

## Extending the Cmpware CMP-DK (Version 2.0 for Eclipse 3.1)

Cmpware, Inc.

### Introduction

The *Cmpware Configurable Multiprocessor Development Kit (CMP-DK)* is a multiprocessor simulation and software development environment. It provides fast and efficient modeling of multiprocessor architectures as well as support for software development on such systems. The goal of supporting software development is achieved by providing an interactive, display-rich environment that permits large amounts of information to be displayed in a fast, simple and uncluttered format. Such capabilities are essential in analyzing the behavior of multiprocessor systems.

The *Cmpware CMP-DK* (version 2.0) for *Eclipse* 3.1 and higher contains customizable models for various popular processors, memories and communication links. These models transparently interface to a powerful *Eclipse*-based Integrated Development Environment (IDE) that provides a powerful multiprocessor architecture and software development environment. This environment features displays for:

- |                         |                             |
|-------------------------|-----------------------------|
| ● Source Code Tracing   | ● General Purpose Registers |
| ● Source Code Variables | ● Special Purpose Registers |
| ● Disassembly           | ● Command Line Interface    |
| ● Memory Display        | ● Link Utilization          |
| ● Power Estimator       |                             |

One of the most powerful and useful features of the *Cmpware CMP-DK* is the ability to *extend* its functionality. The existing *Cmpware CMP-DK* can be added to and modified through a simple and standard procedure. Unlike other systems, this does not require access to the system source code; new code can be added without any changes or modification to the existing system. This procedure also uses standard design and implementation development tools. In fact, the standard *Eclipse* development environment can be used for all development of extensions for the *Cmpware CMP-DK*.



## Extending the Cmpware CMP-DK

The *Cmpware CMP-DK* is an *Eclipse*-based development environment. This means that the popular Java-based *Eclipse* Integrated Development Environment (IDE) framework is used to tie together and provide basic functionality for the *Cmpware* development environment. One of the major benefits of the *Eclipse* development environment is that it is *extensible*. That is, it can be added to in various ways using well-defined and self-contained software modules. In fact, the primary mechanism for using *Eclipse* is to provide such *extensions* to supply new functionality. The entire *Cmpware CMP-DK*, as well as nearly all *Eclipse*-based systems, make extensive use of this extension functionality.

While the ability to extend *Eclipse* is a useful and powerful capability, this functionality can be passed on to subsequent layers of software which use *Eclipse*. In fact, the *Cmpware CMP-DK* permits new windows or *views* to be added to the *Cmpware CMP-DK* with relatively little effort. There is no limit to the sophistication of such displays, including complex graphical and interactive views. More complex displays will, however, make more extensive use of the underlying *Eclipse* functionality. This will typically require a higher level of familiarity with the underlying *Eclipse* system.

A simple example is presented in this document to demonstrate the extension of the *Cmpware CMP-DK*. Such extension is likely to be of great value to architectures or even algorithms that may require non-standard display of system data. Or users may simply wish to establish and use new debug and analysis techniques in their system, further adding value to the *Cmpware CMP-DK* for their particular application.

This example focuses on the details of making the extension successfully interact with the *Cmpware CMP-DK* and underlying the *Eclipse* framework. The actual example is kept relatively simple so as not to obscure this part of the design and development process. But this code should be usable as a starting point for more complex extension.

## The Show All PCs View

As an example, a new *view* showing all of the *Program Counters* (PCs) in the current multiprocessor is implemented. While this is a somewhat artificial example, such a display may be useful in some systems or in certain situations. Most importantly, this demonstration covers all of the significant aspects of producing an extension view without getting overly involved in the details of more complex (but powerful) *Eclipse* functionality. This example will display the data in a simple text window and will be updated along with all of the other displays in the *Cmpware CMP-DK*.



The *Show All PCs* example can easily serve as a template to display similar data or as a starting point for potentially more sophisticated views. In particular, this example aims to document the *Eclipse*-specific steps necessary to make such extensions work together with the rest of the system.

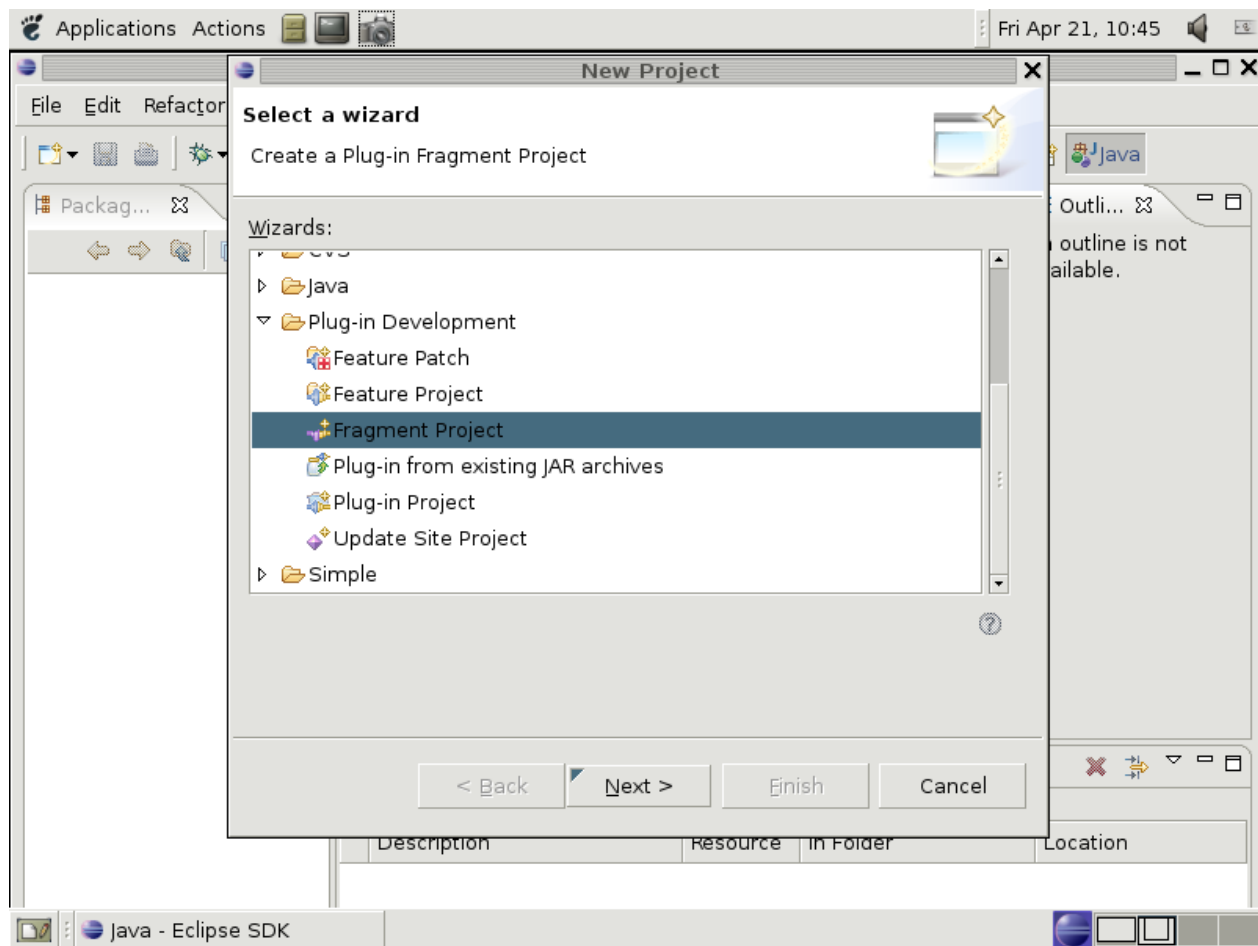


Figure 1: Creating the new AllPCs Fragment development project.

While not overly complex, this particular part of the *Eclipse* system is not especially well documented, and existing documentation often assumes a high level of familiarity with the *Eclipse* system. It is, however, possible to produce such extensions and views with very little overall knowledge of the underlying *Eclipse* system. This example provides all of the necessary steps necessary in producing these extensions.



## The ShowAllPCs Project

This example assumes that *Eclipse* 3.0 or greater is installed on the development system. In addition, the *Cmpware CMP-DK* version 2.0 or greater must also be installed. If the *Cmpware CMP-DK* is not installed see the installation documents from the *Cmpware* web site at <http://www.cmpware.com/>.

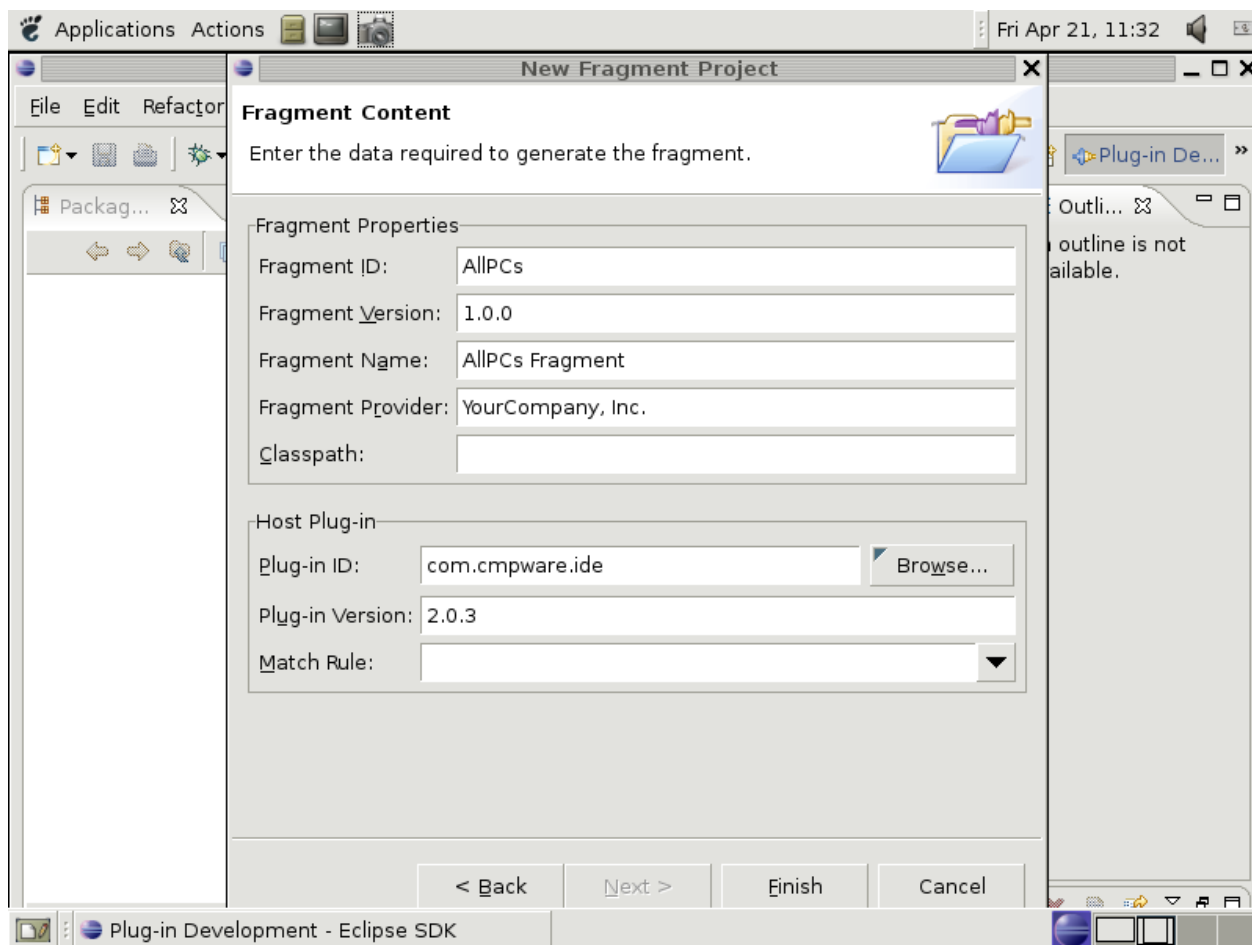


Figure 2: Initializing the Fragment project.

