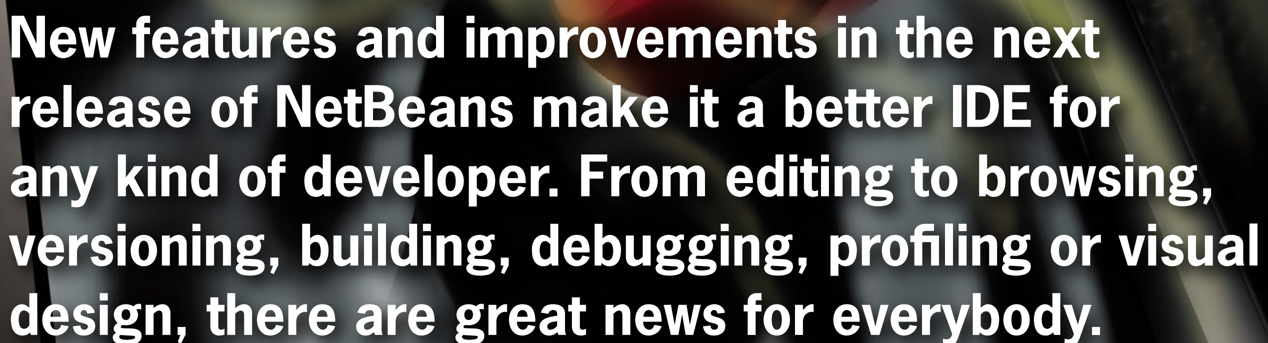
NetBeans 6.0

New Core Features in Depth

Osvaldo Doederlein



New features and improvements in the next release of NetBeans make it a better IDE for any kind of developer. From editing to browsing, versioning, building, debugging, profiling or visual design, there are great news for everybody.

It's that time again. A major, dot-zero release of NetBeans will be available soon – about a year and a half after 5.0, which introduced significant new features like the Matisse GUI builder, and extensive improvements in CVS integration, web services and module development, to cite but a few. In contrast, version 5.5 focused outside the core IDE by supporting several new Packs that increased NetBeans' overall functionality to a level still unmatched by any other open-source IDE. Now, is NetBeans 6.0 worthy of the bump in the major version number? You bet it is, and in this article we'll look at some of the most important and interesting new features in the core IDE.

Javac-powered

Let's begin by looking not at an end-user feature but at a core IDE technology that provides the foundation for many enhancements. Past releases of NetBeans, like many other programming tools, contained custom code to parse Java sources and assist in code understanding and manipulation tasks (like refactorings, hints and fixes, outlining, etc). The result was sometimes limited functionality: simple highlighting, non-bulletproof refactorings, and the lack of support for features like code completion everywhere Java code appears.

The obvious solution would be reusing the mature technology of the javac compiler to do all Java source processing. But javac was not designed to support the requirements of a modern IDE: it was written and tuned for batch execution, and to accept as input full compilation units, perform a complete compilation and produce *.class* files as output.

IDEs have very different requirements, among which the most critical is working in memory only. Suppose that after each character you type, the IDE wants to analyze the entire class again so it can update syntax error indications, perform highlighting, and provide other features that depend on the code structure. One option would be to write the editor's current content to a temporary file, invoke javac and parse the resulting *.class* files. But this would be very inefficient.

A much better solution is to call javac in the same process (as a local library), then pass the current sources as an in-memory parameter and receive in return the data structures containing the same information that would be present in the class files (which wouldn't need to be created). Up to Java SE 5, this solution would be possible, but only using the proprietary – and often unstable – internal APIs of a Java compiler.

This situation changed with Java SE 6, which introduced JSR 199 (Java Compiler API) and JSR 269 (Pluggable Annotation Processing API). The Java Compiler API enables tight and efficient integration with javac (and other Java source compilers), and JSR 269 – although initially designed for annotation processing – provides a source-level equivalent of reflection metadata. Working together, these new APIs allow IDEs and other tools to dig deeply into the structural information that javac extracts from source code. Additionally, javac's implementation was enhanced and tuned for embedded and interactive use.

NetBeans was heavily updated to integrate with these new capabilities, enabling many improvements in the IDE (discussed below). The changes also promise future benefits: when Java SE 7 comes out with a new set of language enhancements, you should expect NetBeans' toolset to catch up very fast.

