Building an Application Using NetBeans™ Framework

By Frank Cohen

Abstract

The NetBeans™ platform is more than the basis for the Sun™ ONE Studio (previously Forte™ for Java™) integrated development environment ("IDE"). With this powerful toolset, any software developer may build powerful desktop application software, including commercial applications that have nothing to do with Java technology development. In this article, Frank Cohen describes how to create your own NetBeans modules and how to create your own branded application using the NetBeans platform.

Introduction to NetBeans

Who could have imagined the tremendous effect on software development made by combining Java software development technology with open-source development methodologies? From the start, Java technology's theme has been to maximize the utility of writing software once and running everywhere Java technology runs. NetBeans technology's theme has been to focus a software developer's attention on writing the core logic in desktop application software and to let the NetBeans platform worry about the mundane issues of file editing, pluggable and downloadable tools, file management, and desktop integration. The combination of Java technology and open-source methodologies gives little reason to write a desktop application today without building it on the NetBeans platform.

The NetBeans platform is a software framework for building desktop application software in the Java programming language. Sun Microsystems was the first company to build a commercial application based on the NetBeans platform, but many others have followed. For example, Project XEMO is a music composition environment built on NetBeans[1]. Companies building application software with NetBeans take advantage of the Mozilla™-style open-source license. The license allows anyone to download, use and improve the NetBeans source code, and even to develop commercial applications with no royalty owed.

1 Additional examples are at http://www.netbeans.org/third-party.html

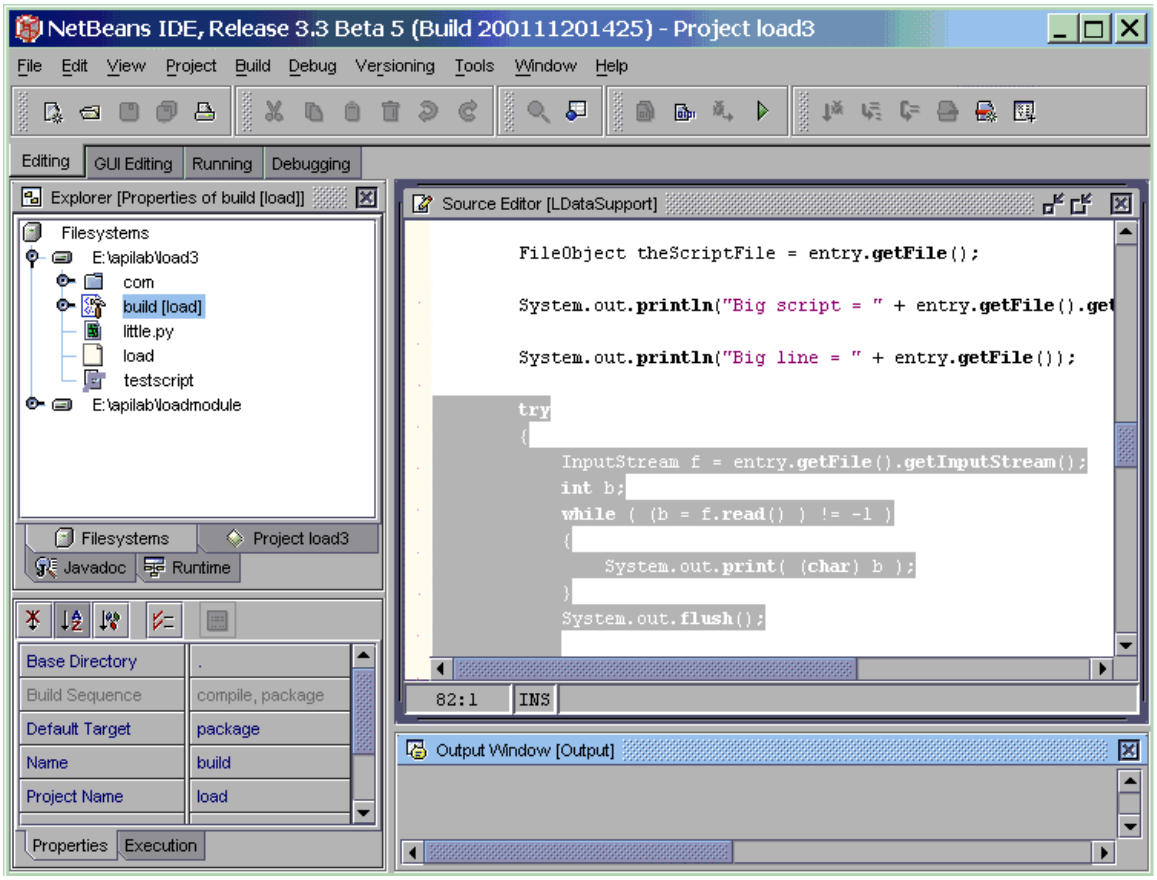

Fig. 1 - The NetBeans Environment

NetBeans technology was conceived as a framework, so almost all functionality is delivered as "modules" within a pluggable architecture. Software developers choose to either make use of existing NetBeans modules or plug their own new modules into the NetBeans platform. Either choice enables software developers to deliver high quality, highly functional software faster and with less maintenance over time.