The first step in generating a new view for the *Cmpware CMP-DK* is to create a new Java project for code development. While any Java development environment may be used, it is most convenient to use the existing *Eclipse* IDE. It contains functionality that will assist in producing the final deployable version of the extension in later



implementation steps.

In general, the process for creating and deploying this new project is nearly identical to the procedure for creating and deploying any Java software for *Eclipse*. This document will cover all of the steps necessary to setup and produce the final deployable software, but the emphasis will be on the unique aspects of the system necessary to interact with the existing *Cmpware CMP-DK*. Other aspects of using the *Eclipse* system are more thoroughly documented in other places, in particular at the *Eclipse* web site at <http://www.eclipse.org/>

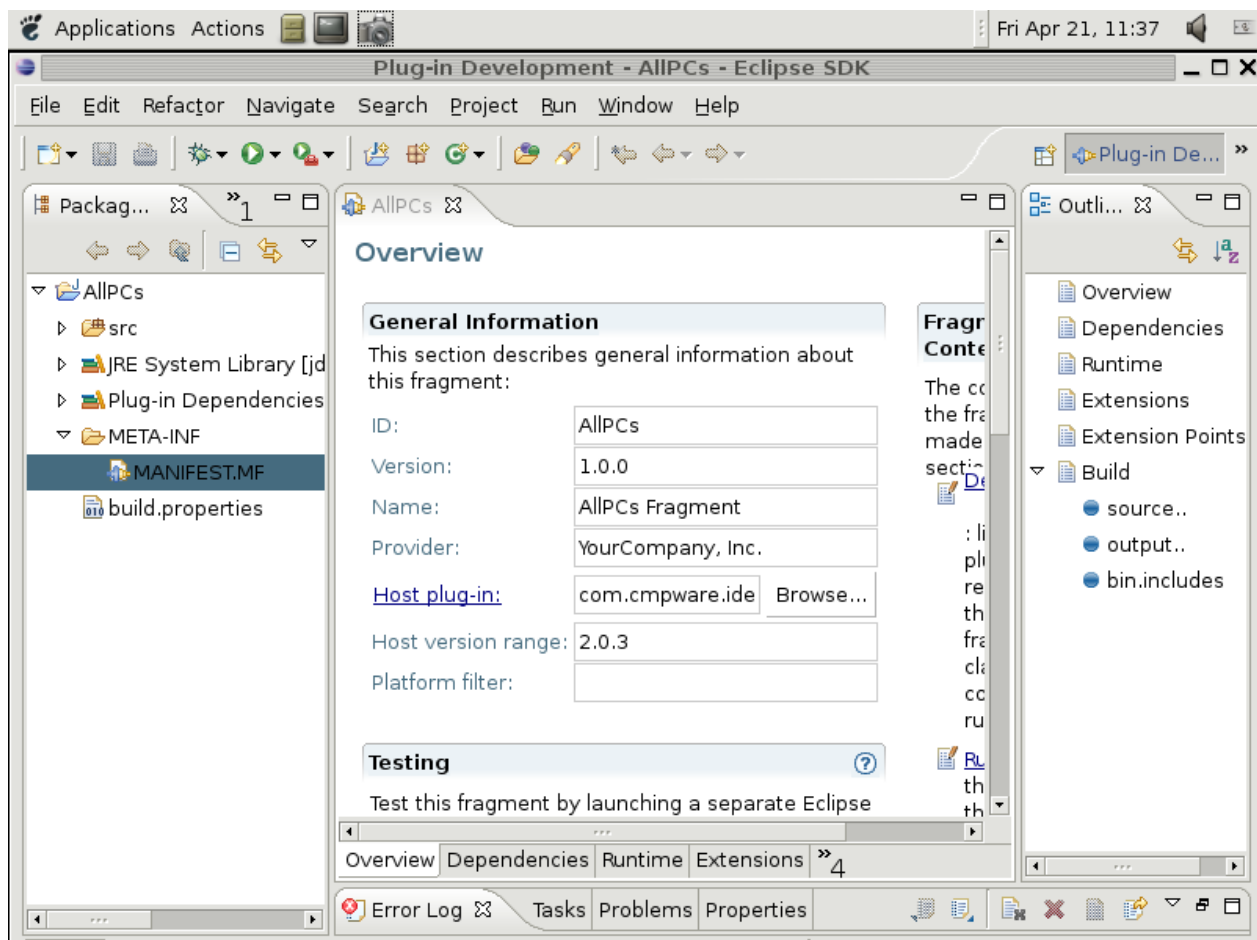


Figure 3: The Fragment project.

To begin, bring up *Eclipse*. If it does not default to the Java development IDE, you can select it with the **Window --> Open Perspective --> Java** menu item. Next create a



new project using the **File --> New --> Project ...** menu item. This brings up the Dialog box in Figure 1 which allows the selection of the type of project to be built. In this case, we are building a **Fragment Project** under **Plugin Development**.

It is important to select the **Fragment Project**. A *Fragment* is an *Eclipse* construct used specifically to extend an existing *Eclipse* plugin. A fragment takes some compiled Java code and a few files used for interfacing and control and performs the connection to the existing *Eclipse* code. There are other ways to accomplish this under Eclipse, but this example the *Fragment* approach is the simplest and most appropriate.

The next step in setting up the Fragment project is to specify some information requested in the dialog box in Figure 2. The offered defaults are all appropriate, and the blank field for the *Fragment Provider* in this example is set to **YourCompany, Inc.** This *Fragment Provider* name is used to demonstrate that the fragment can be completely decoupled from *Cmpware* developed software.

Additionally, the required *Plugin ID* field must be set to **com.cmpware.ide**. This *Plugin ID* is selected from the **[Browse...]** button and the **com.cmpware.ide** item should be selected from the menu. Note that the *Cmpware Plugin ID* must be selected. This is the linkage point which tells the extension about the *Cmpware CMP-DK* plugin. Using an incorrect value in this field will result in the *Fragment* not being able to access the functions of the *Cmpware CMP-DK*, which are essential for this example.

Pressing the **[Finish]** button completes the process and sets up the *Fragment* development environment. A dialog box may also ask if the *Plugin Development Perspective* should be opened. Since this is the default development IDE for *Fragments*, the **[Yes]** button should be selected. This produces the completed *AIIPCs* fragment project as in Figure 3. The parameters set during this initialization, as well as other parameters, can be set and modified in this window shown in Figure 3.

The next step is to input the Java source code necessary to implement the extension. Selecting the **File -> New -> Class** menu item will bring up a dialog box used to initialize a new Java source file. The source code can generally take any form, in multiple files and multiple classes in any number of packages. In this case there will be a single class called *AIIPCs* in a single file called *AIIPCs.java*. This file will be placed in the package **com.yourcompany.ide**, primarily to demonstrate that this *Fragment* can be kept distinct from the *Cmpware* code. Finally, pressing the **[Finish]** button will cause the new file to be brought up in a new Java source code text editing window. The contents of this window contain a 'skeleton' of the necessary portions of code used in the class.

*Appendix A* at the end of this document shows the Java source code used to



implement the *AllPCs* view. It should be entered into this text window exactly as in *Appendix A*. At this point, the code should compile and produce a Java "class" file.

## The ShowAllPCs Java Code

The *ShowAllPCs* Java source code is fairly small and simple considering the functionality provided. Much of it is, however, specific to either *Eclipse* or *Cmpware* and may require some explanation. The code does contain source code comments that may provide insights into the internal workings of this software.

