DZone Refcardz

brought to you by... eppleton

CONTENTS INCLUDE:

Refcard Update! Applies to NetBeans Platform 7.0, 7.0.1, & 7.1

NetBeans Platform 7

A Framework for Building Pluggable Enterprise Applications

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" (such as the code for managing windows, connecting actions to menu items, and updating applications at runtime) that you would otherwise need to write yourself. 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:

Tool	How to Get Started
NetBeans IDE	Download the "Java SE" distribution of NetBeans IDE, which is the smallest NetBeans IDE distribution providing the NetBeans Platform Toolkit, consisting of NetBeans Platform templates & samples, via either the Ant or Maven build systems.

Command-line Maven	Use the Maven archetypes for NetBeans Platform development.
	`GroupId: org.codehaus.mojo.archetypes`
	`ArtifactId:`
	`  netbeans-platform-app-archetype`
	`  nbm-archetype`

Command-line 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.
Other IDEs	Use command-line Maven to set up a Maven-based NetBeans Platform application and then open the POM file into any IDE that supports Maven, e.g., IntelliJ IDEA or Eclipse.

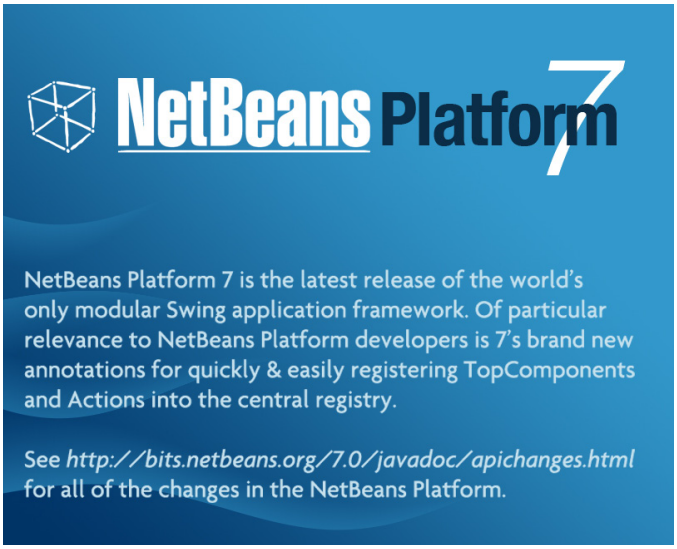
Hot Tip

Join the NetBeans Platform mailing list dev@platform.netbeans.org, where you can discuss problems and share ideas with other developers using the NetBeans Platform as the basis of their software!

MAIN FEATURES

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

Tool	How to Get Started
Module System	Since your application can use either standard NetBeans Platform modules or OSGi bundles, you'll be able to integrate third-party modules or develop your own. The modular nature of a NetBeans Platform application gives you the power to meet complex requirements by combining several small, simple, and easily tested modules. Powerful versioning support helps give you confidence that your modules will work together, while strict control over the public APIs your modules expose will help you create a more flexible application that's easier to maintain.
Lifecycle Management	Just as application servers such as GlassFish provide lifecycle services to web applications, the NetBeans runtime container provide lifecycle 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 from pluggable applications because these enable them to install modules into their running applications. NetBeans modules can be installed, uninstalled, activated, and deactivated at runtime, thanks to the runtime container.
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 (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 any code at all.
Standardized UI Toolkit	Swing is the standard UI toolkit and is the basis of all NetBeans Platform applications. Related benefits include the ability to change the look and feel easily, the portability of Swing across all operating systems, and the easy incorporation of many free and commercial third-party Swing components.
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 to 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, 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 six. None of these 6 provide any UI at all, meaning that you can create server or console applications on the NetBeans Platform, since UI is not mandatory in any way at all.

Complete list of NetBeans Platform modules:

