Common Navigator Framework

A Viewer provides the user with a view of objects using a single content provider, label provider, sorter and filter. The Common Navigator Framework (CNF) extends this idea by allowing a single view to dynamically use multiple and unrelated sets of content providers, label providers, sorters and filters. These can be activated in the view depending on declarative expressions or using API calls. The CNF implemented by the org.eclipse.ui.navigator plugin.

The CNF uses the idea of Navigator Content Extensions (NCE) which can refer to a content provider, label provider, sorter, or drag adaptor (note that filters are configured separately). An NCE has associated expressions that tell it when it is active. NCEs are also presented to the user in the view context menu so that user can turn them on or off in order to show the view in different ways. Examples of NCEs are a resource content extension that controls how resources are presented and Java content extension that shows Java projects. In the IDE's Project Explorer, you can turn off the Java content extension getting a pure resource view of the workspace.

By using the NCEs it's possible to include different model objects directly in the view. These model objects might be related in a cascading fashion. For example a resource object (an IFile) might refer to a Java class object. This Java class object can be presented directly in the view by defining an NCE. The Java class object can have children (representing methods, fields, etc) all of which appear in the view. You can then add another NCE to display model objects related to the Java (or resource object) that display content related to a Java Server Page object. This can all be done without the "lower level" NCEs (resources, Java) being aware of the existence of the "outer" NCEs (JSP).

The CNF has depends only on org.eclipse.ui and is therefore suitable for use both within the IDE and in an RCP application.

The CNF is highly configurable and many of its components can be used separately (though this is not the typical case). For example, it's possible use the service that manages content extensions with a viewer other than the standard CommonViewer.

The CNF documentation has the following sections:

- Configuration Overview - A high level discussion of the configuration.
- Operational Topics/Troubleshooting - A detailed discussion of various topics in the operation of the CNF along with some suggestions for troubleshooting.
- Step-by-step Instructions - Provides configuration instructions for various scenarios.
- Examples - Describes the example projects for the CNF.

Major Components of the Common Navigator Framework

The CNF has the following major parts:

1. View Part - CommonNavigator.
2. Viewer - CommonViewer the implementation of a viewer that is contained in the above view part. You can also use the CNF with your own viewer that can be displayed anywhere.
3. Navigator Content Extensions - org.eclipse.ui.navigator.navigatorContent This extension point is where you configure the collections of content you want to make visible.
4. Navigator Viewer - org.eclipse.ui.navigator.navigatorContent This extension point is used to declare the relationship between the NCEs and a particular viewer. Regular expressions may be used here to allow a viewer to dynamically detect NCEs that it does not know about beforehand. visible.
5. Resource Support - Resources are one of the main types of objects managed by the CNF. To support this the org.eclipse.ui.navigator.resources plug-in provides all of the necessary definitions. The Project Explorer is configured using these definitions.
Navigators - Where the CNF Fits

There are currently 3 major navigators in the Eclipse IDE.

1. **Project Explorer** - This is an instance of the CommonNavigator that is provided by the org.eclipse.ui.navigator.resources plugin. It provides a view of the workspace and has a large number of NCEs contributed for resources, Java, C, Web Tools, Data Tools, PHP, etc.

2. **Package Explorer** - Provided by the Java Development Tools (JDT) UI project, this provides a view of Java classes for the workspace. Generally speaking, the presentation of the Package Explorer and Project Explorer for Java objects is substantially similar. The JDT is not used for projects beyond Java.

3. **Navigator** - This view is an implementation of the now deprecated (as of 3.5) org.eclipse.ui.views.navigator.ResourceNavigator class provided in the org.eclipse.ui.ide plugin. It shows only the workspace resources and does not support extensibility to show other content. In a future release, the Navigator view will be provided by an implementation of the CommonNavigator.

We also provide [instructions](#) for migration from the ResourceNavigator to the CNF.

The only navigator intended for general purpose client use is the CNF, which can be used either by adding NCEs visible in the Project Explorer (if developing an IDE plugin), or using a separate instance of the CommonNavigator as a view.

Navigator Content Extensions (NCE)

The CNF allows you to include not only resources but any type of object. One typical use case is that of a resource (consider a Java file for example) which has several subordinate objects presented as tree nodes representing aspects of the class like imports, methods, etc. The CommonViewer (portion of the Common Navigator) will include the IResource object representing the Java file, and then as its children an object for each import, method, etc. The mechanism for defining this is a [navigator content extension](#).