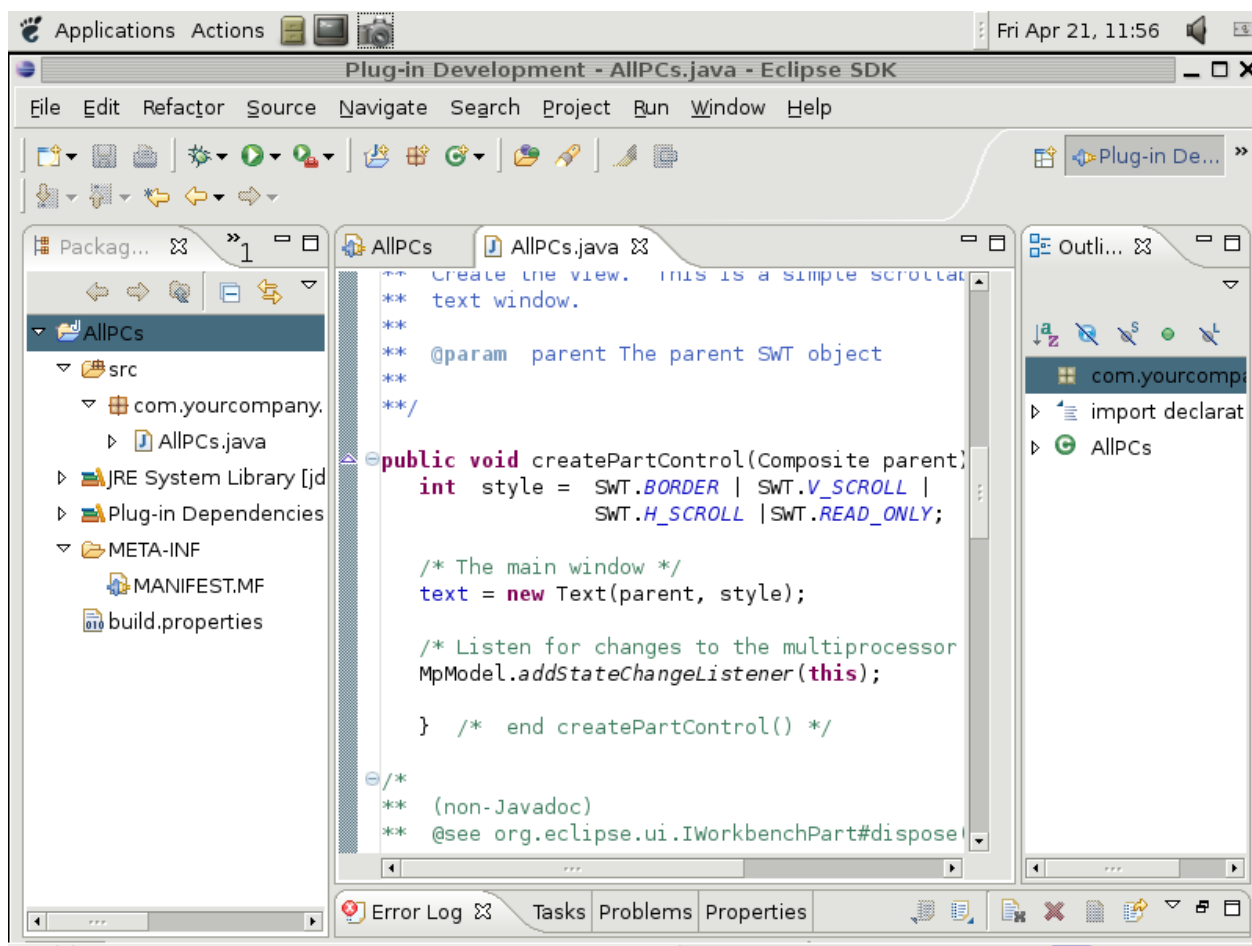


Figure 4: The AllPCs source code development environment.

Beginning from the top of the source code listing, the first thing to note is that *AllPCs* extends a class called *ViewPart* and implements another class called *IListener*.





The **viewPart** class is a component of *Eclipse* that implements the display of a window in the IDE, or a 'view' in *Eclipse* parlance. This requires that some particular methods be implemented for this class. Similarly, the **IListener** also requires that some methods be provided by this class, but **IListener** is provided by *Cmpware*. This is the event notification of change in the state of the multiprocessor model. In general, all displays in *Cmpware* update their information in response to a change in the multiprocessor model. The **IListener** interface provides this functionality.

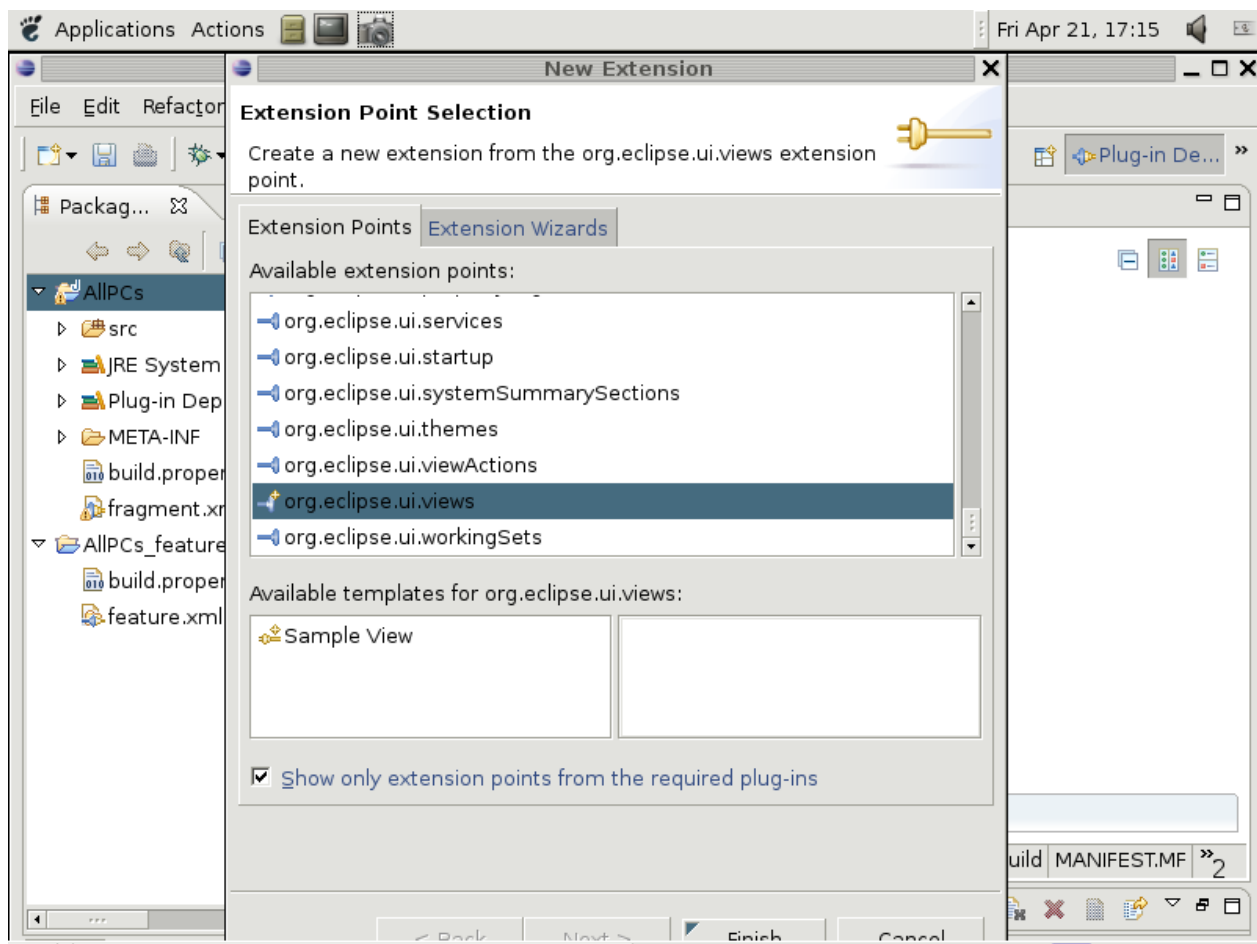


Figure 5: Adding the Extension Point.

The first method in the **AllPCs** class is the **createPartControl()**. This is used to set up the style and contents of the new window. In this example a simple **Text** window is used with horizontal and vertical scrolling. Note that it is important that this definition of graphical user interface elements be performed in this method. It is a





requirement of *Eclipse* and attempting to build or modify the graphical user interface structure in other parts of the code may result in errors.

This method also registers the **AllPCs** class ('**this**') with the multiprocessor event listener. This means that when the multiprocessor state changes, it will notify the **AllPCs** class. This is done by calling another method in this class, **handleEvent()**.

The next method in the **AllPCs** class is **dispose()**. This method is the opposite of **createPartControl()**. It is used to undo everything done in **createPartControl()**. *Eclipse* objects that are manually allocated or other 'housekeeping' chores are performed here.

The **setFocus()** method is also added as required by the **partControl** class, but it performs no particular function in this example and is left blank.

The last method is the **handleEvent()** method as required by **IListener**. This is where most of the work occurs. In this example, a simple text window is updated with both **Text.setText()** and **Text.append()** methods. This simply changes the text displayed in the window. The rest of this code uses methods from the *Cmpware* multiprocessor model, **MpModel**.

First, the size of the multiprocessor array is determined with the **getRows()** and **getCols()** methods from the multiprocessor model. Note that this is done on every update, since it is possible that the multiprocessor model may have changed between updates. Setting row and column parameters to some fixed values, even temporarily, can result in a run time error and should not be attempted. Getting such details correct are very important to producing a robust system.

Finally, the values of these Program Counters are are printed out in an array using the **Text.append()** method. This is all of the Java code necessary to add a new view to the *Cmpware CMP-DK*.

This point about coding errors should perhaps be emphasized further. While the code in **AllPCs** is completely distinct from both the *Cmpware* plugin and the *Eclipse* framework, it does have access to much of the internal machinery of both. This permits a wide variety of alternatives in designing the new extension, but also permits a wide variety of bugs. Illegal code, even in this extension, can have large impacts on the overall system. It is recommended that the designer of the extension take extra care in interactions with the other pieces of the system.



## Adding Extension Points

Once all of the Java code is written, one more piece of information called the *Extension Point* must be added to the project. *AllPCs* connects to the *Cmpware CMP-DK* and *Eclipse* via a well-defined and exported interface. This information must be passed on to the system so that it know how to configure and dynamically link the new extension into the existing code.

In this case, the *AllPCs* project main page has a tab called **Extensions**. The **[Add...]** button in this tab is pressed and it brings up the menu in Figure 5. From this list of extension points, scroll down and select `org.eclipse.ui.views`. This is the standard view interface used by *AllPCs* and *Eclipse*. Pressing the **[Finish]** button adds this extension point.

Now the point is added, but it must be customized. Right click on the new `org.eclipse.ui.views` point in the **All Extension** windows and select the **Add -> View** menu item. This brings up the dialog box in Figure 6. Most of the defaults are acceptable, but some of the blank fields must be filled in. These fields should have the following values:

● **ID:** The Extension ID is set to `com.yourcompany.ide.AllPCs`. This is a unique tag to identify the extension. In this case the full class name is used, since it is a unique identifier, although other values would work just as well.

● **Name:** The Extension Name is set to the string "Show All PCs" and is used for display purposes. This string will be displayed on the top of the *AllPCs* view window as well as in the selection list used to bring up the new window.

● **Class:** The *Extension Class* is set to `com.yourcompany.ide.AllPCs`. This is the actual full Java class name for the *AllPCs* class. This is perhaps the most important item in this list. This is used to load and execute the code in the *AllPCs* class.

● **Category:** The *Extension Category* is set to `com.cmpware.ide`. This is used to group the view command with the other *Cmpware* menu items used to bring up a new window. In this case the *Cmpware CMP-DK* uses this category for its other views.

● **Icon:** The Extension Icon is set to `icons/Cmpware16x16.gif`. This defines the small icon in the upper left corner of the window. The *Cmpware* icon used here is the one in all of the *Cmpware* views. This could just as easily be any valid bitmap in the system including *Eclipse* standard icons or one created and packaged specially for this view.



● **fastViewRatio**: The *Extension fastViewRatio* is set to 33. This is a general guide telling how large the initial window should be. This is the value used by other *Cmpware CMP-DK* windows in the main window and indicates that this window should take up approximately 2/3 of the *Eclipse* display. This is only a default used in the absence of other windows established in the IDE.

