

CONTENTS INCLUDE:

- About NetBeans Platform
- Getting Started
- Main Benefits
- NetBeans Platform Modules
- NetBeans Platform APIs
- NetBeans Platform Gotchas and more...

Essential NetBeans Platform

By Heiko Böck, Anton Epple, Miloš Šilhánek, Andreas Stefik, Geertjan Wielenga, and Tom Wheeler

ABOUT NETBEANS PLATFORM

The NetBeans Platform is a generic framework for commercial and open source desktop Swing applications. It provides the “plumbing” that you would otherwise need to write yourself, such as the code for managing windows, connecting actions to menu items, and updating applications at runtime. The NetBeans Platform provides all of these out of the box on top of a reliable, flexible, and well-tested modular architecture.

In this refcard, you are introduced to the key concerns of the NetBeans Platform, so that you can save years of work when developing robust and extensible applications.

GETTING STARTED

To get started with the NetBeans Platform:

Approach	How to Get Started
IDE	Download NetBeans IDE, which includes NetBeans Platform development tools such as templates, wizards, and complete NetBeans Platform samples out of the box.
Maven	Use the Maven archetypes for NetBeans Platform development: NetBeans Platform archetype: - GroupId: org.codehaus.mojo.archetypes - ArtifactId: netbeans-platform-app-archetype NetBeans Module archetype: - GroupId: org.codehaus.mojo.archetypes - ArtifactId: nbm-archetype
Ant	Download the NetBeans Platform ZIP file, which includes a build harness. The build harness includes a long list of Ant targets for compiling, running, testing, and packaging NetBeans Platform applications.

Join the NetBeans Community mailing lists!

MAIN BENEFITS

The following are the main features of the NetBeans Platform, showing you the benefits of using it rather than your homegrown Swing framework.

Feature	Description
Module System	Modularity offers a solution to “JAR hell” by letting you organize code into strictly separated and versioned modules. Only modules that have explicitly declared dependencies on each other are able to use code from each other's exposed packages. This strict organization is of particular relevance to large applications developed by engineers in distributed environments, during the development as well as the maintenance of their shared codebase.
Lifecycle Management	Just as application servers such as GlassFish provide lifecycle services to web applications, the NetBeans runtime container provides services to Swing applications. Application servers understand how to compose web modules, EJB modules, and so on, into a single web application, just as the NetBeans runtime container understands how to compose NetBeans modules into a single Swing application.
Pluggability	End users of the application benefit because they are able to install modules into their running applications via an update center, since NetBeans modules can be installed, uninstalled, activated, and deactivated at runtime.
Service Infrastructure	The NetBeans Platform provides an infrastructure for registering and retrieving service implementations, enabling you to minimize direct dependencies between individual modules and enabling a loosely coupled architecture with high cohesion and low coupling.
File System	Unified API providing stream-oriented access to flat and hierarchical structures, such as disk-based files on local or remote servers, memory-based files and even XML documents.

Window System	Most serious applications need more than one window. Coding good interaction between multiple windows is not a trivial task. The NetBeans window system lets you maximize/minimize, dock/undock, and drag-and-drop windows, without you providing the code.
Standardized UI Toolkit	Swing is the standard UI toolkit and is the basis of all NetBeans Platform applications. A related benefit is that you can change look & feels very easily, and add internationalization and Java 2D effects to your applications
Generic Presentation Layer	With the NetBeans Platform, you're not constrained by one of the typical pain points in Swing: the JTree model is completely different than the JList model, even though they present the same data. Switching between them means rewriting the model. The NetBeans Nodes API provides a generic model for presenting your data. The NetBeans Explorer & Property Sheet API provides several advanced Swing components for displaying Nodes.
Advanced Swing Components	In addition to a window system, the NetBeans Platform provides many other UI-related components, such as a property sheet, a palette, wizards, complex Swing components for presenting data, a Plugin Manager, and an Output window.
JavaHelp Integration	The JavaHelp API is an integral part of the NetBeans Platform. You can create help sets in each of your modules and the NetBeans Platform will automatically resolve them into a single helpset. You can also bind help topics to UI components to create a context-sensitive help system for your application.