Each content extension also provides a means of associating objects with appropriate icons, labels and menu items.

Content extensions are invoked based on an enablement expression which is defined using core expressions in the extension point. Enablement expressions are commonly defined by sensing the class of an object.

A priority may be associated with a content extension which helps to determine the order of invocation of the NCEs. It's possible that more than one content extension may be enabled for a given object and situation. Priorities are defined with words like “high” or “normal”. A high priority would be associated with an extension that defines the primary UI for frequently manipulating an object. A low priority would be given to relatively infrequent utility sorts of UI.

Content extensions may be activated and deactivated for a given view part in the UI using the [Customize View](#) menu item in the header menu associated with the view part. Common filters are also may also be activated and deactivated the same way.

Common Filters

Filters allow the user to specify which resources or objects to exclude. Like content extensions, filters may be specified centrally and shared.

Common Wizards

References to wizards for new/import/export may be defined. These are included in the appropriate popup menu based on the enablement.

Action Providers
Though the CNF provides a mechanism to work with actions, it is recommended that you instead use the [Workbench Commands](http://www.eclipse.org/eclipse/platform/doc.isv/guide/cnf.htm) instead of actions.

Action providers provide a means to configure the retargetable actions and programmatically configure the popup menu in a CNF viewer. These are useful for when you must perform a computation to determine which items are added to the menu, or to adjust the retargetable actions to ensure that the user keystrokes are handled properly (like for Cut/Copy/Paste).

**Drag and Drop Support**

A drop assistant may be associated with a content extension to provide handling of additional (non-standard) transfer types or validation in the event of a drop.

**Link with Editor Support**

The link helper extension point provides a flexible mechanism of determining the appropriate selection in the navigator when an editor is activated. Conversely, it also provides the appropriate editor to be activated when the selection changes.
Configuring the Common Navigator

This section defines the configuration at a conceptual level; some level of detail is omitted for clarity and conciseness. For the complete details see the extension point documentation or the Operational Topics/Troubleshooting section.

- org.eclipse.ui.navigator.viewer
- org.eclipse.ui.navigator.navigatorContent
- org.eclipse.ui.navigator.linkHelper

Viewer Configuration

As the CommonNavigator is a View, it is added using the org.eclipse.ui.views extension point. The CNF configuration aspects of the view instance are specified with a corresponding org.eclipse.ui.navigator.viewer extension point.

It is possible to use the CNF facilities for an arbitrary TreeViewer (could this be any StructuredViewer). In this case you still need to use this extension point, but it is used only to bind the CNF view information with the NCEs. You then programmatically (how) bind your Viewer with the NavigatorContentService.

The view may be associated with Navigator Content Extensions that define how its content is found and rendered. The entries here are used to look up NCEs, common filters and common wizards.

The view may also be associated with Action Providers that define code for programmatic updating and provision of actions or retargetable actions. The entries here are used to look up action providers.

The view may also be associated with Link Helpers that define the relationship between the selection in the view and the active editor.

Content extensions, common wizards, common filters, action providers and link helpers are bound to the view using an include/exclude mechanism and with the capability of pattern matching. This allows the actual content extensions (defined below) to be specified in a granular fashion such that views can select only those they actually need. The point of the exclude part of the mechanism is to exclude undesired items that were specified in the include statement. For example, the include statement could specify “com.mycompany.content.*” and exclude could remove the test content extensions by saying “com.mycompany.content.test.*”.

A view is always associated with a popup menu. One important part of the popup menu configuration at this level is the insertion points, which are the locations where menu items can be added depending on when and where the menu is shown. By default, a standard set of insertion points is provided (described in the extension point documentation). However, you can define your insertion points directly using popupMenus.

You can also direct the view to use a specific popup menu by id using popupMenuId, and you can indicate whether to ignore platform action contributions to the menu.

In the startup case, it is necessary to call some content extensions on the root node of the tree of the view in order to get the initial set of children. This set of content extensions is specified by setting the isRoot() property on the viewerContentBinding.