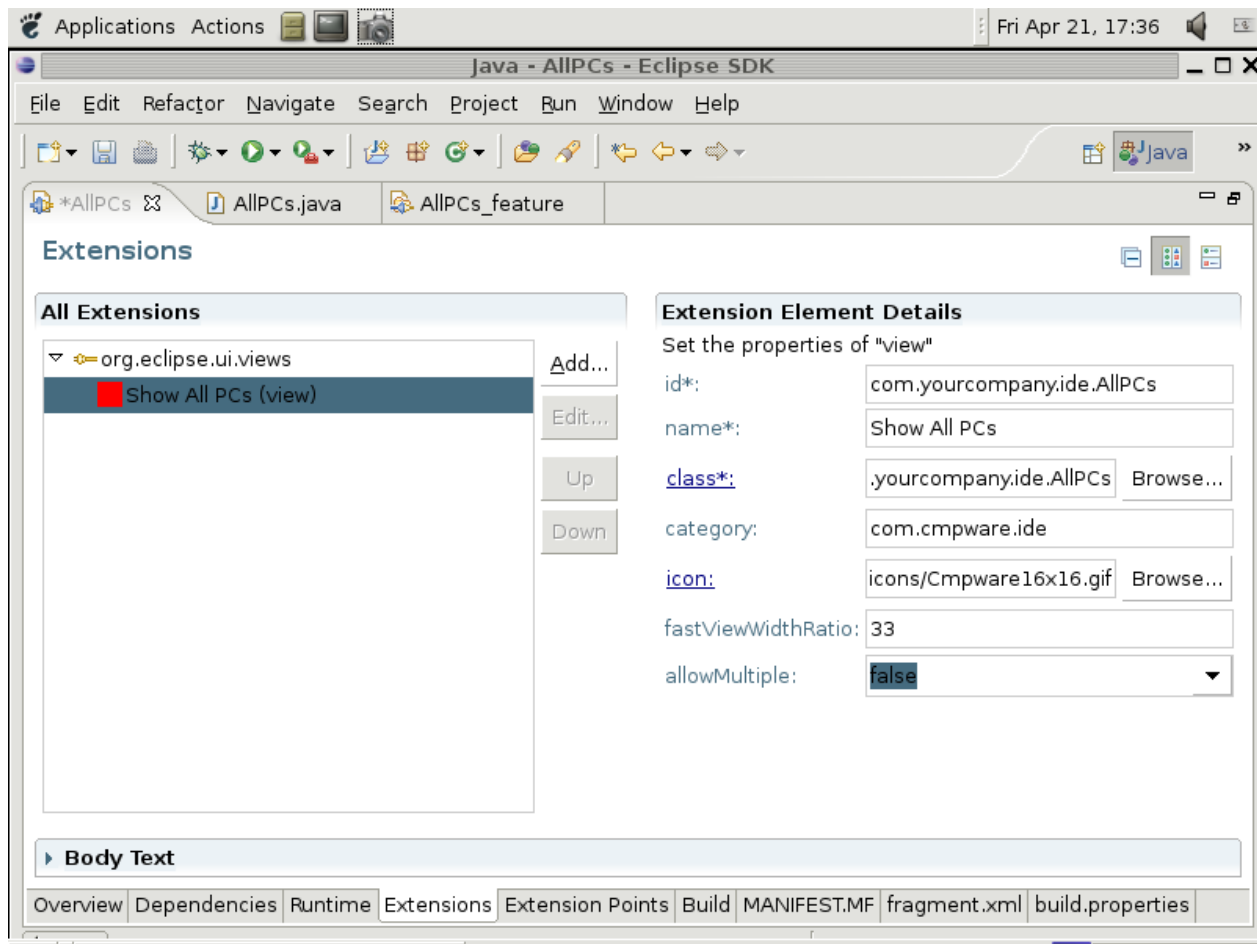


Figure 6: The values used by the AllPCs view extension point.

● **allowMultiple**: The *Extension allowMultiple* flag is a boolean value used to determine if multiple windows of this type may be opened. In this case it is set to `false`, indicating that only one of these windows should be open at a time. While there may be little harm in opening multiple *AllPCs* windows, the data will always be the same. And since all other *Cmpware CMP-DK* views do not allow multiple windows, this value will be used for consistency.



Once these value have been entered and the File **Save** button is pressed in the *Eclipse* IDE, a file called *fragment.xml* is produced. In fact, the entire point of filling in all of these fields is to produce this XML file, along with a manifest file named *MANIFEST.ML*.

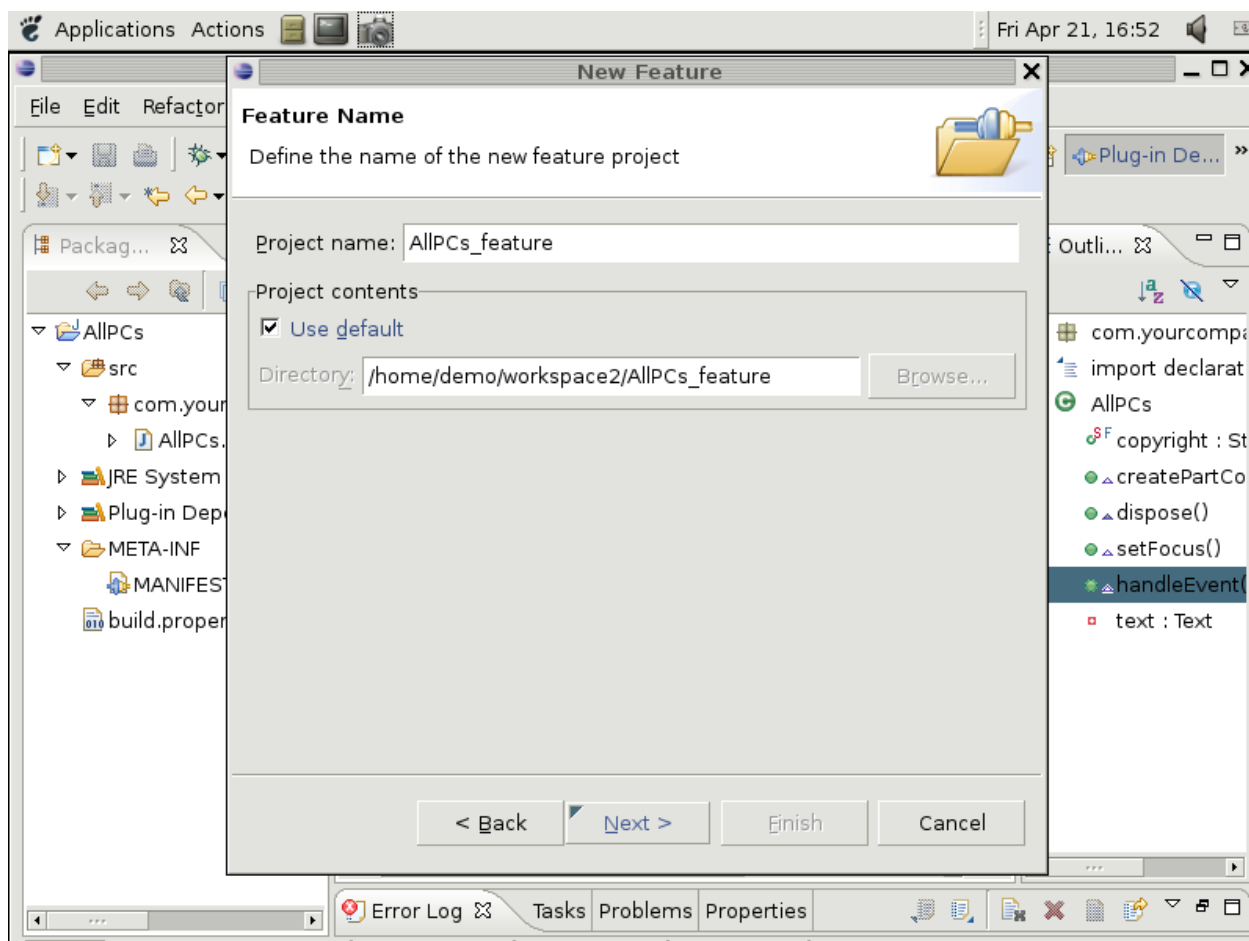


Figure 7: Making the new AllPCs feature.

These files are reproduced in Appendix B, and contains the fields just entered, but in an XML format. XML or manifest files that differ substantially from this one are probably incorrect and the values entered in the IDE should be checked. Also note that two values in this XML file, 'plugin-id' and 'plugin-version' were used in previous version of *Eclipse* but are no longer used. These fields may be included and left blank. This may produce a warning message, but this is harmless and can be ignored.



## The ShowAllPCs Feature

At this point, the *AllPCs* code and its linkage to the existing *Eclipse* and *Cmpware* code has been defined. The next step involves producing the files in a deployable format that can easily be installed by *Eclipse*. This is referred to in *Eclipse* as a *Feature*. While these *features* can be installed in a variety of ways, this example will package this *feature* into an *Eclipse Update Site* which will be placed at a pre-specified WWW URL. This then permits users to easily download and install this *feature* using the built in *Eclipse* 'update' mechanism. This also permits new version of the *feature* to be produced, permitting users to easily update to the new version.

It should be noted that there are other ways to deploy this type of fragment code, notably by manually copying files into directories in the main *Eclipse* distribution. But packaging this code into a feature makes it easier to distribute and easier for users to install. And while this process is documented elsewhere in *Eclipse* documents, it is often in the context of distribution of other *Eclipse* plugin code and often expects a broad knowledge of *Eclipse*. This document seeks to provide a step-by-step description of this process that will permit even novice *Eclipse* user to quickly distribute and install such extensions.

Packaging the *AllPCs* code into an *Eclipse* feature uses the same process as other *Eclipse* code, notably plugins, use. First, a new project must be set up. This is done with the **File -> New -> Project ...** menu item. This brings up a dialog box. Select the **Feature Project** under **Plugin Development**.

A project Name of **AllPCs\_feature** is given as in Figure 7. This fills in the fields in Figure 8 with the defaults shown. Additionally, the *Provider* field is modified to be **YourCompany, Inc.** Pressing the **[Finished]** button results in the project being initialized.

Once the Feature project is initialized, the code in the *AllPCs* project must be included in the feature. The tabs at the bottom of the new project should be viewed one by one and appropriate information included. The default tab, called **Overview** contains some fields that are filled in in this example. In this example the **Update site** is set to be **<http://www.cmpware.com/AllPCsUpdate/>** and the update site name set to **AllPCs Update Site**. This will eventually be the location of the Feature where it can be downloaded and installed by users. You may wish to use other values more appropriate to your system.

The next tab is the **Information** tab, which is itself a tabbed page. These contain copyright and licensing information. This can be filled in with any appropriate



information for the particular feature.

