP. DELIVER.

October 6–9, 2013 • Boston

#PRGS13

www.progress.com/exchange-pug

Special **low rate of $495** for PUG Challenge attendees with the code **PUGAM**

And visit the Progress booth to learn more about the Progress App Dev Challenge!

PROGRESS
software