NETBEANS PLATFORM MODULES

The NetBeans Platform consists of a large set of modules. You do not need all of them. In fact, you only need 5. You also do not need to have a user interface, meaning that you can create server/console applications on the NetBeans Platform.

The complete list of NetBeans Platform modules is provided below. Items in red are mandatory, items in green are optional.

Module	Description
boot.jar core.jar org-openide-filestystems.jar org-openide-modules.jar org-openide-util.jar	Provides the runtime container.
org-netbeans-core.jar org-netbeans-core-execution.jar org-netbeans-core-ui.jar org-netbeans-core-windows.jar	Provides the basic UI components provided by the NetBeans Platform, together with related infrastructure.



NetBeans Platform 6.8 is the latest release of the NetBeans modular framework for Swing applications. Use this reliable and flexible application architecture to save years of development time and work. Develop your application on Ant or Maven; deploy it as a standalone application or via JNLP (web start). Discover why applications around the world are built on the NetBeans platform!

org-netbeans-core-output2.jar org-openide-io.jar	Provides an Output window for displaying processing messages. It also exposes an API that you can use to write to the window and change text colors.
org-netbeans-core-multiview.jar	Provides a framework for multi-tab windows, such as used by the Matisse GUI Builder in NetBeans IDE.
org-openide-windows.jar	Provides the API for accessing the window system.
org-netbeans-modules-autoupdate-services.jar org-netbeans-modules-autoupdate-ui.jar	Provides the Plugin Manager together with the functionality for accessing and processing update centers where NetBeans modules are stored.
org-netbeans-modules-favorites.jar	Provides a customizable window which can be used as a filechooser, enabling the user to select and open folders and files.
org-openide-actions.jar	Provides a number of configurable system actions, such as "Cut", "Copy", and "Paste".
org-openide-loaders.jar	Provides an API that lets an application recognize file types.
org-openide-nodes.jar org-openide-explorer.jar org-netbeans-swing-outline.jar	Provides the API for modeling business objects and displaying them to the user.
org-netbeans-modules-javahelp.jar	Provides the JavaHelp runtime library and enables JavaHelp sets from different modules to be merged into a single helpset.
org-netbeans-modules-mimelookup.jar org-netbeans-modules-editor-mimelookup.jar	Provides an API for discovery and creation of settings specific to file types.
org-netbeans-modules-masterfs.jar	Provides a central wrapper file system for your application.
org-netbeans-modules-options-api.jar	Provides an Options window for user customizations and an API for extending it.
org-netbeans-api-progress.jar org-openide-execution.jar org-netbeans-modules-progress-ui.jar	Provides support for asynchronous long running tasks and integration for long running tasks with the NetBeans Platform's progress bar.
org-netbeans-modules-queries.jar	Provides an API for getting information about files and an SPI for creating your own queries.
org-netbeans-modules-sendopts.jar	Provides an API and SPI for registering your own handlers for accessing the command line.
org-netbeans-modules-settings.jar	Provides an API for saving module-specific settings in a user-defined format.
org-openide-awt.jar	Provides many helper classes for displaying UI elements such as notifications.
org-openide-dialogs.jar	Provides an API for displaying standard and customized dialogs.
org-openide-text.jar	Provides an extension to the javax.swing.text API.
org-netbeans-api-visual.jar	Provides a widget & graph library for modeling and displaying visual representations of data.
org-netbeans-spi-quicksearch.jar	Provides the infrastructure for integrating items into the Quick Search field.
org-netbeans-swing-plaf.jar org-netbeans-swing-tabcontrol org-jdesktop-layout.jar	Provides the look and feel and the display of tabs and a wrapper for the Swing Layout Extensions library.

Some of the items in the list above can be used outside of the NetBeans Platform. In these cases, you can put the JAR on the classpath of a standard Swing application and use the related functionality there: org-openide-fileSystems.jar (to use the virtual filesystem), org-openide-util.jar (to use the Lookup class), org-openide-nodes.jar & org-openide-explorer.jar (to use Nodes and explorer views), org-netbeans-swing-outline.jar (to use a Swing treetable component), org-netbeans-api-visual.jar (to use the widget library).