A new editor

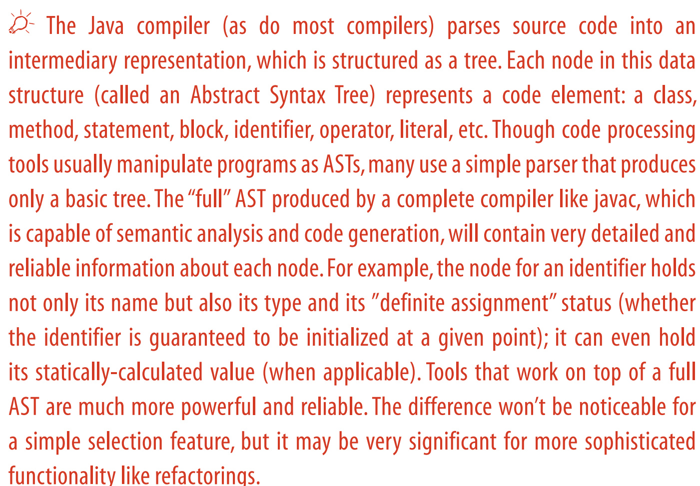
Common sense says no product can be perfect in everything it does, but NetBeans is getting closer each day. Historically, NetBeans users have been proud of the IDE's complete coverage of Java platforms from ME to EE, its support for effective GUI building, and its intuitive UI and open architecture. On the other hand, the IDE lagged in certain areas, like in the code editor or refactoring. This could put off programmers very focused in source code… types who'll pick *emacs* over visual designers any day. Well, these problems are no more with NetBeans 6.0.

jackpot.netbeans.org
The Jackpot project.

www.netbeans.info/downloads/dev.php
NetBeans 6.0 development builds.

AST-based selection

Selecting words or lines is good enough for text editors, but when working with sources you often need to work with ranges of text that form coherent pieces of code. Say you want to copy all the code inside a **for** loop body[1] in order to paste it in another loop with similar logic. Just place the cursor in any blank position inside the loop body, press *Alt+Shift+Up* and you're done. The editor selects the innermost range of text that includes the cursor position, and delimits a node of the source's Abstract Syntax Tree.



The Java compiler (as do most compilers) parses source code into an intermediary representation, which is structured as a tree. Each node in this data structure (called an Abstract Syntax Tree) represents a code element: a class, method, statement, block, identifier, operator, literal, etc. Though code processing tools usually manipulate programs as ASTs, many use a simple parser that produces only a basic tree. The "full" AST produced by a complete compiler like javac, which is capable of semantic analysis and code generation, will contain very detailed and reliable information about each node. For example, the node for an identifier holds not only its name but also its type and its "definite assignment" status (whether the identifier is guaranteed to be initialized at a given point); it can even hold its statically-calculated value (when applicable). Tools that work on top of a full AST are much more powerful and reliable. The difference won't be noticeable for a simple selection feature, but it may be very significant for more sophisticated functionality like refactorings.

Pressing *Alt+Shift+Up* again expands the selection to the next outer node, in this case the complete **for** statement; then a new keystroke may select the entire method, and so forth. *Alt+Shift+Down* will retract the selection to an inner node. **Figure 1** shows this feature being used to select a multi-line statement easily and precisely. I bet you will quickly be hooked on this feature and forget about all the other selection shortcuts! There's nothing like a code editor that groks *code*, not text.

Semantic highlighter

The editor's syntax highlighter was promoted to a semantics-aware highlighter. It can apply styles based not only on the types of tokens (like identifiers, operators or comments), but also based on different meanings that akin tokens may have – for instance, an identifier may be a class name or a local variable name, a parameter, a constant field, etc.

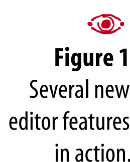
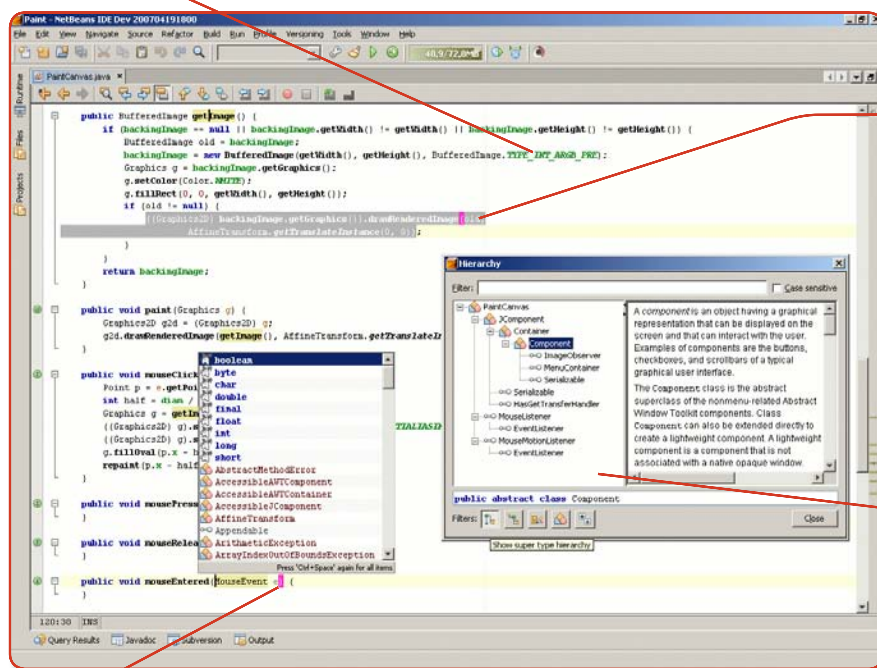


Figure 1
Several new editor features in action.



