

## **NetBeans**

Developing Applications with NetBeans IDE

Release 8.2

**E74654-01**

September 2016

Documentation for NetBeans users that describes how to use the NetBeans IDE and provides detailed information on the functionality available within it.

NetBeans Developing Applications with NetBeans IDE, Release 8.2

E74654-01

Copyright © 2013, 2016 Oracle and/or its affiliates. All rights reserved.

Primary Author: Alyona Stashkova, Catherine Pickersgill

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

**U.S. GOVERNMENT END USERS:** Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

---

---

# Contents

<b>Preface .....</b>	<b>xxvii</b>
Audience.....	xxvii
Documentation Accessibility .....	xxvii
Related Documents .....	xxvii
Conventions .....	xxvii
<b>What's New in This Guide .....</b>	<b>xxix</b>
New and Changed Features for Release 8.2 .....	xxix
<b>1 Introduction to NetBeans IDE</b>	
1.1 About NetBeans IDE .....	1-1
1.2 NetBeans IDE Developer Resources .....	1-1
<b>2 Working with NetBeans IDE</b>	
2.1 About Working with NetBeans IDE.....	2-1
2.2 Working with the Options Window .....	2-2
2.2.1 How to Edit IDE Settings.....	2-2
2.2.2 How to Export IDE Settings as Zip Archive .....	2-2
2.2.3 How to Import IDE Settings.....	2-2
2.3 Managing IDE Windows .....	2-3
2.3.1 How to Move a Window .....	2-4
2.3.2 How to Configure Window Behavior and Appearance .....	2-5
2.3.3 How to Simultaneously Display Multiple Files in the Editor .....	2-5
2.3.4 How to Clone the View of a Single File.....	2-5
2.3.5 How to Manage Open Files.....	2-5
2.4 Customizing Toolbars in the IDE .....	2-6
2.4.1 How to Show or Hide a Toolbar.....	2-6
2.4.2 How to Move a Toolbar .....	2-6
2.4.3 How to Add a Toolbar Button to a Toolbar .....	2-6
2.4.4 How to Remove a Button from a Toolbar .....	2-6
2.4.5 How to Reorder a Button in a Toolbar .....	2-7
2.4.6 How to Change the Size of Toolbar Buttons.....	2-7
2.4.7 How to Add a New Toolbar to the Main Window .....	2-7
2.5 Working with Keyboard Shortcuts .....	2-7
2.5.1 How to Add or Remove a Keyboard Shortcut for a Menu Command .....	2-7

2.5.2	How to Customize Keyboard Shortcuts.....	2-8
2.5.3	How to Switch Between Keyboard Shortcut Sets .....	2-8
2.5.4	How to Create a New Keyboard Shortcut Set.....	2-9
2.5.5	How to Use the Default Menu Shortcuts .....	2-9
2.5.6	How to Customize Keyboard Shortcuts.....	2-15
2.6	Understanding the Source Editor Features .....	2-15
2.6.1	How to Use the Toolbar.....	2-16
2.6.2	How to Use the Left Margin.....	2-17
2.6.3	How to Use the Error Stripe.....	2-18
2.6.4	How to Use the Status Line .....	2-18
2.6.5	How to Use Syntax Coloring and Highlighting.....	2-18
2.6.6	How to Use Insert Mode and Overwrite Mode .....	2-19
2.6.7	How to Set Editor Options .....	2-19
2.6.8	How to Insert and Highlight Occurrences in the Editor.....	2-20
2.6.9	How to Record Macros .....	2-21
2.6.9.1	Macro Keywords for NetBeans Java Editor.....	2-21
2.6.10	How to Modify Source Editor Code Templates.....	2-22
2.6.11	How to Use Code Completion.....	2-23
2.6.12	How to Use Hints .....	2-25
2.6.13	How to Navigate Through Code .....	2-26
2.6.14	How to Work with Import Statements .....	2-27
2.6.15	How to Generate Code.....	2-28
2.6.16	Using General Editor Shortcuts .....	2-28
2.7	Setting Startup Parameters .....	2-31
2.8	Setting Fonts and Colors for IDE Windows .....	2-32
2.8.1	How to Set Fonts and Colors for the Help Viewer .....	2-32
2.8.2	How to Set Fonts and Colors for the Output Window .....	2-33
2.8.3	How to Set Fonts and Colors for the Terminal Window .....	2-33
2.9	Managing Plugins in the IDE .....	2-33
2.9.1	How to Update the IDE from the Update Center .....	2-34
2.9.2	How to Install Downloaded Plugins .....	2-34
2.9.3	How to Activate and Deactivate Plugins .....	2-35
2.9.4	How to Globally Install Plugins .....	2-35
2.9.5	How to Add an Update Center.....	2-36
2.9.6	How to Schedule Update Checks .....	2-36
2.10	Displaying IDE Help in a Web Browser .....	2-36
2.10.1	How to Extract IDE Help from a JAR File.....	2-37
2.10.2	How to View IDE Help in a Web Browser.....	2-37
2.11	Internationalizing Source Code .....	2-37
2.11.1	How to Enable Automatic Internationalization .....	2-37
2.11.2	How to Internationalize a Single File.....	2-38
2.11.3	How to Use the Internationalization Wizard .....	2-38
2.11.4	How to Insert an Internationalized String Into Source Code.....	2-39
2.11.5	How to Internationalize a String With the GUI Builder .....	2-39
2.11.6	How to Test a Bundle for Internationalized Strings.....	2-40
2.11.7	How to Add Arguments for Message Formats .....	2-40
2.12	Managing and Creating Projects .....	2-41