The **Plugins** tab is perhaps the most important part of this set-up. It should add the *AllPCs* fragment to this feature. Pressing the **[Add...]** button brings up the list of available plugins to add to this feature. The **AllPCs** fragment should be on this list and should be selected as in Figure 9.

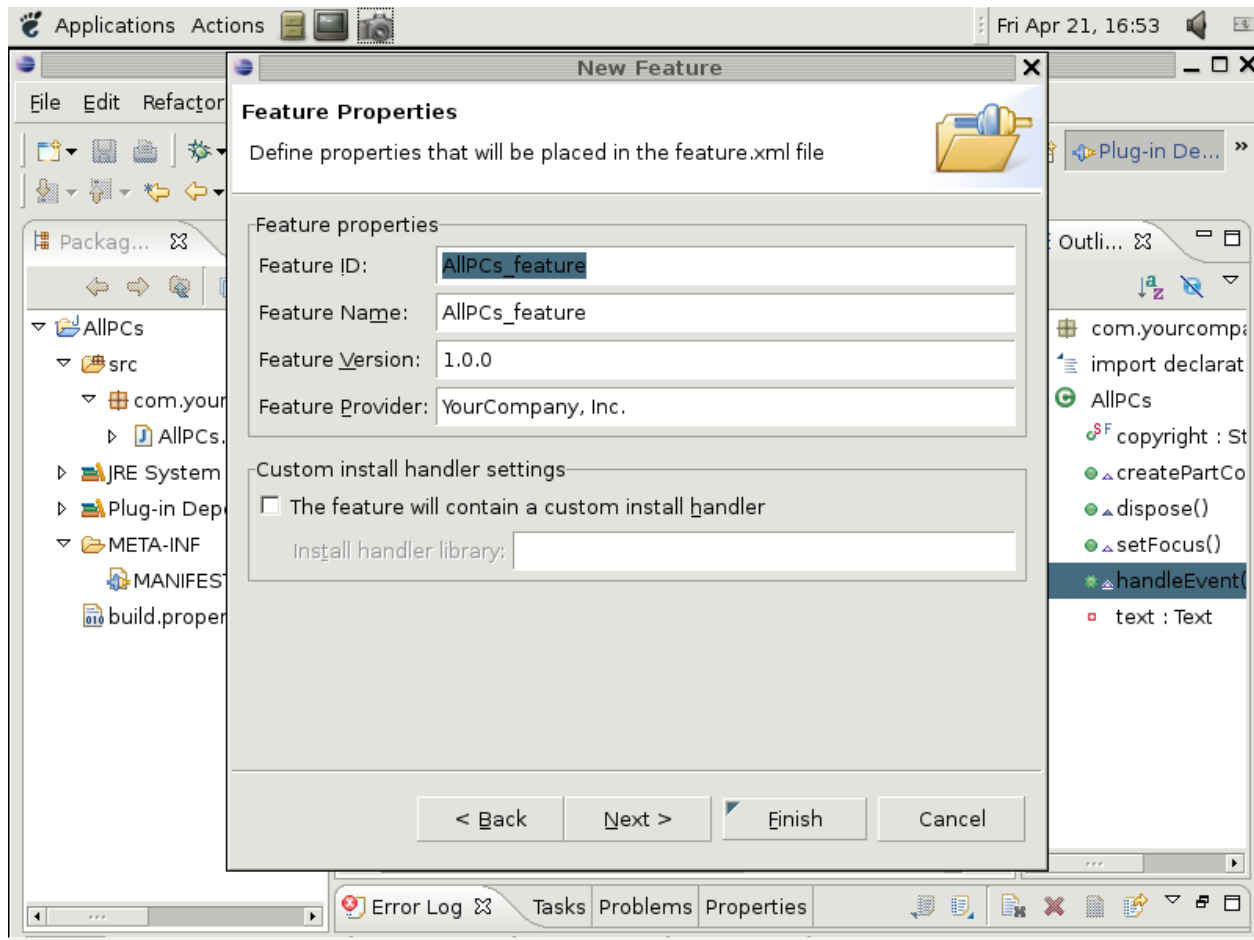


Figure 8: Initializing the new AllPCs feature.

As with the similar *fragment.xml*, the only real purpose of filling in these fields is to produce the file named *feature.xml*. This file is reproduced in Appendix C for reference. Your *feature.xml* file should not be substantially different from the one in Appendix C.



## The ShowAllPCs Update Site

Once a feature has been successfully built from the fragment, it can be packaged into an *Update Site*. An *Update Site* is simply a collection of files placed in a directory on a web site. Any *Eclipse IDE* can be pointed toward the URL of this site and can use the *Update Site* information to download and install new software in the form of features. In this case the feature will be the *AllPCs* fragment, and it will be used to provide a new view in the *Cmpware CMP-DK*.

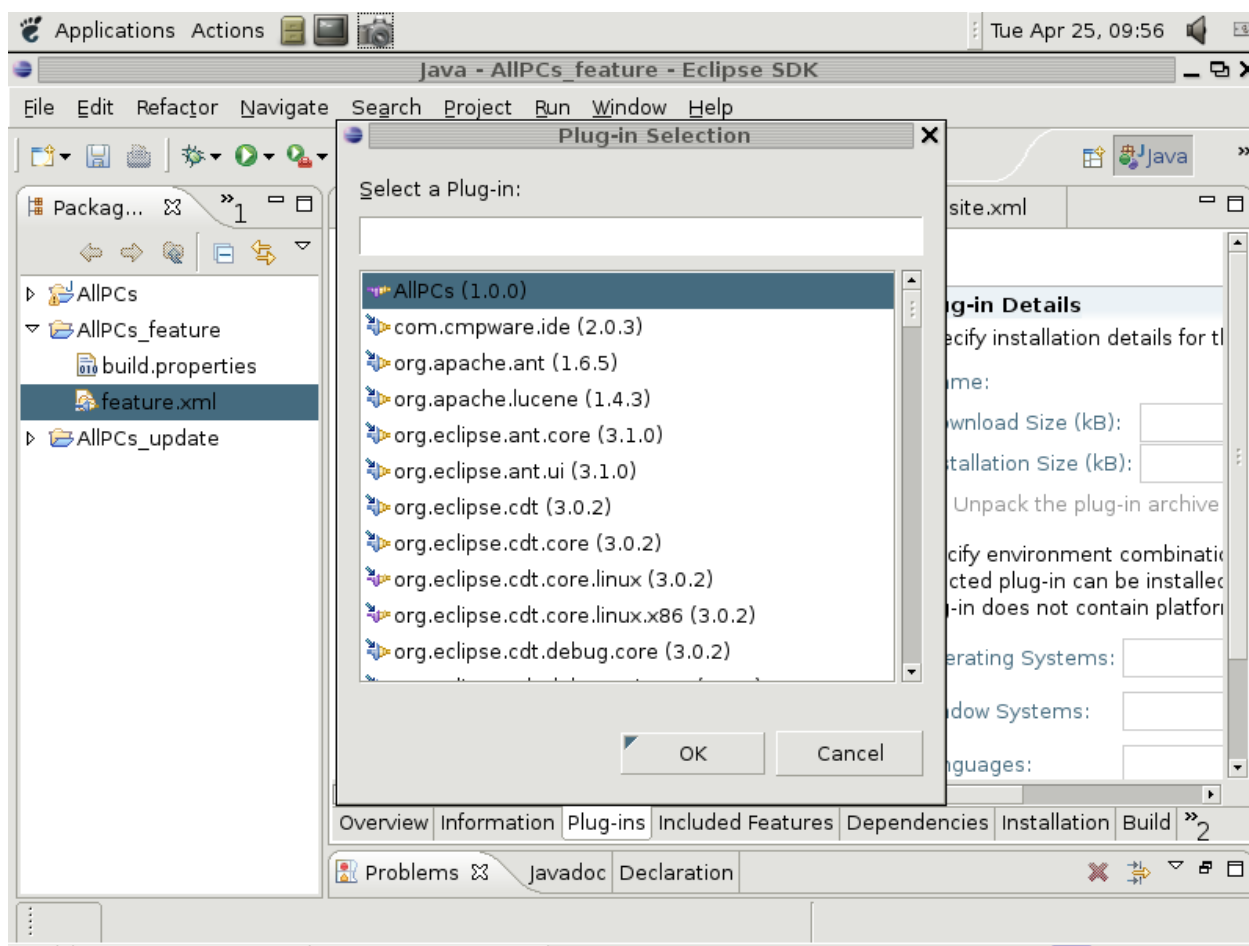


Figure 9: Adding the AllPCs plugin fragment to the new *AllPCs* feature.

As with the feature development, the Update Site is considered a new Project in Eclipse. To build a new update site from the *AllPCs* feature, begin by selecting the





menu items **File -> New -> Project...** . This will bring up the Dialog box in Figure 10. Select the **Update Site Project** under **Plugin Development**.

Clicking the **[Next >]** button brings up a dialog box which requests a *Project Name*. *AllPCs\_update* will be used in this example. Pressing the **[Finish]** button creates a file called *site.xml* and brings up the *Update Site Map*. In this *Update Site map*, press the **[Add Feature ...]** button and select the *AllPCs* feature. This will add the *AllPCs* feature to the Site Map.