Module	Description
boot.jar core.jar org-openide-filesystems.jar org-openide-modules.jar org-openide-util.jar org-openide-util-lookup.jar	Provides the runtime container, consisting of the startup sequence, the module system, the filesystem, the service infrastructure, and utility classes.
org-netbeans-core.jar org-netbeans-core-execution.jar org-netbeans-core-ui.jar org-netbeans-core-windows	(Optional) Provides the basic UI components provided by the NetBeans Platform, together with related infrastructure.
org-netbeans-core-netigso.jar org-netbeans-core-osgi.jar org-netbeans-libs-felix.jar org-netbeans-libs-osgi.jar	(Optional) Provides integration with the OSGI containers Felix and Equinox.
org-netbeans-core-output2.jar org-openide-io.jar	(Optional) 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	(Optional) Provides a framework for multi-tab windows, such as used by the Matisse GUI Builder in NetBeans IDE.
org-openide-windows.jar	(Optional) Provides the API for accessing the window system.
org-netbeans-modules-autoupdate-services.jar org-netbeans-modules-autoupdate-ui.jar	(Optional) Provides the Plugin Manager together with the functionality for accessing and processing update centers where NetBeans modules are stored.
org-netbeans-modules-favorites.jar	(Optional) 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	(Optional) Provides a number of configurable system actions, such as "Cut", "Copy", and "Paste".
org-openide-loaders.jar	(Optional) Provides the API for connecting data loaders to specific MIME types.
org-openide-nodes.jar org-openide-explorer.jar	(Optional) Provides the API for modeling business objects and displaying them to the user.
org-netbeans-modules-javahelp.jar	(Optional) 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	(Optional) Provides an API for discovery and creation of MIME-specific settings and services.
org-netbeans-modules-masterfs.jar	(Optional) Provides a central wrapper file system for your application.
org-netbeans-modules-options-api.jar	(Optional) Provides an Options window for user customizations and an API for extending it.
org-netbeans-api-progress.jar org-openide-execution.jar	(Optional) 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	(Optional) Provides an API for getting information about files and an SPI for creating your own queries.
org-netbeans-modules-sendopts.jar	(Optional) Provides an API and SPI for registering your own handlers for accessing the command line.
org-netbeans-modules-settings.jar	(Optional) Provides an API for saving module-specific settings in a user-defined format.
org-openide-awt.jar	(Optional) Provides many helper classes for displaying UI-elements such as notifications.
org-openide-dialogs.jar	(Optional) Provides an API for displaying standard and customized dialogs.
org-openide-text.jar	(Optional) Provides an extension to the java.swing text API.
org-netbeans-api-visual.jar	(Optional) Provides a widget library for modeling and displaying visual representations of data.
org-netbeans-spi-quicksearch.jar	(Optional) Provides the infrastructure for integrating items into the Quick Search field.
org-netbeans-swing-plaf.jar org-netbeans-swing-tabcontrol org-jdesktop-layout.jar	(Optional) Provides the look and feel and the display of tabs and a wrapper for the Swing Layout Extensions 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 module 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:

Attribute	Defines
OpenIDE-Module	Identifier of a module, provides a unique name. The only required entry in the manifest.
OpenIDE-Module-Layer	(Optional) Location and name of the module's layer file.
OpenIDE-Public-Packages	(Optional) By default, all packages in a module are hidden from all other modules. Via this attribute, you expose packages to external modules.
OpenIDE-Module-Friends	(Optional) By default, all modules in the application can access all public packages. Via this attribute, you can limit access to public packages to specific modules.
OpenIDE-Module-Localizing-Bundle	(Optional) The location and name of a properties file used as the module's localizing bundle.
OpenIDE-Module-Module-Dependencies OpenIDE-Module-Java-Packages-Dependencies OpenIDE-Module-Java-Dependencies	(Optional) Modules can request general or specific versions of other modules, Java packages, or Java itself.
OpenIDE-Module-Provides OpenIDE-Module-Requires	(Optional) Modules can specify dependencies without naming the exact module to depend on. A module may "provide" one or more "tokens", strings in the format of a Java package or class name.
OpenIDE-Module-Specification-Version OpenIDE-Module-Implementation-Version	(Optional) Modules can indicate two types pieces of versioning.
AutoUpdate-Show-In-Client	(Optional) Whether the module is shown in the Plugin Manager.

Hot Tip For details on these and other attributes, see: http://bits.netbeans.org/dev/javadoc/org-openide-modules

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, i.e., without any coding, 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 consort. 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 central registry via 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, e.g., for Projects window
properties	right vertical area, e.g., for Properties window
navigator	left lower vertical area, e.g., for Navigator window
output	horizontal area at base of application
palette	right vertical area, e.g., for items to drag onto a window
leftSlidingSide	minimized state in left sidebar
rightSlidingSide	minimized state in right sidebar
bottomSlidingSide	minimized state in bottom status area

TopComponents are registered in the system as follows:

```
@TopComponent.Description(preferredID = "DemoTopComponent",
iconBase="SET/PATH/TO/ICON/HERE",
persistenceType = TopComponent.PERSISTENCE_ALWAYS)
@TopComponent.Registration(mode = "editor", openAtStartup = true)
@TopComponent.OpenActionRegistration(displayName = "#CTL_DemoAction",
preferredID = "DemoTopComponent")
public final class DemoTopComponent extends TopComponent {
  //
}
```

At compile time, the annotations above are converted to entries in a generated layer file, in the "build" folder.