The Sun Public License and What It Means To You

It is common sense at this point in the open-source revolution that NetBeans source code should be licensed under a variant of the popular Mozilla Public License. Mozilla[2] is the open-source project from which the popular Netscape Navigator™ browser is developed. The NetBeans license, called the Sun Public License ("SPL"), allows anyone to download the NetBeans binary and source files, make changes to those files by adding new features and fixing bugs, and even allows the code to be the basis of commercial applications that are derivatives of NetBeans technology.

The Sun Public License has a few requirements to consider. A few NetBeans

2 http://www.mozilla.org/

modules use other software technologies and products. For example, NetBeans comes with the Apache Tomcat servlet runner to provide an internal Web server. If a product built on NetBeans includes Apache Tomcat, then the SPL for NetBeans requires you to include the Apache license, as it applies to the Tomcat software.

The SPL also requires changes or improvements made to the NetBeans code and shipping with an application to be granted back to NetBeans. This applies only to the NetBeans code and not to your own modules.

The SPL is available with the NetBeans downloads and directly at: http//www.netbeans.org/license.html.

A Framework for Desktop Applications

Desktop application software is rooted in file-oriented metaphors developed in the 1980's by Xerox and Apple, which users have come to expect. The NetBeans platform is designed to be a framework for building file-oriented applications. Desktop applications have open, save and new commands. Once files are open, desktop applications employ the familiar desktop metaphors of a clipboard - with cut, copy, paste, as well as visual editing tools to manipulate file contents.

With the NetBeans software all the expected desktop metaphors are present. A menu bar offers a File drop-down menu to browse directories and open files. Files appear with icons indicating their content. An opened file is viewed and manipulated according to its file type. Java source code appears in a source code editor, while a GIF image appears in an image viewer. Manipulated files display a "modified" icon and the software asks to save the file when the file is to be closed. Contents of one file may be copied onto a clipboard and pasted into a second file. Toolbars display small icons to perform actions. For example, the Compile toolbar button launches a compiler. A built-in configuration dialog empowers users to make changes to the desktop components to match their style and needs; a flexible system for managing default, per-project and per-user settings allows application developers to ship defaults appropriate to that application, encluding enhancing or hiding the basic settings provided by the framework.

The NetBeans IDE platform is a virtuoso desktop application itself. With just a few branding changes, Sun takes the NetBeans platform and tools modules and markets Sun ONE Studio, a powerful desktop integrated development environment to build Java applications. Sun ONE Studio is very popular with developers. As a desktop application, Sun ONE Studio is immediately useful because NetBeans implements the desktop metaphor so well.

Any desktop application will need components to implement the desktop metaphor. Using the NetBeans platform can give a developer a huge lead in the time-to-market.

The NetBeans Environment from a Developer Perspective

Applications typically create, open, modify and save files in one or more steps. In an ideal design, the code to present file contents and user controls is implemented separately from the application's business logic. The business logic manages state, privileges, actions and results in an application. NetBeans is implemented as a core runtime and a set of modules that together supply presentation and user controls. Adding custom business logic is accomplished by writing NetBeans modules that plug into and leverage the core components through the NetBeans Open APIs. For most components, module developers will write to abstractions presented by the APIs. For example, in the NetBeans paradigm, one does not code menu items per se, but rather Actions. The framework will handle the presentation of these items, saving time and effort, but the presentation can be customized if necessary.

NetBeans comes with all the needed components to build high quality applications featuring modern graphical desktop user interface elements. Advanced tools, such as FTP, CVS and database clients are also available as part of the free NetBeans source code. These sources include more than 40 components in the following general categories[3]:

- User interaction components - menu bars, tool bars, status displays, file systems, workspaces, tabbed-window displays, properties, alerts and dialog boxes, printing controls, search, replace, output consoles.

- Tools components - FTP client, CVS[4] client, browser, localization wizard, Ant[5] utility, Python (Jython) scripting, and JAR packager utilities.

- Help components - JavaHelp™ integration for online help including context help and a Javadoc™ utility.

- Storage access - NetBeans is not limited to local disk storage, but can be extended to support arbitrary storage of file-like data.

- Utilities - Module download and installation utility, set-up wizard for NetBeans itself, Linux shell scripts to launch NetBeans and a Windows executable to run NetBeans.

Since NetBeans is designed as a framework, the application developer has many choices in using or reusing existing components or building new components

3 A detailed list of NetBeans modules is at: http://www.netbeans.org/devhome/modules/index.html

4 Concurrent Version System (CVS) is a popular system for teams of engineers to work on a common code base when building and maintaining software applications. Details are at: http//www.cvshome.org/

5 Ant is a popular utility for automating the software compilation and linking process. Details are at: http://jakarta.apache.org/ant/index.html

and functions by using the NetBeans Open APIs. The Open APIs are the public interface to the NetBeans runtime, and thus the toolset available to developers of extensions.

The components are powerful, fully functional and complete by themselves. NetBeans handles component deployment by packaging the Java code of a component with a deployment manifest describing the components interfaces and dependencies. The combined package is called a module.

As an example, ECSI , a company providing mining, geology and resource evaluation software to the mining industry, redesigned their mining application around NetBeans Open APIs to produce the MINEX 5 application. Figure 2 shows the resulting graphical application.