NETBEANS PLATFORM APIs

The NetBeans Platform provides a large set of APIs. You do not need to know or use all of them, just those that make sense in your specific context. Below are the main API groupings, together with the most important information related to the grouping, such as their most important configuration attributes and API classes.

Module System

A module is a JAR file with special attributes in its manifest file. This is a typical NetBeans Platform manifest file:

```
Manifest-Version: 1.0
Ant-Version: Apache Ant 1.7.1
Created-By: 11.3-b02 (Sun Microsystems Inc.)
OpenIDE-Module-Public-Packages: -
OpenIDE-Module-Module-Dependencies: org.openide.util > 7.31.1.1
OpenIDE-Module-Java-Dependencies: Java > 1.5
OpenIDE-Module-Implementation-Version: 091216
AutoUpdate-Show-In-Client: true
OpenIDE-Module: org.demo.hello
OpenIDE-Module-Layer: org/demo/hello/layer.xml
OpenIDE-Module-Localizing-Bundle: org/demo/hello/Bundle.properties
OpenIDE-Module-Specification-Version: 1.0
OpenIDE-Module-Requires: org.openide.modules.ModuleFormat1
```

These are the most important NetBeans-related manifest attributes.

Type	Description
OpenIDE-Module	The identifier of a module, providing a unique name used for recognition by the module system. This is the only required entry in a manifest file for a module.
OpenIDE-Module-Layer	The location and name of the module's layer.xml file, if any.
OpenIDE-Module-Public-Packages	By default, all packages in a module are hidden from all other modules. Via this attribute, you expose packages to external modules.
OpenIDE-Module-Localizing-Bundle	The location and name of a properties file providing a display name and the like.
OpenIDE-Module-Module-Dependencies OpenIDE-Module-Java-Dependencies	Modules can request general or specific versions of other modules ((OpenIDE-Module-Module-Dependencies)), Java packages ((OpenIDE-Module-Package-Dependencies)), or Java itself ((OpenIDE-Module-Java-Dependencies)).
OpenIDE-Module-Provides OpenIDE-Module-Requires	Modules can specify dependencies without naming the exact module to depend on. A module may /provide/ one or more /tokens/. These are strings of conventional meaning, in the format of a Java package or class name.
OpenIDE-Module-Specification-Version OpenIDE-Module-Implementation-Version	In line with the Java Versioning Specification, modules can indicate two pieces of version information about themselves using the [OpenIDE-Module-Specification-Version] and the [OpenIDE-Module-Implementation-Version] tags.
AutoUpdate-Show-In-Client	This attribute determines whether the module is shown in the Plugin Manager.

For details on these and other attributes, see <http://bits.netbeans.org/6.8/javadoc/org-openide-modules>.



To influence the lifecycle of a module, extend org.openide.modules.ModuleInstall, and register it in the manifest under the OpenIDE-Module-Install key.

Window System

The window system handles the display of JPanel-like components and integrates them with the NetBeans Platform. The main classes are listed below.

Type	Description
TopComponent	A JPanel that provides a new window in your application. The window comes with many features for free, such as maximize/minimize and dock/undock.
Mode	A container in which TopComponents are docked. You do not need to subclass this class to use it. Instead, it is configured in an XML file.
TopComponentGroup	A group of windows, which should behave in concert. For example, windows within a group can be opened or closed together. As with Modes, these are defined in an XML file, not by subclassing TopComponentGroup.
WindowManager	Controls all the windows, modes, and window groups. You can request the WindowManager for its windows, modes, and groups. You can also cast it to a JFrame and then set the title bar and anything else that you would do with JFrames.

A mode, that is, a window position, is defined in an XML file, which is contributed to the System FileSystem via registration

entries in the layer.xml file. The NetBeans Platform provides a set of default modes, the most important of which are as follows:

Type	Description
editor	main area of application (not necessarily an actual editor)
explorer	left vertical area, such as for a Projects window
properties	right vertical area, typically for a Properties window
navigator	left lower vertical area
output	horizontal area at base of application
palette	right vertical area for items to drag into a visual pane
leftSlidingSide	minimized state in left sidebar
rightSlidingSide	minimized state in right sidebar
bottomSlidingSide	minimized state in bottom status area


A TopComponent is registered in a mode as follows, note especially the lines in bold below:

```
<folder name="Windows2">
  <folder name="Components">
    <file name="MyTopComponent.settings" url="MyTopComponentSettings.xml"/>
  </folder>
  <folder name="Modes">
    <folder name="editor">
      <file name="MyTopComponent.wstcref" url="MyTopComponentWstcref.xml"/>
    </folder>
  </folder>
</folder>
```

The XML files referred to above are small settings files that NetBeans IDE can generate for you.