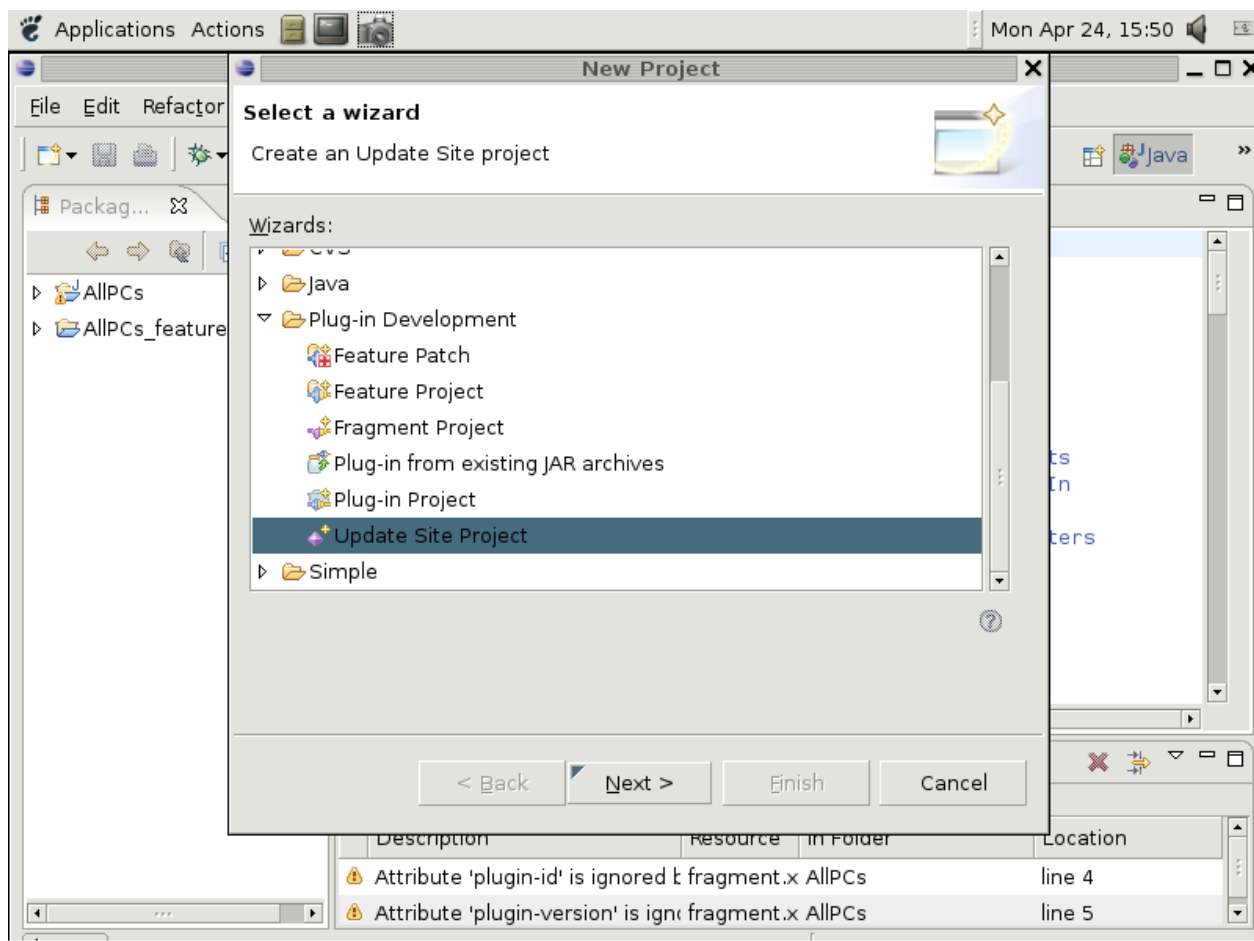


Figure 10: Building the Update site

At this point, all that needs to be done is to identify the web site to be used to contain this Update site. Clicking on the **Archives** tab bring up the Description and Archives page. Type in the URL of the web site location where this feature will be uploaded. In



this example, the site `http://www.cmpware.com/AllPCsUpdate/` is used. Of course, your version of this example will have to use the URL of a web site to which you have access to upload files. This demonstration will, however, be available at this *Cmpware* URL. Finally, some description of the update should be entered.

After adding the new feature to the update site, pressing the file **Save** button will update the *site.xml* file. Like the other XML files in the previous stages of development, this one contains control information. It is reproduced in Appendix D.

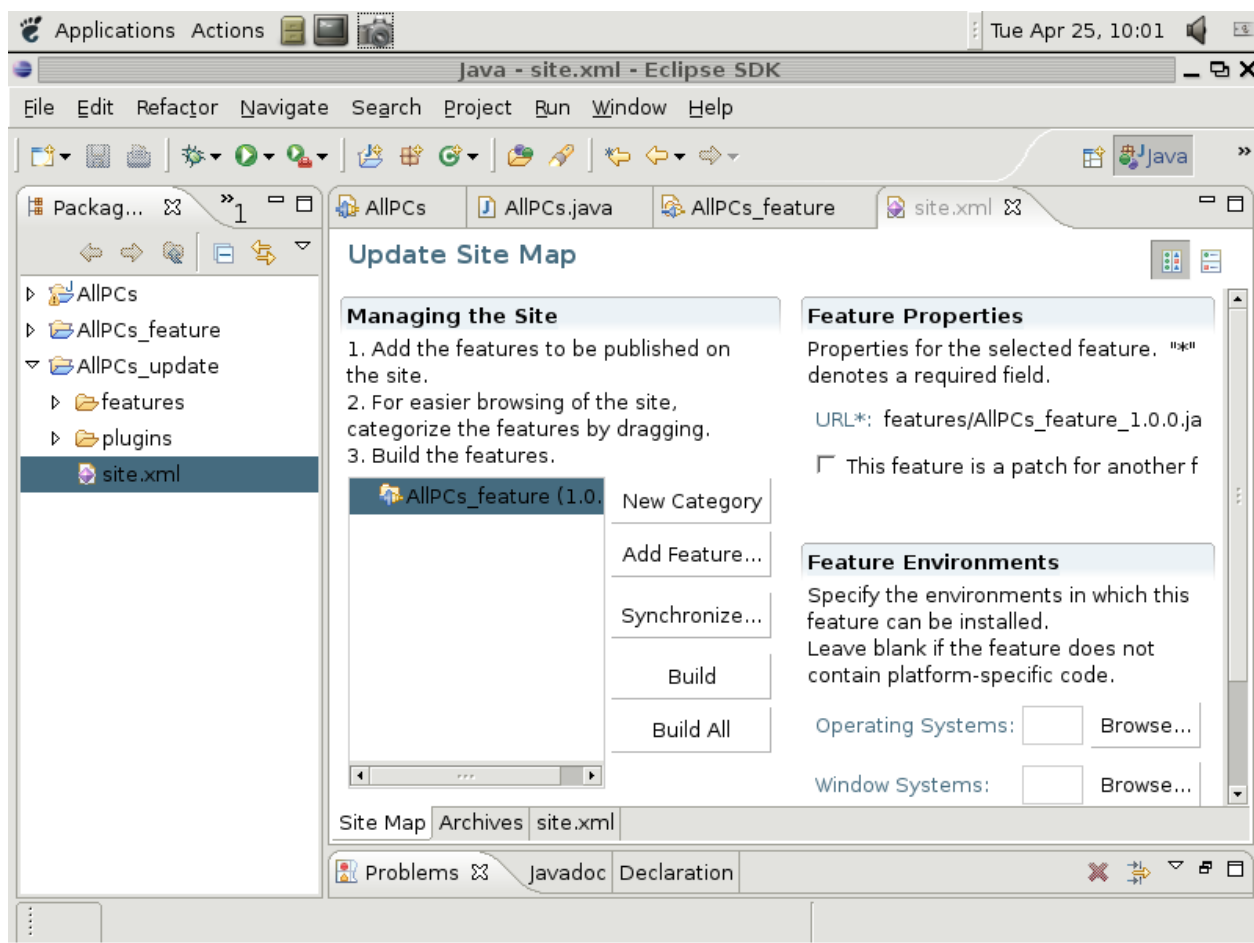


Figure 11: Adding the new Update Site.

Finally pressing the **[Build All]** button as in Figure 11 causes all of the code to be compiled and packaged as necessary for the update site. Specifically, two directories named `features` and `plugins` are created, each containing a JAR file. JAR files are



"*Java ARchive*" files and contain various data used to execute Java software. Both the `site.xml` and these two directories and their contents must now be uploaded to the exact URL of the update site.

## Installing the ShowAllPCs Update

Once all of the files have been uploaded to the update web site, they may be downloaded, installed and run by any user of the *Cmpware CMP-DK* that has access to this web site. This process is very simple and is well-document elsewhere but a simple overview is provided here for convenience. See the *Eclipse* web site

<http://www.eclipse.org/> or other *Eclipse* documentation for more information on installing new *Eclipse* functionality from an Update Site.

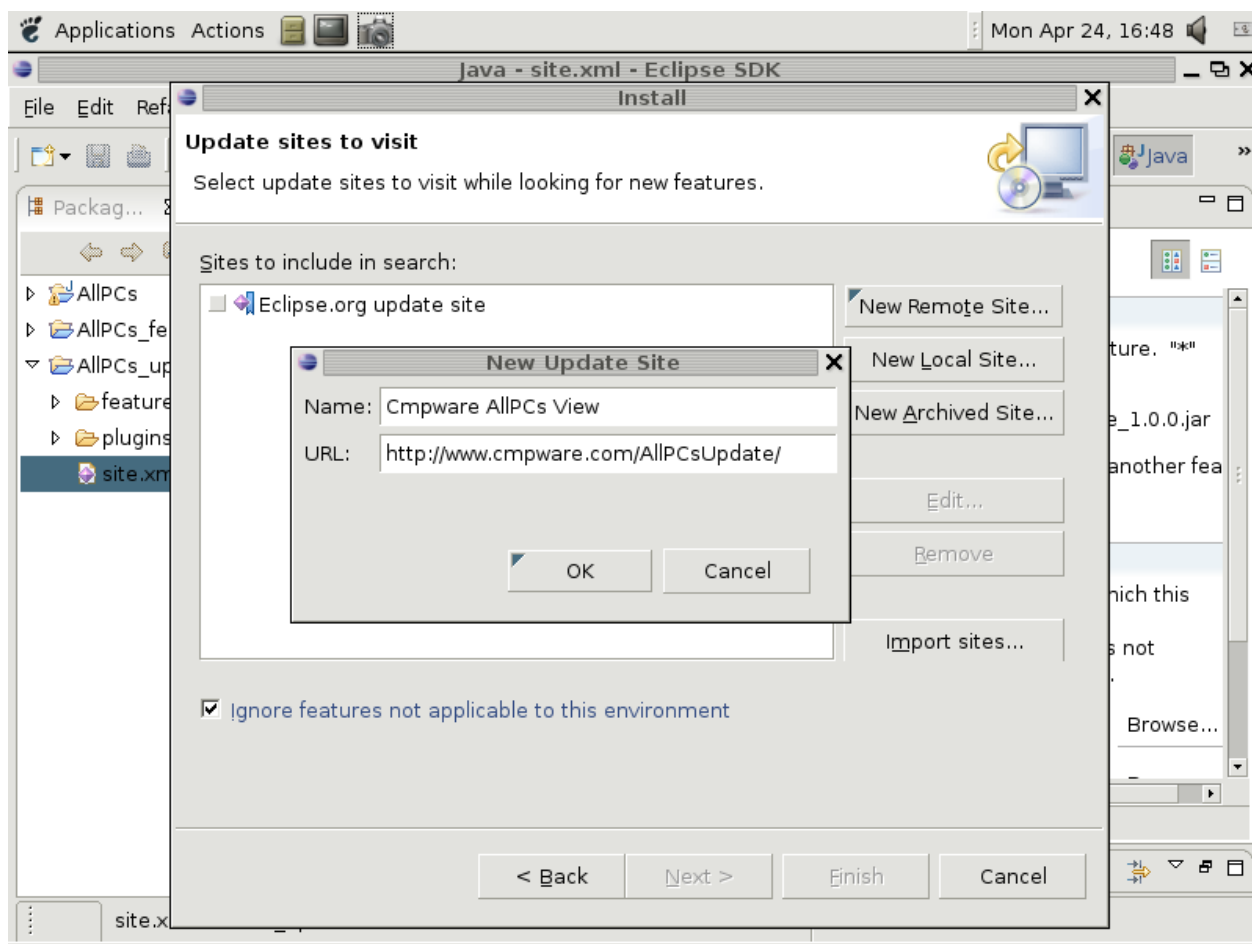


Figure 12: Adding the new Update Site.