2.12.1	How to Create a Project .....	2-41
2.12.2	How to Work with Character Encodings for a Project.....	2-42
2.12.2.1	Changing the Character Encoding of a Project .....	2-42
2.12.3	How to Organize Projects into Groups .....	2-43
2.13	Working with Source Files in the IDE.....	2-43
2.13.1	How to Find Files in Projects .....	2-43
2.13.2	How to Specify Editor Formatting Options.....	2-44
2.13.3	How to Compare Two Files .....	2-45
2.13.4	How to Apply a Diff Patch to a File.....	2-45
2.13.5	How to Access Files Outside of a Project .....	2-46
2.13.6	How to Create a File Template .....	2-46
2.13.7	How to Work with Unknown File Extensions .....	2-47
2.13.8	How to Specify Files to Ignore.....	2-47
2.13.9	How to Work with Character Encodings for Files.....	2-48
2.13.10	How to Specify Action Items .....	2-48
2.13.11	How to Use Bookmarks in Files .....	2-50
2.14	Working with Resource Bundles .....	2-50
2.14.1	How to Create and Delete Resource Bundles.....	2-51
2.14.2	How to Edit a Resource Bundle.....	2-51
2.14.3	How to Add and Remove a Property .....	2-52
2.14.4	How to Add and Remove a Locale .....	2-53
2.14.5	How to Edit a Locale .....	2-53
2.15	Working with Javadoc Documentation .....	2-54
2.15.1	How to Add Javadoc to a Project .....	2-54
2.15.2	How to Add the JDK Javadoc to the IDE .....	2-55
2.15.3	How to View Javadoc Documentation .....	2-55
2.15.4	How to Generate Javadoc Documentation .....	2-55
2.15.5	How to Enter Javadoc Comments in Source Code .....	2-56
2.15.6	How To Analyze and Fix Javadoc Comments .....	2-57
2.16	Viewing IDE Notifications.....	2-57

### 3 Versioning Applications with Version Control

3.1	About Versioning Applications with Version Control.....	3-1
3.2	Versioning Applications with Git.....	3-1
3.2.1	Git Visualization Features .....	3-2
3.2.2	How to Initialize a Git Repository.....	3-2
3.2.3	How to Clone a Git Repository .....	3-3
3.2.3.1	Cloning a Repository from GitHub using SSH protocol .....	3-4
3.2.4	How to Add Files to a Repository .....	3-5
3.2.5	How to Commit Sources to a Git Repository .....	3-6
3.2.6	How to Revert Modifications.....	3-7
3.2.7	How to Reset .....	3-7
3.2.8	How to Create a Tag.....	3-8
3.2.9	How to Compare File Revisions .....	3-8
3.2.10	How to Work with Branches.....	3-9
3.2.10.1	Creating a Branch .....	3-9
3.2.10.2	Checking Out .....	3-10