Though the default modes should be sufficient for most business scenarios, you might like to adjust them. In that case, you are creating your own mode definitions. A mode definition must follow the related DTD, which is as follows:

http://netbeans.org/dtds/mode-properties2_1.dtd



Hot Tip

Hide the existing modes and create your own, if the existing ones are really not sufficient for you.

The window manager handles the display of the windows in the application's main frame. Aside from being able to access its main Frame, you can also query it for information, as listed below:

How do I...	Description
Find a specific TopComponent?	WindowManager.getDefault().findTopComponent("id")
Find a specific mode?	WindowManager.getDefault().findMode("id") Once you have found a mode, you can use Mode.dockInto(tc) to programmatically dock a TopComponent into a specific mode.
Find a specific TopComponentGroup?	WindowManager.getDefault().findTopComponentGroup("id")
Ensure that the application is fully started up?	WindowManager.getDefault().invokeWhenUIReady(Runnable)
Get the active TopComponent?	WindowManager.getDefault().getRegistry().getActivated()
Get a set of opened TopComponents?	WindowManager.getDefault().getRegistry().getOpened()
Get the main frame of the application	WindowManager.getDefault().getMainWindow()

Lookup

Lookup is a data structure for loosely coupled communication. It is similar to but more powerful than the ServiceLoader class in JDK 6 (for example, Lookup supports event notifications) enabling you to load objects into the context

of your application, but also into the context of NetBeans UI components, such as windows and Nodes.

These are the most important Lookups to be aware of:

Type	Description
Global lookup, provides selection management	The Lookup that gives you access to the currently selected UI component, most commonly the focused Node. Lookup lkp = Utilities.actionsGlobalContext();
Local lookup, provides lookup of NetBeans objects such as windows and Nodes	The local context of a specific NetBeans Platform UI object. //For Windows components: Lookup lkp = myTopComponent.getLookup(); //For Nodes: Lookup lkp = myNode.getLookup();
Default lookup	The application's context, comparable to the JDK 6 ServiceLoader class, provided via the META-INF/services folder. Lookup lkp = Lookup.getDefault();

These are typical tasks related to Lookup and how to code them:

How do I...	Description
Register a service?	Annotate a service provider with the @ServiceProvider class annotation, at compile time the META-INF/services folder is created, registering the implementation.
Find the default service implementation?	MyService svc = Lookup.getDefault().lookup(MyService.class)
Find all service implementations?	Collection<? extends MyService> coll = Lookup.getDefault().lookupAll(MyService.class)
Listen to changes in a Lookup?	Lookup.Result lkpResult = theLookup.lookupResult(MyObject.class); lkpResult.addLookupListener(new LookupListener() { @Override public void resultChanged(LookupEvent e)(Result res = (Result) e.getSource(); Collection<? extends MyObject> coll = res.allInstance(); //iterate through the collection }); lkpResult.allInstances(); // first call is needed
Create a Lookup for an object?	//Lookup for single object: Lookup lkp = Lookups.singleton(myObject); //Lookup for multiple objects: Lookup lkp = Lookups.fixed(myObject, other); //Lookup for dynamic content: InstanceContent ic = new InstanceContent(); Lookup lkp = new AbstractLookup(ic); ic.add(myObject);
Merge Lookups?	Lookup commonLkp = new ProxyLookup(dataObjectLookup, nodeLookup, dynamicLookup);
Provide a Lookup for my TopComponent?	//In the constructor of TopComponent: associateLookup(myLookup);
Provide a Lookup for a subclass of AbstractNode?	new AbstractNode(myKids, myLookup);

Central Registry (System FileSystem)

The central registry is organized as a virtual file system accessible by all the modules in a NetBeans Platform application. NetBeans Platform APIs, such as the Window System API, make available extension points enabling you to declaratively register your components. A module's contributions to the system are provided by specialized XML files, called "layer files", normally named "layer.xml".