From the main *Eclipse* menu, select the **Help -> Software Updates -> Find and Install ...** item. This brings up a large dialog box with two choices, *Search for updates of currently installed features* and *Search for new features to install*. Select the second option, since this is a new feature in this *Eclipse* installation and press the **[Next >]** button. This will bring up another dialog box call *Install* which can be seen in Figure 12.

Press the **[New Remote Site ...]** button in this *Install* dialog box and fill in the *New Update Site* fields as in Figure 12. In this example, the Name is **Cmpware AllPCs View** and the URL is `http://www.cmpware.com/AllPCsUpdate/`

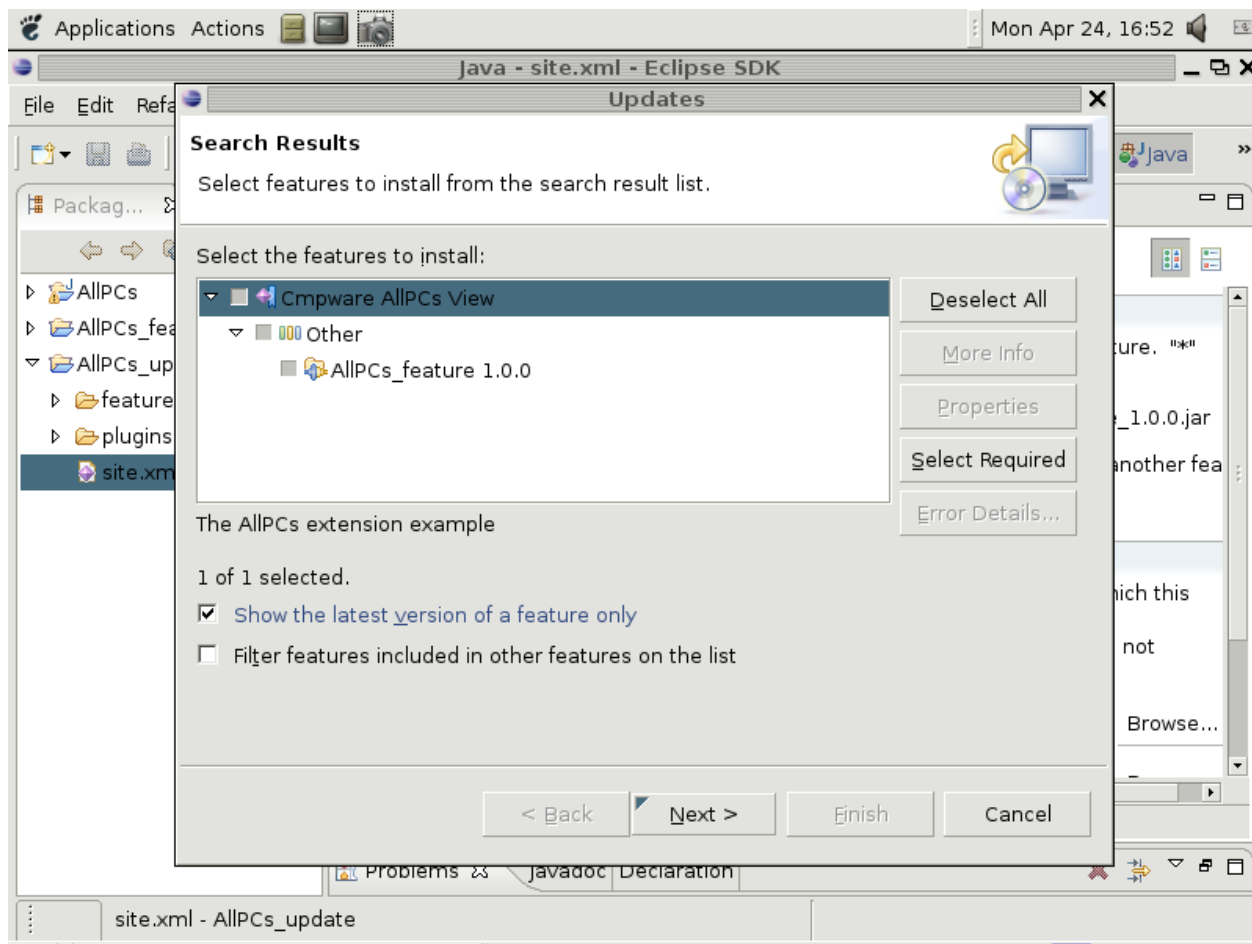


Figure 13: Selecting the AllPCs remote Update Site.

Pressing the **[Ok]** button adds this update site to the list of current update sites for this installation of *Eclipse*. The *Cmpware AllPCs View* should be selected by default. If it is



not, select it now and press the **[Finish]** button.

At this point yet another dialog box should be displayed in Figure 13. Select the *Cmpware All PCs View* again, but this time for download and installation. Pressing the **[Next]** button this time will begin the download and installation process. Subsequent dialog boxes may present a license agreement or other queries. Accept these and as requested and the installation will proceed.

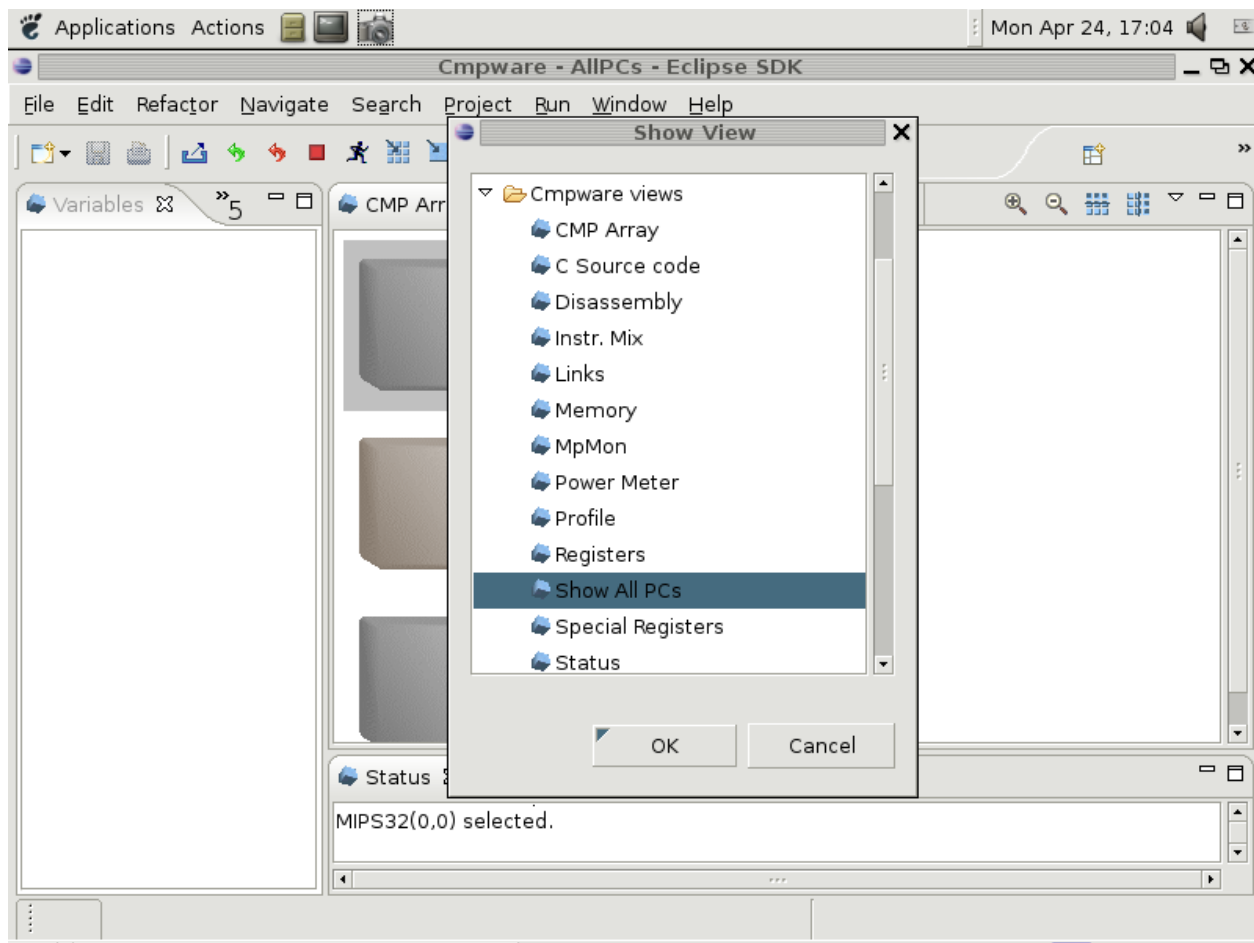


Figure 14: Opening the *Show All PCs* view in the *Cmpware CMP-DK*.

Also note that installing a new feature such as this one may require special access. Minimally if this is to be installed as a part of a shared *Eclipse* installation, the installer must have file write privileges for that directory. Otherwise *Eclipse* may request the user enter a new installation directory for this feature. If this is only to be used by the



current user, it can be installed in that user's local directory. If it is to be shared, it may be best to run *Eclipse* as a more privileged user and install the new features under the standard *Eclipse* directory where it may be shared by all system users. This will be more of an issue in file systems such as Linux where file systems are partitioned by users and write protected. *Windows* based systems should not encounter such file protection issues during installation.