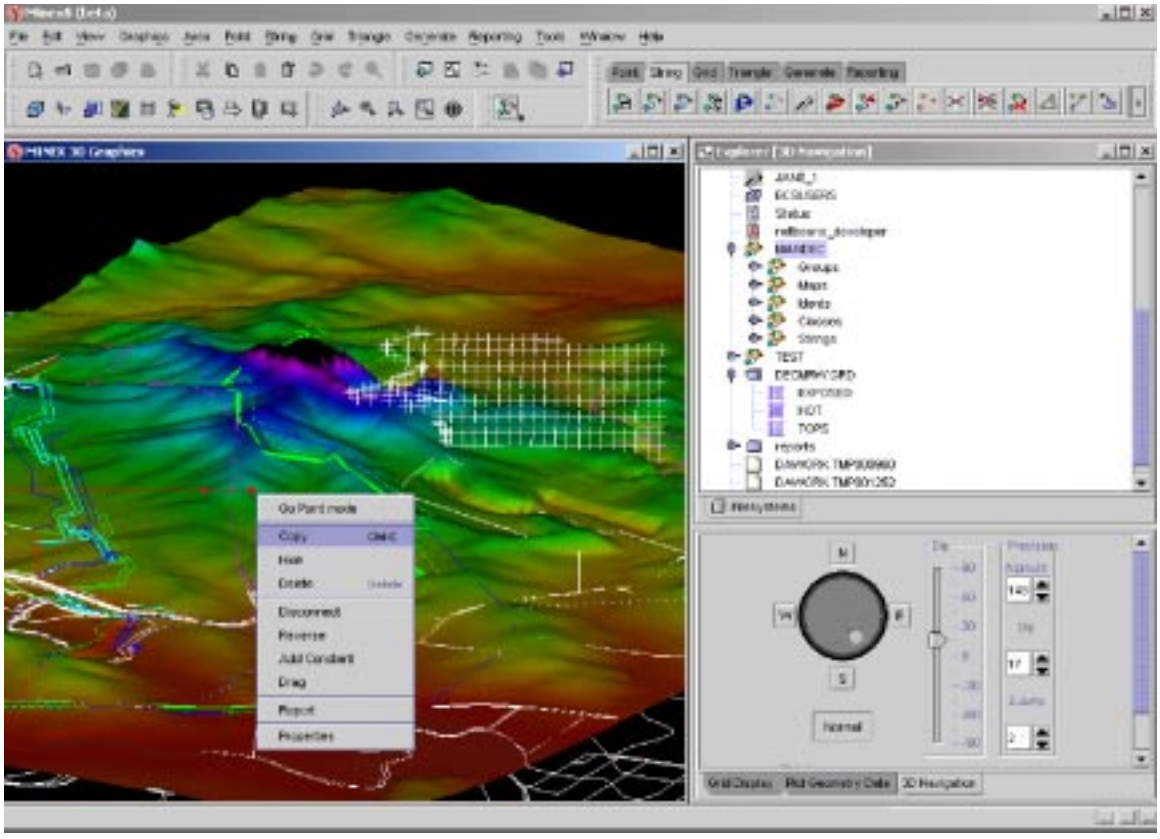

Fig. 2 - ESCI MINEX 5 application based on NetBeans

Using NetBeans, the MINEX 5 interface is customizable, enabling users to change the environment for themselves or a group of users on the fly. Floating windows may optionally be docked to the central window that shows a graphical map of terrain. Now deployed as a Java application based on NetBeans framework, MINEX 5 delivers the look and feel of a modern GUI.

The ECSI developers leveraged NetBeans framework's existing components to

do cascading menus, floating icon-based toolbars, floating help windows, hotkey shortcuts, a file system explorer pane, text editor, image viewer, file search function and a built-in Web browser.

Module Basics

NetBeans technology is based on a thin core that is responsible for the basic aspects of a desktop application. The core provides generic presentation functions, including windows, event mechanisms, file system interfaces and a simple data persistence system. The core is easily extended with plug-in modules written to the Open APIs. Desktop application developers will rarely, if ever, make changes to the core as the bulk of an application's functionality is implemented in a module running above the core.

NetBeans technology's architecture was very well planned for module development. All of the NetBeans APIs stand alone - classes do not have callbacks between modules and the core. Modules may also use the standard Java libraries. The NetBeans runtime environment allows functionality to be added discretely and painlessly without recompiling or restarting the NetBeans core.

Some modules have turned into superstars. For example, Figure 3 shows how the Editor module displays a highly versatile and functional text and code editor. The Editor module enables users to edit text file contents, view syntax-highlighted source code, and manipuate file contents.