[1] Depending on your bracing style, this may not be as easy as selecting a few full lines. There are many other examples, like selecting a complex expression that spans multiple lines.

One benefit of semantic highlighting is that it helps you take extra care when assigning to **static** fields (since many thread-safety and memory-leak bugs involve statics). **Figure 1** shows this off; notice that static fields (and references to these) appear in italics.

There are other powerful uses for the new highlighting engine:

- *Identifying usages* – Select any identifier, and the editor highlights all its uses in the same compilation unit. Again, **Figure 1** exemplifies this: clicking on a method name, all invocations to it are highlighted.
- *Flagging "Smelly code"* – The new editor highlights unused variables and imports, as well as usage of deprecated classes and methods. You don't need to perform a build or run a code lint tool to detect these simple (but frequent) problems anymore.
- *Exit and throw points* – Selecting a method's return type will highlight all **return** statements. Selecting an exception in the method's **throws** list will flag all **throw**'s of that exception type. All invocations to other methods that may throw the same exception are also flagged.

Better code completion

The bewildering amount of APIs you have to use these days makes code completion one of the most critical features of any modern code editor. NetBeans 6.0 has learned many new tricks here:

- *Keyword completion* – If you've just typed a **package** declaration in a new source file (for example), *Alt+Space* will bring only the keywords that are legal in that position: **abstract**, **class**, **enum**, **final**, **import**, **interface** and **public**. **Figure 1** shows another example: after the opening parenthesis of a method declaration, the preferred completions are all primitive types.
- *Type-based variable names* – Completing at "**ConfigurationFile** _", the editor will offer the variable names **cf**, **configurationFile** and **file**. (I'm using "_" to represent the cursor position.)
- *Generics-aware completions* – When assigning a variable with a generic type to a **new** expression, the editor will offer all compatible types, including generic arguments. For example, at "**Map<String, Integer> m = new** _", code completion lists all implementations of **Map**, each with the same **<String, Integer>** parameters.
- *Annotation-aware completions* – When completing after "**@**", you'll be offered all the annotations that can be used in the given scope. And if the selected annotation requires parameters the editor will provide completions for these too.
- *Passing parameters* – At "**x = m(**_", the top completions will be values in scope that are compatible with **m()**'s first parameter. If the method's parameter names are available and there are variables with similar names in scope, this is used to sort the completions further. You'll also be offered full completions with the parameter list filled with those variables.
- *Common constructors* – When you invoke code completion with the cursor positioned between class members, you'll be offered to create a constructor without arguments and one that receives initial values for all fields (if these constructors don't already exist).
- *Catching exceptions* – Completion at "**catch (**_" will only offer exceptions that are thrown in the corresponding **try** block, but haven't been handled yet by previous **catch** blocks.

New browsing views

The editor introduces several new views for source code browsing. The *Members view* shows the members of a Java type together with their javadocs, making it easy to find a particular method, field or inner class. The *Hierarchy* view shows the inheritance tree of a Java type. **Figure 1** demonstrates this view; notice the filter buttons that let you toggle between supertypes or subtypes and between simple and fully qualified class names. You can also choose whether or not to show inner classes and interfaces.

The *Declaration* view summarizes the declaration of the selected Java element (type, method or field). Despite its name, this view also shows the inspected element's source code if it's available. The Declaration View is especially useful when invoking code still under

development, not yet documented with javadoc. Finally, the *Javadoc* view shows the javadocs for the selected Java element.

Editable Diff and Inline Diff

The editor's improved architecture makes it easier for various features that handle source code to integrate editor functionality. This is noticeable in the new Diff (opened, for example, by selecting a source file and choosing *Subversion>Diff*). When it's showing a local file, the right pane is editable, providing the full set of editor features – semantic highlighting and code completion included.

The new Diff adds other interesting tricks, like one-click merging and word-level diff (if a single word is changed in a line, only that word is highlighted). Check out these improvements in **Figure 2**.

You can also enable an Inline Diff feature, which creates a Diff sidebar, highlighting updated sections of a versioned file. The sidebar lets you visualize or rollback changes, and open the full Diff view.

Javadoc hints

You always document all your code, right? Well, if you don't, NetBeans will complain about missing and incorrect javadoc tags. The IDE can help you with automatic fixes that add the missing tags, only asking you to fill in the blanks. And while you're doing that, you can use the new Javadoc view for convenient previewing.

Javadoc checking is active by default, but it's not intrusive: the editor will report missing javadoc tags just for the selected line; only incorrect tags will be reported everywhere. You can customize these and related options through *Tools|Options>Java Code>Hints*.

Other features

The new editor and its framework include other general features, like reusable editor tabs. These are useful for the debugger, to avoid cluttering your environment with editors opened by breakpoints or step-into's. There's also a new *Generate Code* dialog that automates the creation of constructors, getters and setters, **equals()** and **hashCode()**, and delegate methods.