Below are the most important extension points provided out of the box by the NetBeans APIs, represented by folders in a layer file:

Actions, Menu, Toolbars, Navigator/Panels, OptionsDialog, Services, Shortcuts, TaskList, and Windows2.

The NetBeans Platform helps you to register items correctly in the file system by letting you, in some cases, annotate your classes instead of requiring you to manually type XML tags in the layer.xml file by hand. The current set of annotations impacting the layer.xml is listed below:

@AntBasedProjectRegistration, @CompositeCategoryProvider.Registration, @EditorActionRegistration, @LookupMerger.Registration, @LookupProvider.Registration, @NodeFactory.Registration, @OptionsPanelController.ContainerRegistration, @OptionsPanelController.SubRegistration, @OptionsPanelController.TopLevelRegistration, @ProjectServiceProvider, @ServiceProvider, @ServicesTabNodeRegistration

The registry makes use of the FileSystem API to access the registered data.

FileSystem API

The FileSystem API provides stream-oriented access to flat and hierarchical structures, such as disk-based files on local or remote servers, memory-based files and even XML documents.

Items within the folders in the layer.xml file are not java.io.Files, but org.openide.filesystems.FileObjects. The differences between them are as follows:

java.io.File	org.openide.filesystems.FileObject
Create with a constructor	Get from the FileSystem
Can represent something that doesn't exist, such as new File("some/place/that/doesn't/exist")	Typically represents something that exists
Cannot listen to changes	FileChangeListener listens to changes to FileObject, as well as anything beneath the FileObject
Represents a file on disk	Not necessarily a file on disk, could be in a database, FTP server, virtual, or anywhere else
No attributes	Can have attributes, which are key-value pairs associated with a FileObject

Converting between common data types:

How do I get...	Description
a java.io.File for a FileObject?	FileUtil.toFile(FileObject fo)
a FileObject for a File?	FileUtil.toFileObject(File f)
a DataObject for a FileObject?	DataObject.find(theFileObject)
a FileObject for a DataObject?	theDataObject.getPrimaryFile()
a Node for a DataObject?	theDataObject.getNodeDelegate()
a DataObject for a Node?	DataObject dob = (DataObject)theNode.getLookup().lookup(DataObject.class); if (dob != null) { //do something }
a reference to the System FileSystem?	//Get the root: FileUtil.getConfigRoot() //Get a specific folder: FileUtil.getConfigFile("path/to/my/folder")

The NetBeans Platform provides a number of custom URLs for accessing things.

jar	For representing entries inside JARs and ZIPs, including the root directory entry
nbres	A resource loaded from a NetBeans module, e.g. nbres:/org/netbeans/modules/foo/resources/foo.dtd
nbres:loc	Same, but transparently localized and branded according to the usual conventions, e.g. nbres:loc:/org/netbeans/modules/foo/resources/foo.html may actually load the same thing as nbres:/org/netbeans/modules/foo/resources/foonbja.html.
nbinst	Loads installation files using InstalledFileLocator in installation directories, e.g. nbinst:///modules/ext/some-lib.jar may load the same thing as file:/opt/netbeans/ide4/modules/ext/some-lib.jar.
nbfs	Refers to a file object in the System FileSystem (XML layers). For example, nbfs:/SystemFileSystem/Templates/Other/html.html refers to an HTML file template installed in the IDE

Actions

Actions are pieces of code that are invoked when the user presses a menu item, toolbar button, or keyboard shortcut.

Type	Description
Actions.alwaysEnabled	An action that is always enabled. Typically, use this for "File > Open" actions, for example.
Actions.context	An action that is bound to a context. If an action should only be enabled under certain conditions, use this action.
Actions.callback	An action with an assigned key used to find a delegate in the ActionMap of the active component.

When porting your existing application to the NetBeans Platform, you do not need to change your standard JDK

actions (AbstractAction, ActionListener, etc) in any way. Instead, you need to bind them to one of the classes listed above, using the layer.xml file to do so.