This defines:

1. viewerContentBinding - bind a Content Extension or Common Filter specified with the org.eclipse.ui.navigator.navigatorContent/navigatorContent extension point;
2. viewerActionBinding - bind action providers specified with the org.eclipse.ui.navigator.navigatorContent/actionProvider extension point;
3. popupMenus - defines the insertion points (standard places where menu items can be added) for the popup menu associated with the viewer. Mutually exclusive with popupMenuId.
4. popupMenuId - causes the view to use the menu defined by the org.eclipse.ui.popupMenus extension point. Mutually exclusive with popupMenus.
5. dragAssistent - points to code that may provide extra transfer types to be used when starting a drag from the viewer.
6. **options** - specifies options used to control the presentation of the view. For example these allow hiding of menus and buttons shown at the top of the view. Theses options are defined by INavigatorViewerDescriptor

**Navigator Content Extension (NCE)**

A navigator content extension, specified with org.eclipse.ui.navigator.navigatorContent/navigatorContent defines a named collection of attributes (content provider class, label provider class, icon, etc) to be enabled under certain conditions (typically in response to selection in the navigator). An example of a content extension is that for a resource. Content extensions are useful to describe other model objects that might be included in the view.

The definition of content extensions is separated from their association with a particular view instance allowing them to be shared and reused. Each content extension has an id, which is used to bind it with a view, and a display name which is presented to the user to allow the content extensions to be activated or deactivated by the user interface associated with the each view.

Content extensions may be active or not. This can be controlled by the user using the Customize View menu item of the navigator. Inactive content extensions are not considered when processing the view.

The content extension defines the following:

1. **triggerPoints** - specifies the conditions when this content extension is enabled based on a given object. You can specify tests with core expressions on the object for which the content extension should be enabled.
2. **possibleChildren** - like triggerPoints, but used for the case where the parent of the desired content extension is known, like in the drop case where we need to determine which content extension is required to handle the drop based on the drop target which is the parent of the eventual object that will be added to the tree.
3. **enablement** - specifies an enablement that is both a triggerPoints and possibleChildren.
4. **labelProvider** - an ILabelProvider provides the text to be shown in the view.
5. **contentProvider** - an ITreeContentProvider provides the means of getting the parent and child objects for the viewer. Other interfaces are possible, see the extension point documentation for these.
6. **descriptionProvider** - an IDescriptionProvider provides a description that is shown in the status bar.
7. **activeByDefault** - indicates this content extension should be made active in the default configuration of the workbench (e.g. a new workspace).
8. **priority** - used to determine which content extension to use in the event that multiple extensions are enabled (based on their enablement conditions).
9. **icon** - used to associate the object with a specific icon.
10. **providesSavables** - indicates the content provider provides saveables. If true the content provider must adapt to SavablesProvider.

**Common Filter**

A common filter, specified with org.eclipse.ui.navigator.navigatorContent/commonFilter defines a filter that is controllable by the user in the view. The filter definition contains an id, a description that describes the filter, a description that describes what is filtered out, and the conditions (using core expressions) that identify objects the filter suppresses.

The common filter is bound to the view using the viewerContentBinding element of the viewer configuration.

**Common Wizard**

A common filter, specified with org.eclipse.ui.navigator.navigatorContent/commonWizard defines wizard that is to be shown in the new, import, or export menu.

The common wizard is bound to the view using the viewerContentBinding element of the viewer configuration.
**Action Providers**

An action provider, specified with `org.eclipse.ui.navigator.navigatorContent/actionProvider` allows you to specify a class, subclassing `org.eclipse.ui.navigator.CommonActionProvider` that is invoked at right-click and selection time to allow you to contribute to the popup menu or to the action bars.

Action providers may be defined at the top level of the extension point, not associated with any content extension. These action providers are named and then bound to the CommonViewer using the `org.eclipse.ui.navigator.viewer/viewerActionBinding`.

Action providers may also be associated with a content extension, in which case they are active with the content extension. This is done by including the action provider inside of the `org.eclipse.ui.navigator.navigatorContent/navigatorContent` extension point.

**Link Helpers**

A link helper, specified with `org.eclipse.ui.navigator.linkHelper` allows you to control how the "Link with Editor" matches the selection in the navigator with an editor and matches an active editor with the selection in the navigator.
Detailed Configuration Topics

This section covers the operation of the CNF in detail.

Content Provider selection

Selecting the content provider is done by finding one or more NCEs associated with an object. The CNF is registered as a content provider a viewer and thus gets called at the content provider APIs in response to actions by the user on the viewer. In general, if the user is navigating by expanding in the viewer, the getElements() or getChildren() methods are called. However if the user has selected some object (like in an editor) and wishes to show it in the viewer, the getParent() method is used because the viewer needs to be able to figure out the part of the tree between the object and the content currently visible in the viewer.