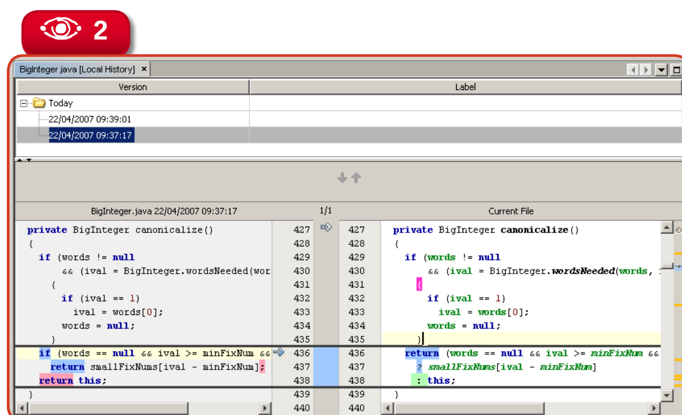


Figure 2
The Local History and the new Diff: editing capability, semantic highlighting and word-level diff.

Refactoring and Jackpot

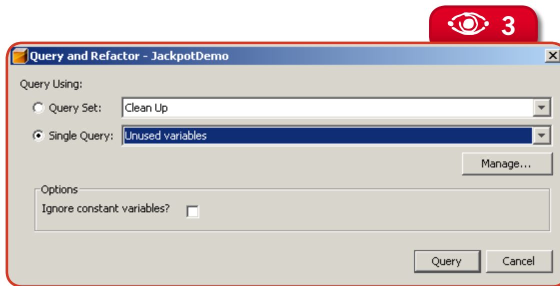
NetBeans 6.0 improves the existing refactoring support extensively. There is a new internal language-independent refactoring API that will allow implementing refactorings for code other than common *.java* sources (e.g., XML or JSF files). The new API also allows Java refactorings to precisely update dependent non-Java elements. This should make the current refactorings safer and easier to use.

The big news here, though, is the breakthrough new technology from project Jackpot, which has been available for some time but is only reaching maturity now. With its inclusion in NetBeans 6.0, Jackpot will be promoted to a standard feature and be more closely integrated with the IDE.

You may have heard that Jackpot is a new refactoring tool, but this really doesn't make it justice. Jackpot is actually a comprehensive framework for general code understanding and manipulation. You can use it as a replacement or foundation for several kinds of features: refactoring support, advanced searching and browsing, quality inspection, macro-like automation of complex editing tasks, and more.

Using Jackpot

Before taking a more in-depth look at Jackpot, let's show how easy it is to use. The new



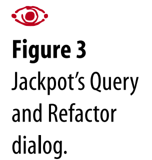


Figure 3 Jackpot's Query and Refactor dialog.

Query and Refactor command will show a dialog like **Figure 3**, where you can pick a Jackpot query or query set. Some queries have options that you can set to preferred values. Click *Query*, and any matches for the selected queries will appear in a view that details each match. Also, if the query involves code changes, you can preview and confirm these changes by clicking on a *Do Refactoring* button.

Jackpot rules

Jackpot's full power comes from its openness. This requires learning a new language but when you realize Jackpot's full potential you will see that the learning curve quickly pays off.

For example, here is a Jackpot query that detects an inefficient code pattern – the use of **equals("")** to check if a **String** is empty – and rewrites the matching code:

```
$s.equals("") => ($s.length() == 0) ::
    $s instanceof java.lang.String;
```

The syntax is **pattern => replacement :: condition**, where the **$** character identifies meta-variables that will bind to any Java program element (identifier, statement, operator, literal, etc.). Let's analyze each clause:

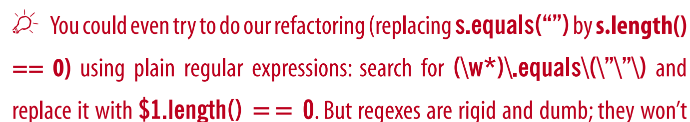
1. The pattern **$s.equals("")** matches invocations to the **equals()** method that pass an empty string as argument.

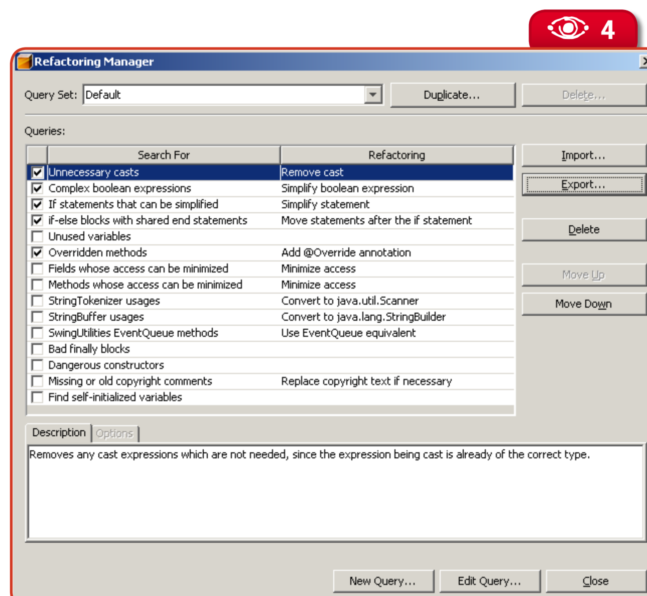
2. The condition is the only optional part of a rule in Jackpot's rule language, but it's a critical part in this particular rule: **$s instanceof java.lang.String** makes sure that the rule only fires when **$s** is a **String**. That's an important constraint, since our rule is specific to uses of **java.lang.String.equals()**, and not to just any implementation of **equals()**.