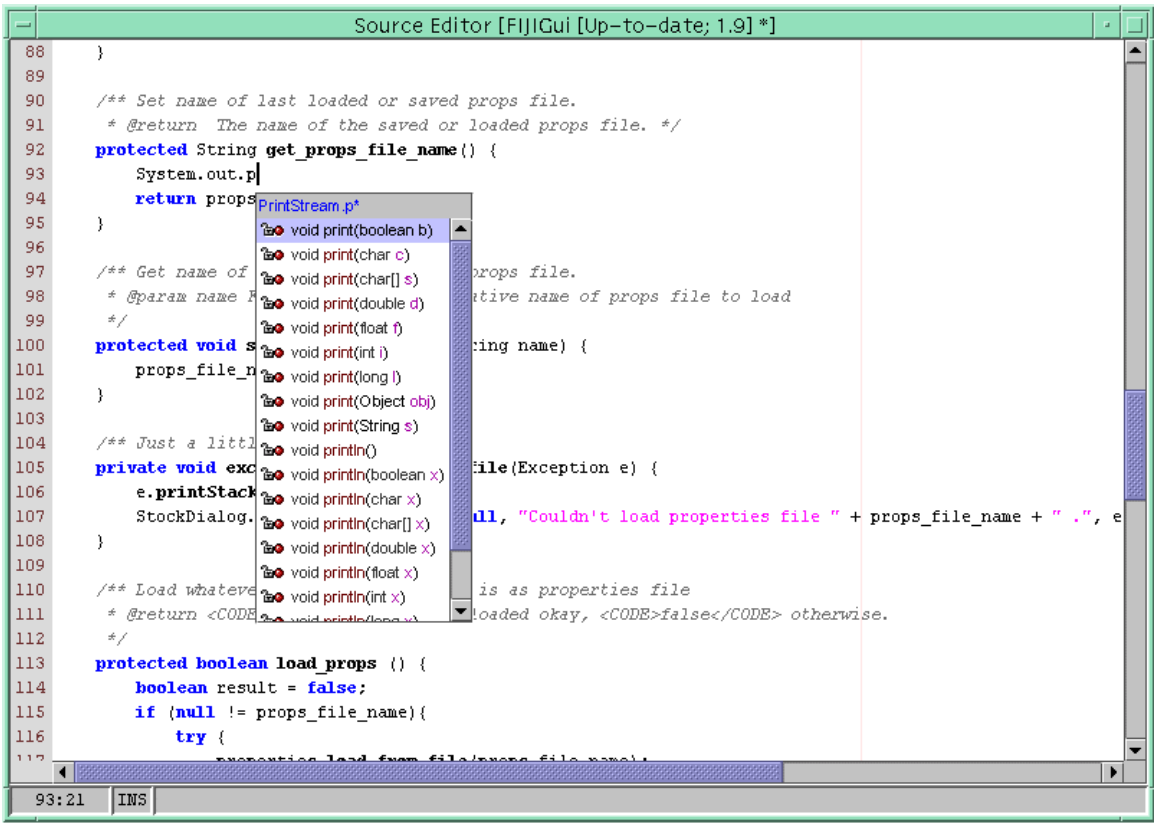

Fig. 3 - The Editor module

NetBeans core components like the Explorer component can be deployed as a module (as shown in Figure 4) or even a stand-alone library in non-NetBeans applications. The Explorer component appears as a window. The Explorer enables users to mount file directories, JAR and Zip archive files and use CVS commands on files. Once a file system is mounted, the Explorer provides a graphical interface showing the hierarchy of directories and files. Individual nodes in the Explorer interface carry actions. For example, right click on an Ant build script and a pop-up menu appears offering to execute the Ant script.

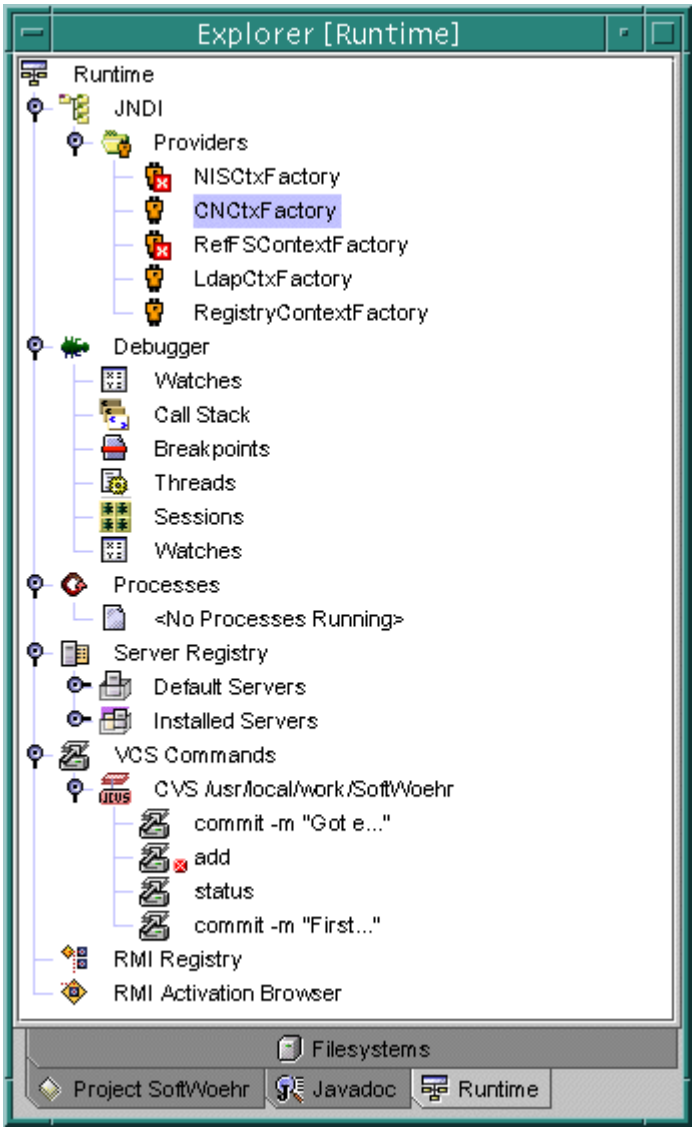

Fig. 4 - The Explorer component

Other modules are sleepy wonders. For example, the Scripting module utilizes the basic text editor module, debugger module and adds the Python[6] scripting language. The NetBeans IDE becomes an excellent environment for developing Python scripts.