When selecting an NCE in reaction to the getElements() or getChildren() call on the viewer, the triggerPoints (or enablement if specified) expression is evaluated against the object. The content provider associated with that NCE is invoked. If there are multiple content providers enabled by their triggerPoints (or enablement) expressions, they are all invoked in order according to priority and their results are concatenated.

When selecting an NCE in reaction to the getParent call on the viewer, the possibleChildren (or enablement if specified) expression is evaluated against the object. The content provider associated with that NCE is invoked. If there are multiple content providers enabled by their possibleChildren (or enablement) expressions, the content providers are invoked in priority order. The first non-null parent returned by the content provider is used.

The overrides element allows an NCE to be overridden by another. In this case the content provider associated with the suppressed NCE will not be invoked to contribute.

Label Provider selection

TBS - talk about the association of the object with the NCE that is associated with the object's content provider.

More ???TBS

Pipelining

Pipelining works with the override mechanism to allow reshaping the tree of objects. To use pipelining service, your content provider must implement the IPipedlinedTreeContentProvider interface.

??? More TBS

Troubleshooting

Check Your Ids

This is the most common problem in configuring the CNF. Make sure the Ids match exactly what you expect them to.

Is it a Wildcard or Regular Expression?

Be sure that the includes and excludes Ids for the navigator content extensions or filters are actually regular expressions and not wild cards. So if you are looking for com.mycompany.myplugin* (this is a wildcard), make sure you use com.mycompany.myplugin.* (which is a matching regular expression). The former will find things like com.mycompany.mypluginnnnnn, but won't find com.mycompany.myplugin.function1.

Tracing

The CNF provides tracing that can be helpful to resolve problems. To enable the tracing go to the Tracing tab in the Launch Configuration Dialog and select Enable Tracing. Then check the org.eclipse.ui.navigator
plugin, and on the right check debug and also one of the following:

- **debug/startup** - Shows the processing of the configuration.
- **debug/dnd** - Shows all aspects of drag and drop processing. an object to its navigator content extension.
- **debug/resolution** - Shows the resolution of an object to its navigator content extension.

If you require support, include the tracing when you report your problem.
Using the Common Navigator with Resources

To create a Common Navigator view that manipulates workspace resources (in the IDE) use the steps below. These steps can also be applied to an RCP application, but you will also need to do some additional work. You don’t need to use any code at all to do this.

The example plugin org.eclipse.ui.examples.navigators.resources shows this.

1. Add the following as dependent plug-ins:
   a. org.eclipse.ui.navigator
   b. org.eclipse.ui.navigator.resources

2. Add a View extension (org.eclipse.ui.views) which uses the class org.eclipse.ui.navigator.CommonNavigator.

   <extension
     point="org.eclipse.ui.views">
     <view
       name="View"
       class="org.eclipse.ui.navigator.CommonNavigator"
       id="example.view">
     </view>
   </extension>

3. Add a org.eclipse.ui.navigator.viewer extension that has:
   o viewerActionBinding, point this to your View Id above (example.view)
     ■ includes of org.eclipse.ui.navigator.resources
   o viewerContentBinding, point this to your View Id above (example.view)
     ■ includes of:
       ■ org.eclipse.ui.navigator.resources
       ■ org.eclipse.ui.navigator.resourceContent
       ■ org.eclipse.ui.navigator.resources.*

   <extension
     point="org.eclipse.ui.navigator.viewer">
     <viewerActionBinding
       viewerId="example.view">
       <includes>
         <actionExtension pattern="org.eclipse.ui.navigator.resources.*" />
       </includes>
     </viewerActionBinding>
     <viewerContentBinding
       viewerId="example.view">
       <includes>
         <contentExtension pattern="org.eclipse.ui.navigator.resourceContent" />
         <contentExtension pattern="org.eclipse.ui.navigator.resources.filters.*" />
       </includes>
     </viewerContentBinding>
   </extension>
Using the Common Navigator in an RCP Application

To add the Common Navigator to an RCP application and have it manipulate the workspace resources, first follow the **steps** and then follow these. If you don't want to have the CN to use resources, omit the resources steps.