3. Finally, the replacement – **($s.length() == 0)** – rewrites the matching code.

There's a lot of sophistication behind this apparently simple behavior. For one thing, look at Jackpot's **instanceof** operator. It walks and quacks like Java's **instanceof**, but it's not the same thing. Java's **instanceof** is a runtime operator whose left-hand operand is an object reference. Jackpot's **instanceof**, however, is a compile-time (static) operator; its left-hand operand is any node of the program's AST.

Because Jackpot – like the new editor – relies on javac's source analysis engine, it's able to fully attribute all types in the processed code. This includes the most complex cases, like inferred generic types. Other code analysis tools often resort to heuristics that approximate types but might fail to calculate types for some expressions.


You could even try to do our refactoring (replacing **s.equals("")** by **s.length() == 0)** using plain regular expressions: search for **(\w*)\.equals\(\"\"\)** and replace it with **$1.length() == 0**. But regexes are rigid and dumb; they won't



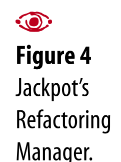
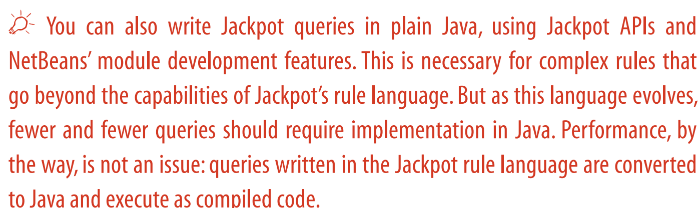


Figure 4 Jackpot's Refactoring Manager.

even exclude text that's inside comments or string literals, and a simple line break will prevent detection. This is obviously a straw man example (other tools, like PMD and FindBugs, are much smarter than regexes – although not up to javac-like precision), but it shows the value of smarter tools/features.

There are Jackpot operators without Java counterparts, from simple ones like **isTrue(*node*)**, which matches boolean expressions that can statically be proven to always evaluate to **true** – to more powerful operators like **isSideEffectFree(*node*).** The latter matches a statement, block or method that doesn't modify any variable outside its scope.

Again, such detections resemble existing code inspection tools, which detect problems like "dead code". But Jackpot's reliance on the full javac technology results in fewer false positives in detections, and higher safety in automatic replacements.



You can also write Jackpot queries in plain Java, using Jackpot APIs and NetBeans' module development features. This is necessary for complex rules that go beyond the capabilities of Jackpot's rule language. But as this language evolves, fewer and fewer queries should require implementation in Java. Performance, by the way, is not an issue: queries written in the Jackpot rule language are converted to Java and execute as compiled code.

Figure 4 shows Jackpot's *Refactoring Manager*. This configuration dialog allows you to inspect all installed queries and organize them into query sets. You can also import new queries. If you write a new query script, just click *Import* and the new query will be available in the *Query and Refactor* dialog.

Usage and perspectives

Jackpot ships with a library of predefined queries, containing many rules for code clean-up and detection of common programming mistakes or code anti-patterns, as well as migration of deprecated API usage.

As I write this, Jackpot has just been integrated into NetBeans. So we have a hybrid system with Jackpot co-existing with traditional refactoring and code manipulation features. This means that commands like *Rename method* are still implemented in the old-fashioned way, even though they could be implemented by a Jackpot rule. The same holds for code validations ("hints") and their automatic fixes. Some of this functionality will certainly be re-implemented on Jackpot in the future. Also, because Jackpot makes the development of such things much easier, you should expect an increasing number of refactorings, validations and other code-crunching features to be added to the IDE.

Extended Ant and JUnit support

Ant support in NetBeans 6.0 has been updated to Ant 1.7.0, a major new release that adds such features as support for JSR 223-compatible scripting languages. There's also a new progress indicator for Ant processes.

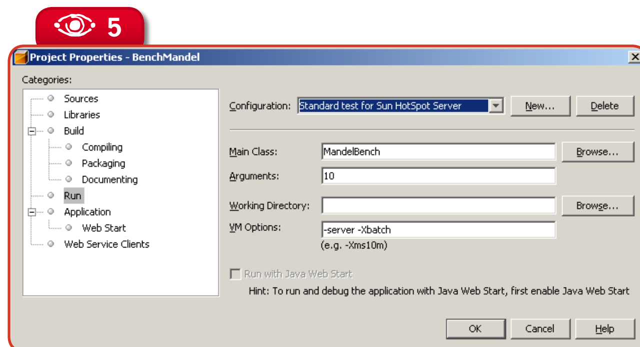
The IDE's JUnit support now handles the annotation-driven test cases of JUnit 4. Old JUnit 3.8 test cases are still supported. Also, the project properties editor is improved with classpath entries specific to unit tests.