Actions.alwaysEnabled:

```
<file name="your-pkg-action-id.instance">
  <attr name="instanceCreate"
    methodvalue="org.openide.awt.Actions.alwaysEnabled"/>
  <attr name="delegate"
    methodvalue="your.pkg.YourAction.factoryMethod"/>
  <attr name="displayName" bundlevalue="your.pkg.Bundle#key"/>
  <attr name="iconBase" stringvalue="your/pkg/YourImage.png"/>
  <!-- if desired: <attr name="noIconInMenu" boolvalue="false"/>
  <!-- if desired: <attr name="asynchronous" boolvalue="true"/>
</file>
```

Actions.context:

```
<file name="action-pkg-ClassName.instance">
  <attr name="instanceCreate"
    methodvalue="org.openide.awt.Actions.context"/>
  <attr name="type" stringvalue="org.netbeans.api.actions.Openable"/>
  <attr name="selectionType" stringvalue="ANY"/> (or EXACTLY_ONE)
  <attr name="delegate" newvalue="action.pkg.YourAction"/>
  <attr name="key" stringvalue="KeyInActionMap"/>
  <attr name="surviveFocusChange" boolvalue="false"/>
  <attr name="displayName" bundlevalue="your.pkg.Bundle#key"/>
  <attr name="iconBase" stringvalue="your/pkg/YourImage.png"/>
  <!-- if desired: <attr name="noIconInMenu" boolvalue="false"/>
  <!-- if desired: <attr name="asynchronous" boolvalue="true"/>
</file>
```

Actions.callback:

```
<file name="action-pkg-ClassName.instance">
  <attr name="instanceCreate"
    methodvalue="org.openide.awt.Actions.callback"/>
  <attr name="key" stringvalue="KeyInActionMap"/>
  <attr name="surviveFocusChange" boolvalue="false"/>
  <attr name="fallback" newvalue="action.pkg.DefaultAction"/>
  <attr name="displayName" bundlevalue="your.pkg.Bundle#key"/>
  <attr name="iconBase" stringvalue="your/pkg/YourImage.png"/>
  <!-- if desired: <attr name="noIconInMenu" boolvalue="false"/>
  <!-- if desired: <attr name="asynchronous" boolvalue="true"/>
</file>
```

For all the details, see <http://bits.netbeans.org/6.8/javadoc/org-openide-awt/org-openide-awt/Actions.html>

An action is registered in the layer.xml file in the "Actions" folder, within a folder reflecting its place in the action pool.

```
<folder name="Actions">
  <folder name="Window">
    <file name="your-pkg-action-id.instance">
      ...
    </file>
  </folder>
</folder>
```

Depending on your business requirements, once you have actions registered in the layer.xml file, you need to bind them to menu items, toolbar buttons, and keyboard shortcuts.

Menu Items

```
<folder name="Menu">
  <folder name="Window">
    <file name="your-pkg-action-id.shadow">
      <attr name="originalFile" stringvalue="Actions/Window/your-pkg-action-id.instance"/>
      <attr name="position" intvalue="50"/>
    </file>
  </folder>
</folder>
```

Toolbar Buttons

```
<folder name="Toolbars">
  <folder name="Window">
    <file name="your-pkg-action-id.shadow">
      <attr name="originalFile" stringvalue="Actions/Window/your-pkg-action-id.instance"/>
      <attr name="position" intvalue="50"/>
    </file>
  </folder>
</folder>
```

Shortcuts

```
<folder name="Shortcuts">
  <file name="M.shadow">
    <attr name="originalFile"
      stringValue="Actions/Edit/org-netbeans-modules-
      some-MyAction.instance"/>
  </file>
  <file name="D-M.shadow">
    <attr name="originalFile"
      stringValue="Actions/Edit/org-netbeans-modules
      some-CtrlMyAction.instance"/>
  </file>
  <file name="O-M.shadow">
    <attr name="originalFile"
      stringValue="Actions/Edit/org-netbeans-modules
      some-AltMyAction.instance"/>
  </file>
</folder>
```

As key code, use `KeyEvent.VK_keycode` without `VK_` prefix, as described in Javadoc of `org.openide.util.Utilities.stringToKey()` and `keyToString()` methods.