1. Add the following as dependent plug-ins:
   a. org.eclipse.ui.navigator

2. Add a View extension (org.eclipse.ui.views) which uses the class org.eclipse.ui.navigator.CommonNavigator.

   ```xml
   <extension point="org.eclipse.ui.views">
     <view name="View"
           class="org.eclipse.ui.navigator.CommonNavigator"
           id="example.view">
     </view>
   </extension>
   ```

3. Update your perspective factory (IPerspectiveFactory) code to show the new View (this is necessary when adding any View):

   ```java
   public void createInitialLayout(IPageLayout layout) {;
       String editorArea = layout.getEditorArea();
       layout.setEditorAreaVisible(false);
       layout.setFixed(true);

       layout.addStandaloneView("example.view", true /* show title */, IP);
   }
   ```

   Note that for the moment you need to specify "true" to show title, otherwise the viewer will not render correctly ([bug 235171](https://bugs.eclipse.org/bugs/show_bug.cgi?id=235171)).

4. Add a org.eclipse.ui.navigator.viewer extension that has:
   o viewerContentBinding, point this to your View Id above (example.view)
     - includes of:
       - com.yourcompany.yourapp.yournav.*

   ```xml
   <extension point="org.eclipse.ui.navigator.viewer">
     <viewerContentBinding viewerId="example.view">
       <includes>
         <contentExtension pattern="com.yourcompany.yourapp.yournav.*" />
       </includes>
     </viewerContentBinding>
   </extension>
   ```

5. Add navigator content extensions as required:

6. If you need resources, add the following to your WorkbenchAdvisor
   a. To get the resource workspace as input, override this method:

   ```java
   public IAdaptable getDefaultPageInput() {
       IWorkspace workspace = ResourcesPlugin.getWorkspace();
       return workspace.getRoot();
   }
   ```
b. If you need resources, you need to have the workbench adapters correctly hooked up, so add this code to the initialize() method:

```java
public void initialize(IWorkbenchConfigurer configurer) {
    IDE.registerAdapters();
}
```
Migrating from the ResourceNavigator

First follow the steps to add the Common Navigator for resources. If your application is an RCP application, also do the RCP steps.

The following steps discuss the migration component for each part of the ResourceNavigator

1. org.eclipse.ui.ide.resourceFilters
   Add the corresponding org.eclipse.ui.navigator.navigatorContent/commonFilter extensions.

2. FrameList support get/setFrameList()
   The FrameList support is used for the Go Into functionality. Support for this is included in the ProjectExplorer subclass of CommonNavigator. You should subclass ProjectExplorer instead of CommonNavigator.

3. ResourcePatternFilter support get/setPatternFilter()
   Configure the ResourcePatternFilter using the org.eclipse.ui.navigator.navigatorContent/commonFilter extension. Then you can access the instance of the filter using the following code (where yourViewer is the instance of the CommonViewer and yourFilterId is the string of the Id of your common filter configured above:

   ```java
   INavigatorContentService contentService = yourViewer.getNavigatorContentService();
   INavigatorFilterService filterService = contentService.getFilterService();
   ICommonFilterDescriptor[] fds = filterService.getVisibleFilterDescriptors();
   for (int i = i; i < fds.length; i++) {
       if (fds[i].getId().equals(yourFilterId))
           return filterService.getViewerFilter(filterDescriptor);
   }
   ```

   Note: (???) possibly don't deprecate ResourcePatternFilter, or move this into org.eclipse.ui.navigator.resources as it is; this class does not depend on ResourceNavigator.

4. IWorkingSet support get/setWorkingSet()
   ??? figure out the corresponding ProjectExplorer way to do this.

5. ResourceSorter support get/setResourceSorter()
   ??? Similar to the filter service.

6. ResourceComparator support get/setResourceComparator()
   ??? Have to use ResourceSorter instead for now as the CN does not directly support ResourceComparators
Examples for the Common Navigator

This describes the example projects for the CNF.

Talk about how to install the examples, they are in the standard UI examples feature

Using the CNF with Resources and Model Objects

This example builds on the org.eclipse.ui.examples.navigator.resources to add support for model objects. The model objects are derived from the contents of a text editor containing standard Java properties. To use the example:

1. Run the example using the Navigator Properties1 launch configuration (note it's very important to use the launch configuration because it defines exactly the plugins required and we don't want to get the other example plugins).
2. Go to Window -> Show View and find the CNF Examples category (which may be under Other) and show the Properties Example View.
3. Create a project and then create a file with the suffix .properties Add some properties to the file, like "prop1=value1". Note that the file in the view can be expanded to show the properties. You can also use the context menu associated with each property to open or delete the property.

Adding More Model Objects

This example extends the above to add some additional model objects. If you look at the Navigator Content Extension point you can see how this is configured.