Project and build features

Editing code is fundamental, but for most non-trivial projects a well-structured and powerful build system is critical too. NetBeans' project management and build system was improved with many new features.

In addition to its Ant support, NetBeans can open and understand Apache Maven 2 projects. Though the new Maven-based proj-

Figure 5 Multiple Configurations and support for Java Web Start in the new Project Properties dialog's Run page.



ect support is not intended to replace Ant projects anytime soon, it will be welcome to Maven fans or to anybody needing to build a project that requires Maven.

Also, now you can specify packages or classes to exclude from the source tree. This is useful for working with large projects, when you're not interested in seeing or running all of their code and a partial build is viable.

If you have many correlated projects, you can organize them into Project Groups, so certain operations like opening projects can be applied to the group as a whole. And if you write Java SE projects with many entry points (classes with **main()** methods), or with command-line parameters that require frequent edits of the project properties, the *Run Configurations* feature will make your life easier. The project properties' *Run* page shows a new *Configuration* option. Each configuration allows you to define the main class, arguments and VM options, independently of other configurations. See an example in **Figure 5**.

Furthermore, the new Java Web Start support automates the creation and maintenance of JNLP files, and makes it easier to run tests without needing a browser. In the Project Properties, check *Application>Web Start>Enable WebStart*, and off you go. Java Web Start support integrates with the Run Configurations feature, by creating a *Web Start* configuration. So you can test the same project with or without JAWS.

Version control

Robust version control is an essential feature, even for simple projects written by one developer over a weekend. For one thing, it's critical to enable "fearless programming", e.g. using techniques like refactoring (manual or automatic) without worry. NetBeans 6.0 brings plenty of news in this area too.

CVS

NetBeans has traditionally supported the CVS version control system and this support was already excellent in NetBeans 5.5. Version 6.0 adds several updates in usability, like exporting a diff patch of files selected in the Search view; a new command to open a specific revision, tag or branch; and an improved history search feature with new Summary and Diff views. There are also new advanced operations like changing the CVS root and doing a partial merge.

Subversion

The biggest news for many users, though, is support for the increasingly popular Subversion version control system. NetBeans 6.0 is the first release to integrate complete first-class support for SVN. Even though NetBeans 5.5 now offers a Subversion module in the Update Center, you really want version 6.0 if you are a heavy Subversion user.

Local History

No matter which Version Control System you prefer, you'll love the new Local History feature, already depicted in **Figure 2**. NetBeans 6.0 automatically keeps an internal history of recent changes to project resources. Every time you save a file, this is registered as a "commit" of a new version of the file in the local history. So file changes are tracked with fine granularity – somewhat like a persistent undo feature. You can inspect the "versions" in the local history and diff them against the current files.

Be warned, however, that this feature is mostly useful for undoing mistakes that escape the editor's undo capacity, e.g. after closing the editor or restarting the IDE. You can then revert to a previous state that you haven't yet committed to a safer VCS repository, perhaps because the new code was still rough and untested. The Local History feature is powerful and is sometimes a lifesaver, but it's not a full replacement for a real VCS.

Debugging

The debugger is of course among the most critical features of an IDE, and NetBeans is already very complete in this area. So what's left to improve in 6.0? First off, the Java SE 6 release contains two

The new NetBeans Installer. As of this writing, you must follow a link to a directory where you'll navigate to the installer page for a specific build

important new JVM debugging features which require an updated debugger to use. (The debuggers from NetBeans 5.5 or older releases won't benefit from these even if you run them on top of Java SE 6.) There are also other debugger improvements that are not dependent on the JRE version, so you'll benefit even if you are chained to some stone-age Java runtime like 5.0 or, heavens forbid, 1.4.2.

Forcing return values

Suppose you're stepping anywhere in a method and you'd like to force it to return immediately and produce a specific return value. This is now supported in the 6.0 debugger, letting you check "what-if" scenarios and reproduce bugs more easily. You won't need hacks like patching the source code with **return** statements (and having to unpatch it later). As I write, this feature is not yet implemented, but it should be before the final release.

Expression stepping

Expression stepping is another smart timesaver. In complex expressions containing method calls, you can step into individual invocations, and when such a call returns you can see the returned value even it's not assigned to any local variable. You no longer have to break expressions into simple parts and introduce temporary locals for the single purpose of helping debugging. Also, the Local Variables view will show the value returned by invoked methods.

Expression stepping will work in any Java runtime, but showing values returned by invoked methods requires Java SE 6.

Multithreading support

Another new feature that's very useful is *Debug current thread*: you can instruct the debugger so that only a given thread will stop in breakpoints. This is crucial for debugging concurrent applications that have several threads running the code of interest. Since *we* developers are not multithreaded, we're easily overwhelmed when setting a breakpoint causes the debugger to stop twenty threads at once!