The NetBeans source base includes more than 40 useful modules, and developing a new desktop application is accomplished by using the NetBeans Open APIs to write new modules, supplying branding information and packaging the application for distribution. Application developers ship only those modules needed by the finished application.

---

6 The NetBeans Scripting module bundles Jython, the Java implementation of Python. More details are at: http://scripting.netbeans.org/ and http://www.jython.org/

Introduction to the NetBeans Open APIs

The NetBeans platform provides a set interfaces for interoperation with its core components and other modules. An understanding of the APIs and the deployment mechanism is important to building a module.

Each module is deployed as a JAR file containing a manifest describing how to install the module and the module's dependences and the Java classes that implement the module's functions. The Java classes may make calls to other modules and components and also to any Java API found in the Java platform.

The NetBeans Web site, on-line help systems and developer community often expect module developers to have JavaBeans™ component experience. Helpful information on learning JavaBeans conponents is found on the JGuru[7] and JavaWorld[8] Web sites.

The NetBeans Open APIs reside in the Java package org.openide and its subpackages, but there is nothing tying these APIs to implementing only development environments. The Open APIs work equally well to develop a graphics editor, a spreadsheet, and other desktop applications.

The Open APIs are grouped into 16 packages, including:

- The Modules API handles loading modules, versioning and cross-compatibility checks. Also available is an AutoUpdate module which will allow application developers to publish updates to their application and download and install them directly into NetBeans.
- The Services API describes how to handle execution of files and other functionality that involves data processing. For example, the scripting module defines how to edit and run Python scripts.
- The Filesystems API delivers interfaces to access files, JAR and Zip archives and virtual XML file systems. It can be extended through a Service Provider Interface to support other forms of storage, such as FTP or CVS servers or databases.
- The DataSystems API is used to identify data types and the actions performed on the data types. Data types are registered with the DataSystems API to generically tell NetBeans how to perform operations on data objects.
- The Explorer API provides both a user interface and methods to operate on hierarchical trees of objects. The objects may be file system directories, configuration settings, a remote database or a thread stack in the debugger, or anything else that can be organized into a hierarchy.

7 http://www.jguru.com/
8 http://www.javaworld.com/

- The Nodes API provides a generic, content-neutral definition of a hierarchy. For example, a Node may represent a file stored in a directory or a breakpoint in the debugger. Node objects define user options to control the object, including cut, copy, paste operations, custom actions and context sensitive help. Explorer components provide visual rendering of Nodes.
- The Actions API provides context sensitive commands that may be issued using the toolbar, and menu elements in the graphic interface or the keyboard.
- The Window System API controls the graphical container elements of the desktop application, and provides options such as window docking and varying window styles. NetBeans supports the Multiple Document Interface (MDI) style where one large window contains multiple smaller windows. NetBeans also supports the Single Document Interface (SDI) where many windows may be opened on the desktop.The Projects API deals with user-level projects. A project consists of a particular set of files, environment settings and windows.

The additional APIs are more specific to developing an integrated development environment. These modules provide sophisticated file editors, program debuggers and powerful methods to establish the context of an IDE's operations.

- The Compiler API enables external compilers to be accessed to build software projects.
- The Execution API provides control over running a user's Java program, including running applets, applications, or non-standalone components such as JPanels. This API also enables implementation of custom class loaders.
- The Debugger API specifies how a debugger should interact with the IDE.
- The Editor API provides access to the basic functionality of the Editor and its support for different content types. It provides the ability to insert custom editors conforming to the Swing Editor Kit conventions.
- The Java Hierarchy API encapsulates the result of parsing either Java source documents or Java class files. A structured representation of the Java class objects and their members that allows module code to manipulate Java classes programmatically.

The Open APIs are a powerful, pluggable mechanism for building NetBeans modules. Building a new module requires several steps, beginning with getting a development environment to construct a module.

Installing the NetBeans IDE

Download the NetBeans IDE from the NetBeans Web site[9]. NetBeans offers a stable release and a development build on the download page. NetBeans also offers a choice of precompiled builds or the source code itself. For the purposes of this paper, please download the current stable release. NetBeans requires Java technology version 1.3 or higher, which is available from java.sun.com for Linux, Solaris™ Operating Environment and Microsoft Windows. Install the NetBeans IDE by unzipping the downloaded archive. The archive is configured to create a NetBeans directory by default.