The WindowManager handles the display of the windows in the application's main frame. It is the most important class when you need to manipulate the window system:

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 TopComponent Group?	WindowManager.getDefault().findTopComponentGroup("id")
Ensure that the application is fully started up?	WindowManager.getDefault().invokeWhenUIReady(MyRunnable())
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). This enables 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 TopComponents, Nodes, and DataObjects.	The local context of a specific NetBeans Platform UI object.
	`//For Windows:`
	`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();`

Hot Tip What's a cookie? A cookie is a dynamically assigned capability, e.g., to save an object , lookup its SaveCookie and call save() on it.

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 s = Lookup.getDefault().lookup(MyService.class)
Find all service implementations?	Collection<? extends MyService> coll = Lookup.getDefault().lookupAll(MyService.class)

Listen to changes in a Lookup?
 	
Tip: Keep a reference to the result object, otherwise it will be garbage collected.
	`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(); // need first call`

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 Node?	`new AbstractNode(myKids, myLookup);`
	`new BeanNode(myDomainObj, myKids, myLookup);`
	`new DataNode(myDataObject, myKids, myLookup);`

Provide a Lookup for any other component?	I implement Lookup.provider.

For all the details, see:
http://wiki.netbeans.org/NetBeansDeveloperFAQ#Lookup

Hot Tip Follow the 4-part beginner's tutorial on Lookup here: http://platform.netbeans.org/tutorials/nbm-selection-1.html

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, OptionsDialog, Services, Shortcuts, TaskList, and Windows2.

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

```
@ActionID, @ActionReference, @ActionReferences, @ActionRegistration,
@AntBasedProjectRegistration, @CompositeCategoryProvider.Registration,
@ConvertAsJavaBean, @ConvertAsProperties, @EditorActionRegistration,
@LookupMerger.Registration, @LookupProvider.Registration,
@MimeLocation, @MimeRegistration, @MimeRegistrations,
@NbBundle.Messages, @NodeFactory.Registration,
@OptionsPanelController.ContainerRegistration,
@OptionsPanelController.SubRegistration,
@OptionsPanelController.TopLevelRegistration,
@ProjectServiceProvider, @ServiceProvider, @ServiceProviders,
@ServicesTabNodeRegistration, @TopComponent.OpenActionRegistration,
@TopComponent.Registration
```

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/doesnt/exist")	Represents something that already 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...	Description
Get a java.io.File for a FileObject?	FileUtil.toFile(FileObject fo)
Get a FileObject for a File?	FileUtil.toFileObject(File f)
Get a DataObject for a FileObject?	DataObject.find (FileObject fo)
Get a FileObject for a DataObject?	theDataObject.getPrimaryFile()
Get a Node for a DataObject?	theDataObject.getNodeDelegate()

Get a DataObject for a Node?	DataObject dob = n.getLookup().lookup (DataObject.class);
	if (dob != null) {
	//do something
	}

Get a reference to the central registry?	//Get the root:
	FileUtil.getConfigRoot()
	//Get a specific folder:
	FileUtil.getConfigFile("path/to/my/folder")

The NetBeans Platform provides custom URLs:

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.
nbresloc	Same, but transparently localized and branded according to the usual conventions, e.g. nbresloc:/org/netbeans/modules/foo/resources/foo.html loads the same as nbres:/org/netbeans/modules/foo/resources/foo_nb_ja.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/ide/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.

For all the details, see:
http://bits.netbeans.org/dev/javadoc/org-openide-filesystems

Actions

Actions are functions invoked when the user presses a menu item, toolbar button, or keyboard shortcut.

When porting your existing application to the NetBeans Platform, you do not need to change the code in your standard JDK actions (AbstractAction, ActionListener, etc).

Instead, you need to register them via class-level annotations which, when the module is compiled, will result in entries generated in the module's layer file.

Below is shown how an action is registered:

```
@ActionID(category = "File",
id = "org.demo.project.actions.DemoAction")
@ActionRegistration(iconBase = "org/demo/resources/icon.png",
displayName = "#CTL_DemoAction")
@ActionReferences({
    @ActionReference(path = "Menu/File", position = 0),
    @ActionReference(path = "Toolbars/File", position = 0),
    @ActionReference(path = "Shortcuts", name = "D-SPACE")
})
@Messages("CTL_DemoAction=Demo")
public final class DemoAction implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {
        // TODO implement action body
    }
}
```

The entries generated into the layer file from the annotation above are used by the NetBeans Platform to construct the application's actions, menus, toolbars, and keyboard shortcuts.