Other features

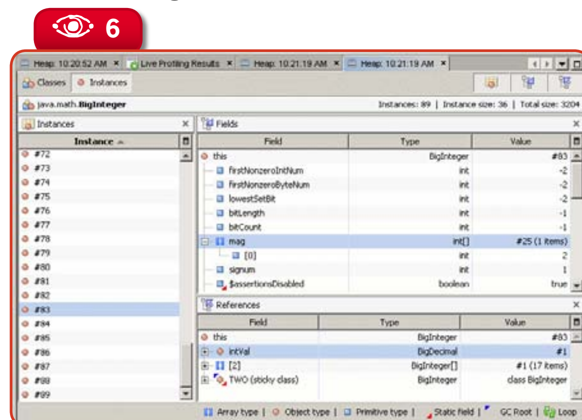
There are also general improvements to other features, like better handling of broken breakpoints (e.g. with incorrect conditions), and a command to copy call stacks to the clipboard.

New Profiler features

In NetBeans 6.0, the Profiler becomes part of the core distribution, and there's a range of important improvements.

- *Better performance* – Performance is good anywhere but it's always a critical issue in profilers. The NetBeans Profiler, which derives from Sun's JFluid research project, pioneered a new technology that allows profiling apps nearly at full speed by dynamically instrumenting code. Also, the Profiler itself should be fast to analyze and present data collected from the JVM – especially online data that's constantly updated as the program runs. The new release improves significantly the performance of the Live Results categorization and drill down, so you'll find yourself using this feature more often.
- *Classloading telemetry* – The VM Telemetry view now shows the number of loaded classes together with the number of threads.
- *Memory snapshot comparison* – Your application has a method that's suspect of leaking? Take heap snapshots before and after running it then diff the two snapshots.
- *Heap Walker* – The ultimate tool for leak hunting and any kind of memory allocation analysis. You can load a heap dump and visualize the full object graph in the heap (see **Figure 6**).

Figure 6 The Profiler's Heap Walker, inspecting a particular instance of BigInteger.



- *Load generation* – The Profiler supports integration with load generation tools (currently only Apache JMeter is supported but more is to come).
- *Profiling Points* – These are a profiler's equivalent of debugger breakpoints. You can define places in your source code where the profiler should start/stop the clock, reset profiling results or take a snapshot. The Profiling Points feature removes most bureaucratic profiling work: never again will you need to step or pause code to get snapshots in critical events; you also won't need to tweak code to measure the latency of a region that doesn't coincide with a full method.

GUI and usability

An IDE should have a beautiful, efficient and productive GUI as much as any other application. NetBeans 6.0 makes new strides in this direction.

Linux and Solaris users will certainly welcome the much improved GTK L&F, which is now activated by default on these platforms. The activated-by-default part depends on Sun's JRE 6 Update 1 (or better), which contains its own share of important GTK updates. NetBeans will respect all settings from the active GTK theme.

The new NetBeans Installer (NBI) makes installation easier and faster. In the downloads page, you can select which packs you want (e.g. Enterprise, Mobility). Then you'll be offered a custom installer that includes all chosen features and will install these in a single go. NBI is especially convenient for system administrators that need to install the same IDE configuration in multiple machines, and for trainers who often land in unprepared laboratories.

NetBeans also includes redesigned icons, and the SDI windowing option (a relic from ancient NetBeans releases) was removed. Now you have undockable/floating windows. Finally, in the QA front, the new Report Exception tool streamlines reporting of detailed error data to NetBeans' developers, while the UI Gestures Collector can submit data about your IDE usage patterns. This data is useful not only for research, but also to implement a kind of "tip of the day" hint system not based on **Math.random().** I tested this, and the NetBeans Analytics site offered me a tutorial about profiling multithreaded programs, which was highly correlated with the tasks I had been performing in recent days.

Matisse and visual web development

There are only two core IDE features I'm not covering here. Both are award-winning tools and top reasons for many developers having moved to NetBeans: the Matisse visual editor, and the Visual Web Pack. NetBeans 6.0 brings significant updates to both. For Matisse, check out the article "UI Design in NetBeans 6.0" in this issue, where you'll find detailed information about what's new.

Currently, the most important changes in the Visual Web Pack refer to its integration into the NetBeans core. Actually, there won't be an external Web Pack for 6.0. The IDE already offered support for web application development, so it was a little odd to have some of that in the core and the rest in an external Pack. Historically, this happened because the Web Pack technology was originally developed as a separate product (Sun's Java Studio Creator), which was based on a fork of a very old NetBeans version. So its implementation became partially redundant with NetBeans' web tooling. Now this chasm is closed and there will be no more duplicate code or effort. The merge results in a simpler IDE for all users: from visual-design lovers to tag-writing diehards.

There are several new features in the integrated web tooling but as we write they are still under heavy development, so it wasn't viable to cover the new functionality in this issue. However, don't miss the article "Visual Web Application Design with NetBeans" for an updated tutorial on the last stable version.