Run the NetBeans IDE using a launcher (runide.exe for Windows) or shell script for Unix® found in the netbeans/bin directory. Once it is running, select Update Center from the Tools menu. Following the steps and instructions in the wizard, download the OpenAPI support module. While not absolutely necessary to develop modules, this module contains numerous tools to make developing modules faster and more pleasant, and includes additional documentation for the Open APIs.

Finally, create a new Project by using the Project Manager command in the Project drop-down menu. Name the project.

Building A Module

To illustrate a method of building NetBeans modules, this paper describes another open-source project that uses the NetBeans platform for building a graphical application. See the resources section at the end of this paper to download the sample module source code.

The TestMaker[10] project is an open-source utility to create intelligent test agents that test Web services for scalability and performance. TestMaker uses Python as a scripting language to drive a library of test objects that implement HTTP, HTTPS, and SOAP protocol handlers.

TestMaker's core business logic plugs into NetBeans as a module. The module leverages NetBeans technology's ability to manage, edit and manipulate script files and to run those files in the built-in Python[11] interpreter. The TestMaker module extends and changes the NetBeans platform's existing functions, rather than reinventing what the platform already delivers.

Using the Open APIs

The TestMaker module implements a manifest file and *layer* file to install its custom functions into NetBeans framework. If needed the module could also perform additional customization calling the NetBeans Open APIs at runtime.

---

9 http://www.NetBeans.org

10 Complete details on TestMaker are at http://www.pushtotest.com.

11 The NetBeans scripting module actually bundles Jython, a Python interpreter written entirely in Java. For details see: http://www.jython.org

The directory structure for the module is important since the manifest file must fully qualify where to find the module's objects. The module is shipped as a standard JAR file so the manifest.mf file could be anywhere in the module's directory structure. The example module uses this directory structure:

```
com
  -> pushtotest
     -> modules
        -> testmaker
           Manifest.mf (Manifest file)
           AgentDataLoader.java
           AgentDataNode.java
           Bundle.properties (text strings and attributes)
           etc.
  -> build.xml (ant file)
```

The Manifest.mf file tells NetBeans where to find a module's objects and the dependencies the object has on other modules.

```
Manifest-Version: 1.0
OpenIDE-Module: com.pushtotest.modules.testmaker
OpenIDE-Module-Name: TestMakerModule
OpenIDE-Module-Short-Description: Implements functions for the TestMaker
         functional and load testing utility

OpenIDE-Module-Layer: com/pushtotest/modules/testmaker/Layer.xml

Class-Path: ext/djava.jar ext/bsh-1.01.jar ext/jython.jar

Install-Before: org.netbeans.modules.text.TXTDataObject
Install-After: org.netbeans.modules.java.JavaDataObject
```

The manifest syntax provides instructions for installing modules into the NetBeans platform. The manifest is processed from the top to the bottom. The OpenIDE-Module-Layer command tells the platform to look for additional configuration parameters in the layer.xml file. The Install-Before and Install-After commands identify the loading order of this module. Later in this paper we will look at highly complex modules and the *layers* file approach to configuring modules.

Adding support for new file types to NetBeans

In the example module, the AgentDataLoader object registers methods to find and operate on the module's script file extensions and data types. When the file ends in *.a* then NetBeans will use the module to handle the file. Any number of file extensions may be mapped to files with this method. More complex mapping of file data types to DataLoader objects is also possible using MIME types.

```
   /** Identifies .a TestMaker script files */
  protected void initialize () {
       ExtensionList ext = new ExtensionList();
       ext.addExtension("a");
       setExtensions(ext);
```

```
    }
```

AgentDataLoader establishes a list of actions that are available to the user for the script files it recognizes.

```
    /** Gets default system actions. Overrides superclass method. */
    protected SystemAction[] defaultActions() {
        return new SystemAction[] {
            SystemAction.get(OpenAction.class),
            SystemAction.get(ExecuteAction.class),
            SystemAction.get(FileSystemAction.class),
            null,
            SystemAction.get(CutAction.class),
            SystemAction.get(CopyAction.class),
            SystemAction.get(PasteAction.class),
            null,
            SystemAction.get(DeleteAction.class),
            SystemAction.get(RenameAction.class),
            null,
            SystemAction.get(PropertiesAction.class),
        };
    }
```

The defaultActions method instructs NetBeans to handle file open, execute, cut, copy, paste, delete, rename and view properties functions. Menu dividers are inserted using nulls. NetBeans uses the DataTestMakerer object to handle presentation and functions. For example, right-click on a script file and NetBeans displays a pop-up menu with the choices defined in the DataTestMakerer object.

DataTestMaker handles the recognition of file types in NetBeans framework. DataTestMakerer is a DataObjects factory. DataObjects represent individual files and provide programatic access to their content.

```
    public AgentDataTestMaker() {
        super("com.pushtotest.modules.load.aDataObject"); // NOI18N
    }
```

AgentDataTestMaker instantiates a DataObject and consequently a Node user-presentation object when the user creates a new script file. The DataObject also establishes a Cookie object for the script file type. Unlike a Web browser's cookie file, NetBeans Cookie objects handle actions related to the file type - they are a generic way to look up arbitrary functionality (such as saving or opening files) on a DataObject or Node. In the module example, AgentDataObject creates a Cookie for script files, which opens a text editor and instantiates an EditorSupport object. EditorSupport performs actions like opening and saving a file.