Nodes

A Node is a generic model for a business object, which it visualizes within an Explorer View. Each Node can have visual attributes, such as a display name, icon, and actions. The list of Nodes is below, which you can use as is or extend as needed:

Type	Description
Node	Base class for representing business objects to the user.
AbstractNode	The usual base class for Node implementations.
DataNode	Specialized Node class for wrapping a file and displaying it as a Node to the user.
BeanNode	Specialized Node class that wraps a JavaBean and presents it to the user as a Node. It also provides simplistic access to property sheets.
FilterNode	Specialized Node class that decorates an existing Node by adding/removing features to/from it.

A Node is a container for its own child Nodes. These classes create child Nodes:

ChildFactory	Factory for creating child Nodes. Optionally, these can be created asynchronously, that is, when the user expands the Node.
Children.Keys	Older version of ChildFactory. You should be able to replace any implementation of Children.Keys with ChildFactory.

`Children.Array`, `Children.Map`, `Children.SortedArray`, `Children.SortedMap`: These classes are less frequently used and are no longer well supported.

Explorer Views

Explorer views are Swing components that display Node hierarchies.

BeanTreeView	JTree for displaying Nodes.
OutlineView	JTree for displaying Nodes, with a JTable for displaying the related Node properties.
PropertySheetView	Property sheet displaying the properties of a Node.

`IconView`, `ListView`, `ChoiceView`, `ContextTreeView`, `MenuView`, `TableView`: These classes are less frequently used and are no longer well supported.

For all the details, see <http://bits.netbeans.org/6.8/javadoc/org-openide-explorer>.

Visual Library

The NetBeans visual library is a generic widget and graph library, providing a collection of predefined widgets that integrate well with other NetBeans Platform objects, such as Nodes and windows. Below is the list of widgets provided by this library.

Type	Description
Scene	Provides the root element of the hierarchy of displayed widgets.

LayerWidget	Provides a transparent surface, comparable to a JGlassPane.
ComponentWidget	Provides a placeholder widget for AWT/Swing components.
ImageWidget	Provides images to the scene.
LabelWidget	Provides a label as a widget.
IconNodeWidget	Provides an image and a label.
ConnectionWidget	Provides connections between widgets, with anchors, control points, and end points.
ConvolveWidget	Provides coil/twist effect, i.e., a convolve filter to a child element.
LevelOfDetailsWidget	Provides a container for widgets, with visibility and zoom features.
ScrollWidget/ SwingScrollWidget	Provides a scrollable area, with/ without JScrollbar as scroll bars.
SeparatorWidget	Provides a space with thickness and orientation.

For all the details, see <http://bits.netbeans.org/6.8/javadoc/org-netbeans-api-visual>.

Dialogs

The NetBeans Dialogs API provides a number of standard dialogs for displaying standard messages for information, questions (such as "Are you sure?", when saving), input, and error messages to the user. Each dialog comes with a standard appearance, buttons, and icons.

Type	Description
Information Dialog	<code>NotifyDescriptor d = new NotifyDescriptor.Message("Text");</code>
Question Dialog	<code>NotifyDescriptor d = new NotifyDescriptor.Confirmation("Title", "Text");</code>
Input Dialog	<code>NotifyDescriptor d = new NotifyDescriptor.InputLine("Title", "Text");</code>

Add the Dialogs API to your module and then use the table above to create dialogs as follows (in the example below, we display an information dialog):

```
NotifyDescriptor d = new NotifyDescriptor.Message("Text");
DialogDisplayer.getDefault().notify(d);
```

For all the details, see <http://bits.netbeans.org/6.8/javadoc/org-openide-dialogs>.

The Dialogs API also provides a group of Wizard classes for multipage dialogs!

Other Useful NetBeans APIs

Type	Description
<code>org.apache.tools.ant.module.api.support.ActionUtils</code>	Used for running Ant targets.
<code>org.openide.util.ImageUtilities</code>	Provides useful static methods for manipulation with images/icons, results are cached.
<code>org.openide.awt.Html-Browser.URLDisplayer</code>	Displays URLs, opens browsers, distinguishes embedded vs. external browsers.
<code>org.netbeans.api.progress.ProgressHandle</code>	Integrates visualization of long running tasks with the NetBeans Platform progress bar.
<code>org.openide.util.RequestProcessor</code>	Performs asynchronous threads in a dedicated thread pool.
<code>org.openide.awt.UndoRedo</code>	Listens to editing changes and activates the "Undo" and "Redo" buttons.