Plugin Manager

NetBeans' open, extensible architecture is one of its core advantages and it's also very easy to use and integrate with. You may be surprised that the *Tools>Update Manager* has disappeared, though.

[2] Incidentally, several menu options were simplified in NetBeans 6.0; for instance, *Java Platform Manager* became *Java Platforms*.

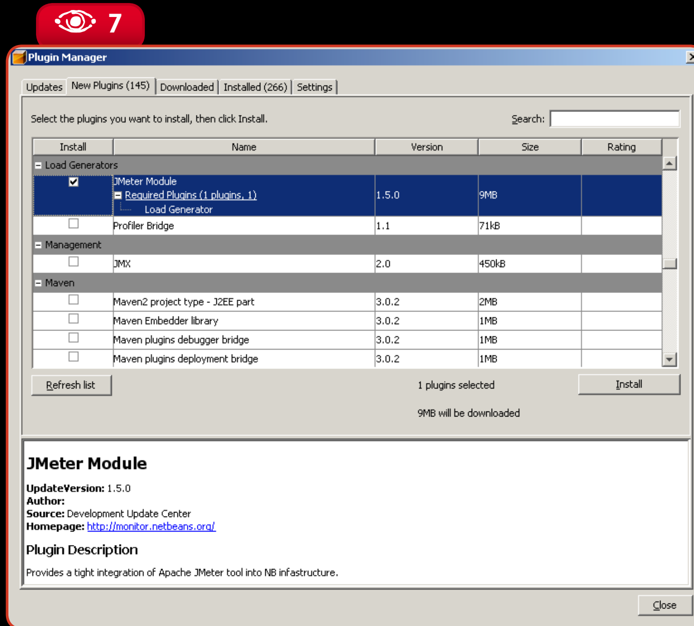


Figure 7 The new Plugin Manager.

But just look again, at *Tools>Plugins*[2], and you'll see **Figure 7**.

The new UI unifies and better organizes the old Update Center (see the *Updates*, *New Plugins*, *Downloaded* and *Settings* tabs), and also the old module manager (see the *Installed* tab). There are new features too: for example, when you select a plugin (like we did for the JMeter Module in **Figure 7**), a *Required Plugins* node will appear if applicable; you can expand it to see any dependencies that must also be installed.

Conclusions

NetBeans 6.0 comes with a massive number of new and improved features and certainly deserves the major version bump. If NetBeans 5.5 was wide, NetBeans 6.0 is also deep. Developers upgrading to the latest version will have not only extensive support for all kinds of Java development but also a best-of-breed feature set in every important functionality area.

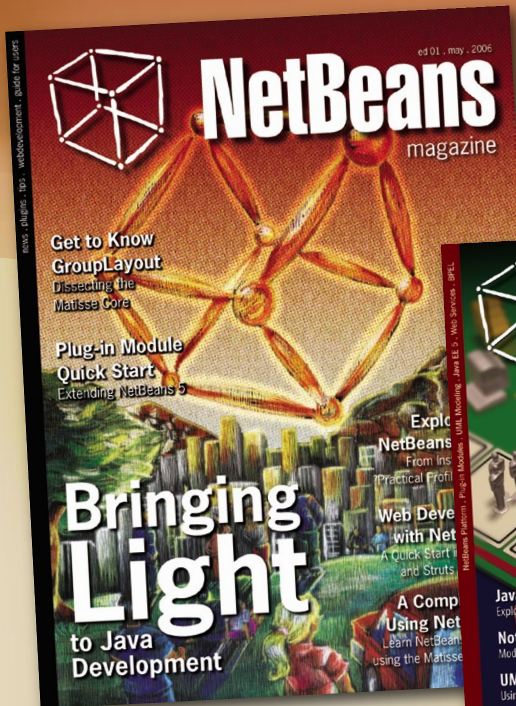
Many NetBeans power users may have gone through this article and found features that were already available for previous versions via additional modules. From several editor enhancements to Run Configurations, to the Local History, you could find an *nbm* file that would provide some level of support for your need. However, you can now just install the core IDE and have all these features out of the box – and they're superior, more polished and better integrated than what's provided through external modules. This happens of course with every new release, but NetBeans 6.0 makes a very noticeable effort to catch up with its RFEs, embracing a large number of improvements that first surfaced as contributions from the broader community. This can only be viewed as great news, and as evidence of a project that moves fast in the direction users want.

Osvaldo Pinali Doederlein (*opinali@gmail.com*) is a software engineer and consultant, working with Java since 1.0 beta. He's an independent expert for the JCP, having served for JSR-175 (Java SE 5), and is a Technology Architect at Visionnaire Informatica. Osvaldo has an MSc in Object Oriented Software Engineering, is a contributing editor for Java Magazine and maintains a blog at *weblogs.java.net/blog/opinali*.

1 year

NetBeans
magazine

With this third issue, NetBeans Magazine is completing its first anniversary. Kudos and thanks to the NetBeans developer community for enabling us to spread the word even more about this wonderful IDE and Platform!



netbeans.org/community/magazine

pH DESIGN