AgentDataObject also instantiates an AgentDataSupport object to handle the Execute function. When the user chooses to execute an open script, the start() method in AgentDataSupport, as defined by AgentDataObject, finds the path and file name of the script, and invokes the Python interpreter to run the script.

Python output is logged to the output window.

```
/* AgentDataSupport: Starts the class. */
public void start ()
{
    FileObject theScriptFile = entry.getFile();
    File theFile = FileUtil.toFile(theScriptFile);

    PythonInterpreter interp = new PythonInterpreter();

    try
    {
        interp.execfile( FileUtil.toFile(theScriptFile).getPath() );
    }
    catch ( PyException e )
    {
        System.out.println( "Agent Error" );
        System.out.println( "Found on line " + e.traceback.tb_lineno );
    }
}
```

Understanding Which APIs To Use

All of these objects may appear confusing and arbitrary at first. Keep in mind that every module uses the same API set. Once mastered, a module developer has the ability to build any application through these same APIs. Also, NetBeans comes with 40 modules whose source code might act as sample code to learn from. After downloading the NetBeans source code, the Scripting module is found at: netbeans-src/scripting. Feel free to download, copy and modify the NetBeans source code to build your better mousetrap.

To better understand the objects used in the example module, here is a summary of the classes:

- AgentDataLoader - Register the file extensions, icons, object type and tasks to work with script files. Also instantiates a DataObject.
- AgentDataObject - Provides main functionality for the AgentDataLoader, including executing the script.
- AgentEditorSupport - Uses a NetBeans Cookie object to handle saved-status for a file. Cookie objects are powerful ways to represent actions. In the example, the Cookie object is used later to see if the editor should save a modified file.

When a module includes these objects, any user running the NetBeans platform with the module installed is able to open an existing script, edit the file contents, save the file and execute it.

Here is another way of looking at the Open APIs:

- Nodes - JavaBeans components or other property containers within a hierarchical tree.

- Cookies - add behaviors to objects and nodes. For example, a Cookie object attached to the node for a script file implements the Execute action for a script file.

- Actions - user interactions, including toolbars and menus. Actions are almost always singletons, so their state is the same regardless of where they are called from - pop-up menu or window object.

- Layers - instructions for installing new objects and behaviors using XML notation. The Open APIs give access to all of NetBeans through Java programming. As an intermediate step, layers enable declarative, non-programmatic installation of objects into NetBeans framework and are much easier to configure and maintain than writing Java installation code. A layer is a "virtual XML filesystem" which adds "files" to NetBeans framework's internal system filesystem. These pseudo-files may in turn be factories for Java objects, via semantics defined in the Modules API.

Next, imagine adding a new toolbar command that runs the currently open script. This is the place for an Action object. The Layer.xml file adds an Action object. Action objects tell the IDE the special commands that may be run for a given object type. An example of this is an icon on a toolbar which runs the currently open script. This Action registers the EvaluateAction class as a user-invokable command.

Examining the EvaluateAction class shows how the performAction() method will find the Node and Cookie objects assigned to the target script document type.

```
/** Performs action based on specified nodes. Implements
         superclass abstract method. */

protected void performAction(Node[] nodes) {
. . .
    AgentDataObject ldataObject =
         (AgentDataObject)nodes[0].getCookie(AgentDataObject.class);
    if(ldataObject != null) {
        ldataObject.evaluate();
    }
    else
    {
        EditorCookie editorCookie =
         (EditorCookie)nodes[0].getCookie(EditorCookie.class);
        if(editorCookie == null)
            return;

        JEditorPane[] panes = editorCookie.getOpenedPanes ();
        if(panes == null)
            return ;

        Component component = panes[0];
```

```
        while(component != null) {
            component = component.getParent();
            if(component instanceof EvaluateConsole) {
                ((EvaluateConsole)component).evaluate();

                break;
            }
        }
    }
}
```

Clicking the toolbar button issues a call to execute the above performAction() method of EvaluateAction.

The remaining classes enable NetBeans to create a new script file. Specifically, the Layer file contains instructions that will drive a Wizard. To see a Wizard in action choose New from the File drop-down menu. Each of the file types here is implemented in its respective module and extends the Wizard using a Layer file.

Module Development Lifecycle

NetBeans source code comes with Ant, a popular open-source make/build utility written in Java programming language. The Open APIs Support module installs a special extension to Ant to temporarily install compile modules without having to restart NetBeans, so modules may be tested within the development environment.

The build.xml file is an Ant script that handles compiling the Java code and packaging a JAR file to be loaded as a module by NetBeans framework. The Ant build script calls the special module installer when the script encounters an nbinstaller element.

```
<target name="install" depends="init, compile, package">
  <nbinstaller module="load.jar" action="reinstall" />
</target>
```

Finding Additional Help

In addition to this paper, the OpenAPI JavaDoc provides much information and detail on these major portions of NetBeans framework. For example, the Modules section of the org.openide.modules package contains a long introduction to the possibilities of modules.

Creating A Branded Distribution

In most cases it is possible to customize the branding - the colors, styles, logos and layouts - of NetBeans modules without coding. Most of NetBeans framework's configuration is managed in files. For example, the menu bar in the Main Window is set-up using from a Layer file.