When needing to change a component in the menu bar or toolbar, extend AbstractAction and implement Presenter:

```
@ActionID(category = "Build", id = "org.demo.module1.DemoAction")
@ActionRegistration(iconBase = "org/demo/module1/icon.png",
displayName = "#CTL_DemoAction")
@ActionReferences({
    @ActionReference(path = "Menu/File", position = 0),
    @ActionReference(path = "Toolbars/File", position = 0),
    @ActionReference(path = "Shortcuts", name = "D-M")
})
public final class DemoAction extends AbstractAction implements
Presenter.Toolbar
{
    public void actionPerformed(ActionEvent e) {
        // TODO implement action body
    }
    public Component getToolbarPresenter() {
        // TODO define the component to be displayed
    }
}
```

For all the details, see:
http://bits.netbeans.org/dev/javadoc/org-openide-awt

Nodes, Explorer Views & Property Sheets

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, properties, 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 you would use for your 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. Factory classes:

Type	Description
ChildFactory	Factory for creating child Nodes. Optionally, these can be created asynhronously, 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.

Less Used: Children.Array, Children.Map, and Children.SortedArray.

Explorer views are Swing components that display Node hierarchies.

Type	Description
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.
PropertyPanel	A generic GUI component for displaying and editing a single JavaBeans property.

Less Used: IconView, ListView, ChoiceView, ContextTreeView, MenuView, and TableView.

The NetBeans Platform provides default property editors for:

```
Boolean, Color, Dimension, Font, Insets, Integer, Point, Rectangle,
File, Class, String, URL, Date, Properties, ListModel, and TableModel.
```

For all the details, see:
http://bits.netbeans.org/dev/javadoc/org-openide-explorer/org/openide/explorer/doc-files/propertyViewCustomization.html

Visual Library

The Visual Library provides predefined widgets with predefined actions, layouts, and borders. The list of predefined widgets is as follows:

Type	Description
Scene	Provides the root element of the hierarchy of displayed widgets.
LayerWidget	Provides a transparent surface, like 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/dev/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	NotifyDescriptor d = new NotifyDescriptor.Message("Text");
Question Dialog	NotifyDescriptor d = new NotifyDescriptor.Confirmation("Title", "Text");
Input Dialog	'NotifyDescriptor d = new NotifyDescriptor.InputLine("Title", "Text");

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/dev/javadoc/org-openide-dialogs

Hot Tip The Dialogs API also provides a group of Wizard classes for multipage dialogs.

Other Useful NetBeans APIs

Type	Description
org.apache.tools.ant.module.api.support.ActionUtils	Used for running Ant targets.
org.openide.awt.HtmlBrowser.URLDisplayer	Displays URLs, opens browsers, distinguishes embedded vs. external browsers.
org.openide.awt.StatusDisplayer	Lets you control the status line, i.e, you can write into it or change it.
org.openide.util.ImageUtilities	Provides useful static methods for manipulation with images/icons, results are cached.
org.openide.util.RequestProcessor	Performs asynchronous threads in a dedicated thread pool.
org.openide.windows.IOProvider	Lets you create new tabs in the Output window and write into them.

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

REUSABLE SWING COMPONENTS

In addition to the NetBeans APIs, many UI components can be used as-is, simply by including the related JAR (or JARs) in your application. Normally, they also provide an API for accessing and changing their default behavior.

Component Palette, Database Explorer, Debugger, Diff Viewer, Editors (Properties, Java, XML, HTML, SQL, and more), External Browser, File Browser (Favorites Window), JavaHelp Window, Navigator Window, Options Window, Output Window, Progress Bar, Properties Window, Plugin Manager, Project System, Quick Search Field, Task List, Terminal Window, UndoRedo Manager, Validation System, and Versioning Systems (CVS, Subversion, Mercurial).

BRANDING

Everything in the NetBeans Platform can be customized to fit your specific business requirements. For example:

Icons, Look & Feel, Splash Screen, Strings (Menu Items, Title Bar, etc.)

Right-click an application in the Projects window, choose Branding, and use the Branding Editor.

NETBEANS PLATFORM GOTCHAS

The following are frequently asked questions:

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 have not exposed the Node, with its Properties, to the Lookup. For example, 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 registered in the 'editor' mode, where the palette is available.

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	platform.netbeans.org
NetBeans API javadoc	http://bits.netbeans.org/dev/javadoc/
Tutorials	platform.netbeans.org/tutorials
Blogs	planetnetbeans.org
FAQ	http://wiki.netbeans.org/NetBeansDeveloperFAQ
Mailing List	dev@platform.netbeans.org
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: David Beer, Tim Boudreau, Thibaut Colar, Tim Dudgeon, Jeremy Faden, Laurent Forêt, Jesse Glick, Aleksandar Kochnev, Manikantan, Ernie Rael, Bernd Ruehlicke, Kris Scorup, Andrea Selva, Timothy Sparg, and Antonio Vieiro.

ABOUT THE AUTHOR






Heiko Böck is the author of the well-known "The Definitive Guide to NetBeans Platform."

Anton Epple (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 BOOK

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

DZone

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, cheat sheets, 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

ISBN-13: 978-1-936502-03-5
ISBN-10: 1-936502-03-8
50795
9 781936 502035
$7.95

Version 2.0