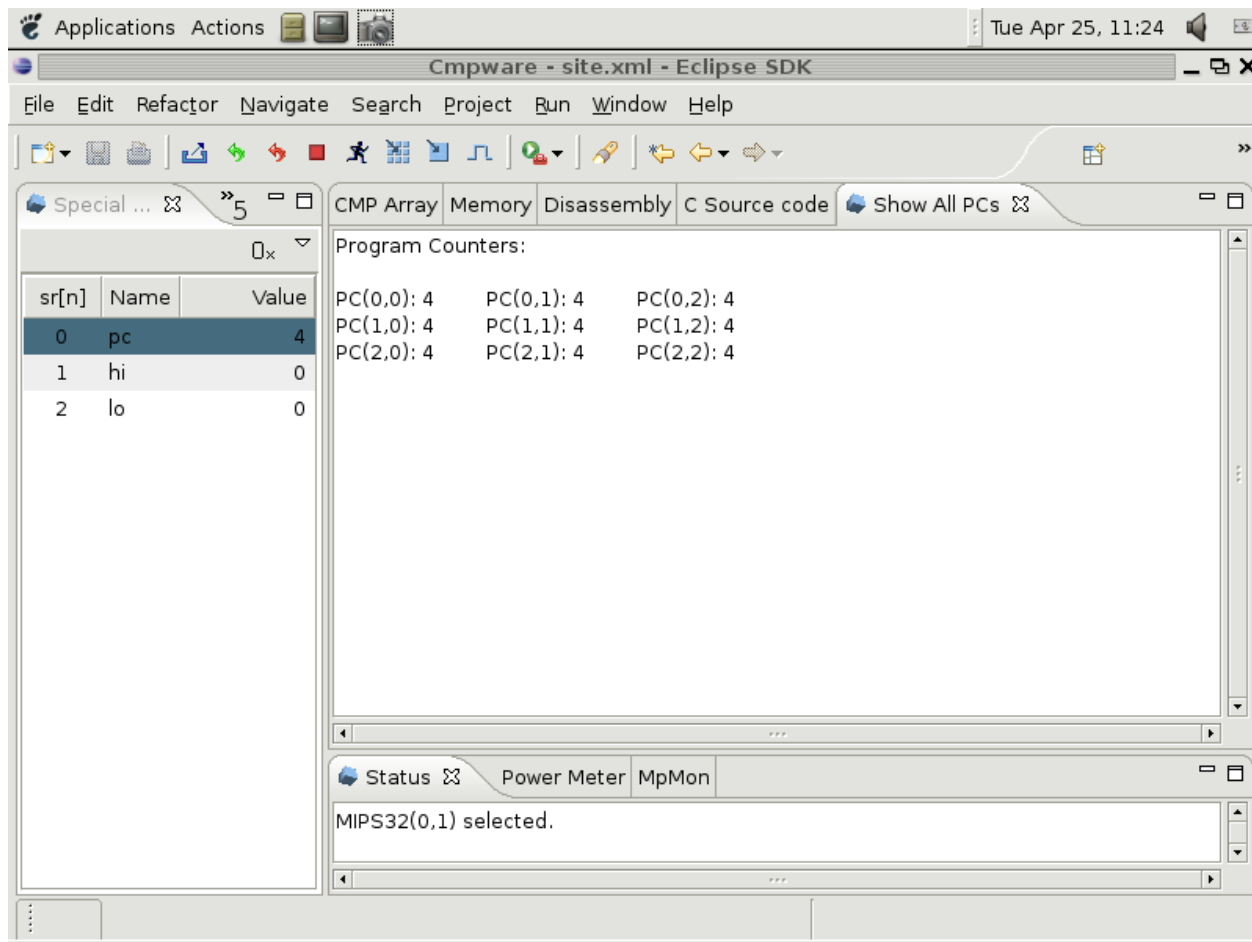


Figure 15: Running the *Show All PCs* view in the *Cmpware CMP-DK*.

Once the new feature is downloaded and installed, it may be necessary to restart *Eclipse*. A dialog box asking permission to restart is usually presented. Restarting the system gives access to the newly installed feature.



## Using the ShowAllPCs View

Once *Eclipse* has been restarted, open the *Cmpware CMP-DK* with the menu item **Window -> Open Perspective -> Other ...** and select the **Cmpware** item. This will bring up the default *Cmpware CMP-DK*. The new *AllPCs* view will not be available by default, but it should be listed under the available views for the *Cmpware CMP-DK*.

The new view can be found with the menu item **Window -> Show View -> Other...**. This brings up the dialog box with all of the available views in the current *Eclipse* installation. Under the *Cmpware* views, select the **Show All PCs** item as in Figure 14.

This will open the *Show All PCs* view, but it may be tabbed in one of the three panes in the *Cmpware CMP-DK*. Double clicking on the tab expands this view to full screen, or 'dragging' the tab can move the view to another collection of tabbed views.

The *Show All PCs* view is blank at first, but stepping the clock in the simulation will update the display. Figure 15 shows the new display with all of the updated values for the Program Counters for each of the processors in the simulation model. Each time the clock is stepped, all of the Program Counters in the display will change in unison. This example just uses the default simulation configuration. Processors running actual user code or a different multiprocessor model would give different results.

It is interesting to note that even though the types of processor models may vary and the number and type are not known when the code is written, that the new display is able to successfully display the data. This is one of the strong points of the *Cmpware CMP-DK* design. Because processors are defined as standard and interchangeable units, they can be accessed and manipulated uniformly. This makes creating extensions such as *Show All PCs* simpler to implement while also making them applicable to a wide range of simulation models.

## Conclusions

The *Cmpware CMP-DK* permits end users to 'extend' its functionality and modify and enhance its behavior. Perhaps the simplest and most useful way to extend the *Cmpware CMP-DK* is to add a new window, or view. This view can be used to display any data desired. Typically this would be data unique to a new architecture that may not be supported by the default values, or it may be some new and different way of providing information about the multiprocessor system.

This extension facility permits functionality and value to be added to the *Cmpware CMP-DK* without having to use the existing source code or even the source code of





*Eclipse*. Complete stand-alone functionality can be provided in an easily deployable standard format.

The process used to produce these extension is relatively simple. Java code is written to provide the new functionality. In the example used in this document, a very small number of lines of Java code produce a completely new multiprocessor data display.

Once this Java code is written, it can be packaged as a standard *Eclipse* feature and then further packaged into an *Eclipse Update Site* for simple, widespread deployment. This gives all users of the *Cmpware CMP-DK* the ability to download and install this new functionality. This also permits new versions of these features to be updated and provided to all users. This permits an upgrade / update path to replace 3rd party code with newer enhanced versions. Such functionality is often unavailable in other, similar systems.

For more information on the commercial version of the *Cmpware CMP-DK* see our web site at:

[\*http://www.cmpware.com/\*](http://www.cmpware.com/)

or send an email to:

[\*info@cmpware.com\*](mailto:info@cmpware.com)



## Appendix A: AllPCs.java Source Code

```
package com.yourcompany.ide;

import org.eclipse.ui.part.ViewPart;

import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Text;
import org.eclipse.swt.SWT;

import com.cmpware.ide.IListener;
import com.cmpware.ide.MpModel;
import com.cmpware.cmp.Processor;

/**
 * This class demonstrates the use of plugin fragments
 * to add new functionality to an existing plugin. In
 * Particular this fragment produces a View in the
 * Cmpware CMP-DK displaying all of the Program Counters
 * (PCs) in the multiprocessor in a single window.
 * Copyright (c) 2006 by Cmpware, Inc.
 */

public class AllPCs extends ViewPart implements IListener {

    /** Copyright string */
    public final static String copyright =
        "Copyright (c) 2006 Cmpware, Inc. All Rights Reserved.";

    /**
     * Create the view. This is a simple scrollable
     * text window.
     *
     * @param parent The parent SWT object
     */

    public void createPartControl(Composite parent) {
        int style = SWT.BORDER | SWT.V_SCROLL |
            SWT.H_SCROLL | SWT.READ_ONLY;

        /* The main window */
        text = new Text(parent, style);
    }
}
```



```
/* Listen for changes to the multiprocessor model state */
MpModel.addStateChangeListener(this);

} /* end createPartControl() */

/*
** (non-Javadoc)
** @see org.eclipse.ui.IWorkbenchPart#dispose()
** */

public void dispose() {
    MpModel.removeStateChangeListener(this);
    super.dispose();
} /* end dispose() */

/*
** (non-Javadoc)
** @see org.eclipse.ui.IWorkbenchPart#setFocus()
** */

public void setFocus() {}

/*
** This is called every time the Multiprocessor Model
** changes state. This is most typically after the
** end of a simulation cycle.
**
** (non-Javadoc)
** @see com.cmpware.ide.IListener#handleEvent()
** */

public void handleEvent(int eventType) {
    int i;
    int j;
    int pc = 0;
    int rows;
    int cols;
    Processor p;

    text.setText("Program Counters:\n\n");

    /* Get the current multiprocessor size */
    rows = MpModel.getMultiprocessor().getRows();
    cols = MpModel.getMultiprocessor().getCols();

    for (i=0; i<rows; i++) {

        for (j=0; j<cols; j++) {
```



```
/* Get a processor */
p = MpModel.getMultiprocessor().get(i, j);

/* Get the processor PC */
pc = p.getPC();

/* Print out the PC value */
text.append("PC("+i+", "+j+"): "+ Integer.toHexString(pc)+" \t");

} /* end for(j) */

text.append("\n");

} /* end for(i) */

} /* end handleEvent() */

/** The Text Widget for output */
private Text text = null;

} /* end class AllPCs */
```



## Appendix B: fragment.xml / MANIFEST.MF Source Code

Fragment.xml:

```
<?xml version="1.0" encoding="UTF-8" ?>
<?eclipse version="3.0"?>
<fragment
  plugin-id=""
  plugin-version="">
  <extension
    id="com.yourcompany.ide"
    name="AllPCs"
    point="org.eclipse.ui.views">
    <view
      allowMultiple="false"
      category="com.cmpware.ide"
      class="com.yourcompany.ide.AllPCs"
      fastViewWidthRatio="33"
      icon="icons/Cmpware16x16.gif"
      id="com.yourcompany.ide.AllPCs"
      name="Show All PCs" />
    </extension>
  </fragment>
```

MANIFEST.MF:

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: AllPCs Fragment
Bundle-SymbolicName: AllPCs; singleton:=true
Bundle-Version: 2.0.3
Bundle-Vendor: Cmpware, Inc.
Fragment-Host: com.cmpware.ide;bundle-version="2.0.3"
Bundle-Localization: plugin
Export-Package: com.yourcompany.ide
```



## Appendix C: feature.xml Source Code

```
<?xml version="1.0" encoding="UTF-8" ?>
<feature id="AllPCs_feature" label="AllPCs_feature" version="1.0.0" provider-
    name="YourCompany, Inc.">
    <description>The All PCs Extension View</description>
    <copyright>Copyright (c) 2006 by Cmpware, Inc.</copyright>
    <license url="http://www.cmpware.com/Docs/EULA.pdf">See the Cmpware CMP-DK
        license.</license>
    <url>
        <update label="AllPCs Update Site"
            url="http://www.cmpware.com/AllPCsUpdate/" />
    </url>
    <plugin id="AllPCs" download-size="0" install-size="0" version="0.0.0"
        fragment="true" unpack="false" />
</feature>
```



## Appendix D: site.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<site>
  <description url="http://www.cmpware.com/AllPCsUpdate/">The AllPCs
    extension example</description>
  <feature url="features/AllPCs_feature_1.0.0.jar" id="AllPCs_feature"
    version="1.0.0" />
</site>
```