The NetBeans platform's branding mechanism lets you patch things too. The

XML layer entries that may accompany a module's manifest are loaded when the module loads. These layer entries include branding variants that are loaded with the module - for example, a layer entry may insert a new menu item specific to the module. All the variants are merged together at load time, with the more specific brandings taking precedence. The merging is done by org.openide.FileSystems.MultiFileSystem and follows the same behavior used to merge project and user directory customizations with module-supplied defaults.

Using configuration files has another benefit: maintainability. The most maintainable method to extend NetBeans is to use configuration files and declarations rather than writing code. For example, adding a new menu to the menu-bar using configuration files is easier to maintain than writing a Java class that programmatically adds the new menu. The module examples in this paper use configuration files.

As an example of using configuration files, the splash screen that appears when an application built on the NetBeans platform runs may be configured by changing an image file located in /org/netbeans/core/resources/splash.gif. Additional branding details are found on the Localization[12] section of the NetBeans web site.

The NetBeans platform provides the interface elements using a "salad bar" approach. Application developers provide default arrangements for components, and users may customize their environment to suit their needs. For example, windows may be visually glued to each other by the user. It should not be obvious to the user which interface elements belong to which modules. In creating the TestMaker example, the original implementation created a script editor window with an output pane. The more efficient mechanism was to use the existing script editor and output window elements, and then use NetBeans framework's configuration options to glue them together into one window. A module could also configure the windows to appear already glued and let the user unglue them.

Understanding Module Dependencies

The NetBeans IDE weighs in at 16 Megabytes compressed, including all of the modules in a standard IDE distribution, most of which will not be needed by your application if it is not an IDE. To slim it down, NetBeans allows modules to be removed. NetBeans is distributed with Ant build scripts for the core and each module. One way to go is to build only the modules needed. Another way is to start with a clean install of NetBeans and delete the unnecessary JAR files from the netbeans/modules directory. Of course, the core modules and utility modules have dependencies. Check http://openide.netbeans.org/ web site for information on the dependencies between modules.

Module manifest files support dependency checking using the Java Versioning

12 http://www.netbeans.org/i18n/index.html

Specification. In the TestMaker example, the manifest tells NetBeans framework that if the other loaders are installed to use the order defined in the Manifest.

```
Install-Before: org.netbeans.modules.text.TXTDataObject
Install-After: org.netbeans.modules.java.JavaDataObject
```

Resources

The NetBeans open source project offers more than 40 modules, with hundreds of APIs and more than 800,000 lines of code. Learning NetBeans may appear a daunting challenge, but one well worth it. Due to its segregation into APIs by function, it is not necessary to have digested the entire codebase or API set in order to be productive building modules. Additionally, all development and design are done on public mailing lists, in accordance with open source practice. The NetBeans open source community spends a huge amount of energy each day to offer assistance, resources, code samples and support to developers. These resources are aimed at helping anyone using NetBeans technology to develop desktop application software.

Web sites

These Web sites provide useful information while developing modules for the NetBeans platform:

JGuru[13] - frequently asked questions, tutorials and example code for Sun's Forte for Java, a Java technology integrated development environment (IDE)

Java World[14] - an online magazine with articles and sample code for JavaBeans components and Java technology in general.

NetBeans[15] - the mother ship. The NetBeans web site offers several sections specific to developing desktop application software using Open APIs[16]

Google[17] - online search engine that has a good index of NetBeans-related articles, including resources that are hard to find on the NetBeans.org web site.

Mailing lists

NetBeans operates 3 email mailing lists to answer questions regarding NetBeans.

- nbdev@netbeans.org - serves the needs of developers working with the Open APIs. Usually highly technical in nature.

13 http://www.jguru.com
14 http://www.javaworld.com
15 http://www.netbeans.org
16 http://apisupport.netbeans.org/
17 http://www.google.com

- dev@openide.netbeans.org - ongoing design and discussion of the Open APIs. A good place to ask questions about usage of a particular Open API.
- nbusers@netbeans.org - answers user questions about using the NetBeans IDE.
- nbannounce@netbeans.org - used to announce new versions and deliver the NetBeans weekly newsletter, a volunteer-run publication covering announcements and interesting threads from the mailing lists for the week. The mailing list is published every Monday.

The email lists require a free subscription to post questions and receive email messages. The email lists are also served via NNTP News readers at news://news.netbeans.org. Subscribe to the email lists at the Mailing List Page[18] on NetBeans.org.

Sample Code

The TestMaker utility discussed in this paper is available for free download from the PushToTest[19] web site.

About The Author

Frank Cohen is the principal architect for three Internet systems: Sun Community Server™, Inclusion.net and TuneUp.com. These are Internet messaging, collaboration and e-commerce systems, respectively. Each needed to be tested for performance and scalability. Frank developed TestMaker, an open-source tool featuring test objects and a scripting language. Sun, Inclusion and TuneUp put TestMaker to work testing for performance and scalability. TestMaker is available for free download at http://www.pushtotest.com.

18 http://www.netbeans.org/devhome/community/lists.html
19 http://www.pushtotest.com