Explore the Utilities API (`org.openide.util.jar`) for many useful classes that all NetBeans Platform applications can use.

NETBEANS PLATFORM GOTCHAS

The following are frequently asked questions, returning over and over again, with their answers:

FAQ	Answers
What's the difference between a 'NetBeans Platform Application' and a 'Module Suite'?	The former gives you a subset of the NetBeans Platform, the latter a subset of NetBeans IDE. Use the former when building on the NetBeans Platform, the latter when building on NetBeans IDE.
Why is my explorer view not synchronized with the Properties window?	Because you need to add this line to the TopComponent constructor: associateLookup(ExplorerUtils.createLookup(em, getActionMap()));
I created a palette but it isn't showing when I open the related TopComponent.	Because your TopComponent is not in 'editor' mode.







NETBEANS PLATFORM ON-LINE

Technical information on the NetBeans Platform is available on-line in many different forms. The most important of these are listed below:

Site	URL
Homepage	http://platform.netbeans.org
Tutorials	http://platform.netbeans.org/tutorials
NetBeans API javadoc	http://bits.netbeans.org/6.8/javadoc/
Blogs	http://planet.netbeans.org
FAQ	http://wiki.netbeans.org/NetBeansDeveloperFAQ
Mailing List	dev@platform.netbeans.org http://netbeans.org/platform/lists/dev/archive
Screencast	http://platform.netbeans.org/tutorials/nbm-10-top-apis.html

This Refcard could not have been made without the technical insights provided by the following: Tim Boudreau, Tim Dudgeon, Jeremy Faden, Laurent Forêt, Jesse Glick, Manikantan, Ernie Rael, Antonio Vieiro and Tom Wheeler.

ABOUT THE AUTHORS

-  **Heiko Böck** is the author of the well-known "The Definitive Guide to NetBeans Platform"
-  **Anton Eppler** (<http://eppleton.sharedhost.de/>) is a NetBeans Platform consultant & trainer.
-  **Miloš Šilhánek** is a Java, NetBeans Platform, 3D and AI enthusiast and Czech translator of Heiko Böck's book.
-  **Andreas Stefik** is an Assistant Professor at Southern Illinois University Edwardsville.
-  **Tom Wheeler** (<http://www.tomwheeler.com>) has worked with NetBeans Platform nearly every day for the past five years and is a consultant, trainer and member of the NetBeans Dream Team.
-  **Geertjan Wielenga** works on the NetBeans team and is co-author of "Rich Client Programming: Plugging into the NetBeans Platform"

RECOMMENDED BOOKS



The Definitive Guide to NetBeans™ Platform is a thorough and definitive introduction to the NetBeans Platform, covering all its major APIs in detail, with relevant code examples used throughout.

BUY NOW

books.dzone.com/books/definitive-guide-netbeans



Rich Client Programming will help you get started with NetBeans module development, master NetBeans' key APIs, and learn proven techniques for building reliable desktop software.

BUY NOW

books.dzone.com/books/richclientprog

Professional Cheat Sheets You Can Trust

"Exactly what busy developers need: simple, short, and to the point."

James Ward, Adobe Systems

Upcoming Titles

- Blaze DS
- Domain Driven Design
- Virtualization
- Java Performance Tuning
- Expression Web
- Spring Web Flow
- Continuous Integration

Most Popular

- Spring Configuration
- jQuery Selectors
- Windows Powershell
- Dependency Injection with EJB 3
- NetBeans IDE JavaEditor
- Getting Started with Eclipse
- Very First Steps in Flex

Download Now
Refcardz.com



DZone communities deliver over 6 million pages each month to more than 3.3 million software developers, architects and decision makers. DZone offers something for everyone, including news, tutorials, cheatsheets, blogs, feature articles, source code and more. "DZone is a developer's dream," says PC Magazine.

DZone, Inc.
140 Preston Executive Dr.
Suite 100
Cary, NC 27513
888.678.0399
919.678.0300
Refcardz Feedback Welcome
refcardz@dzone.com
Sponsorship Opportunities
sales@dzone.com