3.2.10.3	Merging .....	3-11
3.2.10.4	Deleting .....	3-11
3.2.11	How to Work with Remote Repositories .....	3-11
3.2.11.1	Fetching .....	3-11
3.2.11.2	Pulling .....	3-12
3.2.11.3	Pushing .....	3-12
3.2.12	How to Set Git Global Options .....	3-12
3.2.13	How to Shelve Changes (Git).....	3-13
3.2.14	How to Rebase (Git) .....	3-13
3.3	Versioning Applications with Subversion .....	3-14
3.3.1	Subversion Visualization Features.....	3-14
3.3.2	Working with Subversion.....	3-14
3.3.3	How to View File Status Information.....	3-15
3.3.4	How to Work with Version Histories .....	3-16
3.3.4.1	Searching for Specific Revisions.....	3-16
3.3.4.2	Comparing Revisions in the Search History Window .....	3-17
3.3.5	How to Set Up Subversion .....	3-17
3.3.6	How to Check Out Files From a Remote Repository (Subversion).....	3-18
3.3.7	How to Place Projects Under Version Control (Subversion).....	3-19
3.3.8	How To Update Files in a Local Working Directory (Subversion) .....	3-19
3.3.8.1	Updating Projects with Dependencies .....	3-20
3.3.9	How to Compare File Revisions in Subversion.....	3-20
3.3.9.1	Comparing File Revisions Graphically .....	3-20
3.3.10	How to Commit Local Changes to a Remote Repository (Subversion).....	3-21
3.3.10.1	Ignoring Files.....	3-22
3.3.11	How to Work with Branches in Subversion .....	3-22
3.3.11.1	Checking Out Branches .....	3-22
3.3.11.2	Creating Branches.....	3-23
3.3.11.3	Switching to a Branch .....	3-23
3.3.12	How to Revert Modifications (Subversion) .....	3-24
3.3.13	How to Recover Deleted Files (Subversion) .....	3-24
3.3.14	How to Merge File Revisions in Subversion.....	3-25
3.3.15	Resolving Merge Conflicts in Subversion .....	3-25
3.3.16	How to Create and Apply a Patch (Subversion) .....	3-26
3.3.17	How to Shelve Changes (Subversion) .....	3-27
3.4	Versioning Applications with Mercurial.....	3-27
3.4.1	About Mercurial Visualization Features .....	3-28
3.4.2	How to View File Status Information in Mercurial .....	3-28
3.4.2.1	Viewing Revision Information .....	3-28
3.4.3	How to Set Up Mercurial.....	3-29
3.4.4	How to Clone an External Mercurial Repository.....	3-30
3.4.5	How to Place Projects Under Version Control .....	3-31
3.4.6	How to Use the Mercurial Diff Viewer.....	3-32
3.4.7	How to Use the Mercurial Status Window .....	3-33
3.4.8	How to Merge File Revisions .....	3-34
3.4.9	How to Switch Branches in the Repository .....	3-34
3.4.10	How to Commit Changes to the Repository .....	3-35

3.4.10.1	Updating Tasks .....	3-36
3.4.10.2	Ignoring Files.....	3-36
3.4.10.3	Pushing Local Changes to the Shared Repository .....	3-36
3.4.11	How to Set Mercurial Project Properties .....	3-37
3.4.12	How to Set Mercurial Global Options .....	3-37
3.4.13	How to Enable Support for Mercurial Queues.....	3-38
3.4.14	How to Create a Patch.....	3-38
3.4.15	How to Refresh a Patch With Local Modifications.....	3-39
3.4.16	How to Compare Patch Revisions.....	3-40
3.4.17	How to Switch Between Patches .....	3-40
3.4.18	How to Finish Applied Patches .....	3-41
3.4.19	How to Shelve Changes (Mercurial).....	3-41
3.4.20	How to Rebase (Mercurial) .....	3-42
3.5	Versioning Applications with CVS .....	3-42
3.5.1	How to Work with CVS.....	3-42
3.5.2	How to Use CVS with the IDE.....	3-42
3.5.3	How to View File Status Information (CVS).....	3-43
3.5.3.1	Viewing Revision Information .....	3-44
3.5.4	How to Work with Version Histories .....	3-45
3.5.4.1	Comparing Revisions in the Search History Window .....	3-46
3.5.5	How to Set Up CVS .....	3-46
3.5.6	How to Adjust CVS Settings in the IDE .....	3-46
3.5.7	How to Check Out Files from a Remote Repository (CVS).....	3-47
3.5.8	How to Update Files in a Local Working Directory (CVS).....	3-48
3.5.8.1	Updating Projects with Dependencies .....	3-48
3.5.9	How to Compare File Revisions in CVS.....	3-48
3.5.10	How to Commit Local Changes to a Remote Repository (CVS).....	3-49
3.5.10.1	Excluding Files from a Commit.....	3-50
3.5.10.2	Ignoring Files.....	3-50
3.5.11	How to Revert Modifications (CVS) .....	3-50
3.5.12	How to Recover Deleted Files (CVS) .....	3-51
3.5.13	How to Work with Tags .....	3-51
3.5.14	How to Work with Branches in CVS .....	3-51
3.5.14.1	Checking out Branches .....	3-52
3.5.14.2	Switching to a Branch .....	3-52
3.5.14.3	Creating a Branch .....	3-52
3.5.15	How to Merge File Revisions from a Branch (CVS) .....	3-53
3.5.16	How to Resolve Merge Conflicts .....	3-53
3.5.17	How to Create and Apply a Patch .....	3-54
3.6	About Local History .....	3-55
3.6.1	About the IDE's Local History Tools .....	3-55
3.6.2	How to Browse Local File History .....	3-55
3.6.3	How to Recover Deleted Files.....	3-56
3.6.4	How to Revert a File to a Previous Version .....	3-56
3.6.5	How to Adjust Local History Settings.....	3-57

## **4 Working in a Collaborative Environment**

4.1	About Working in a Collaborative Environment.....	4-1
4.2	Working with Tasks.....	4-2
4.2.1	About Task Repositories.....	4-2
4.2.2	How to Work with Tasks.....	4-2
4.2.2.1	Finding and Opening Tasks.....	4-3
4.2.2.2	Creating and Saving Task Queries.....	4-3
4.2.2.3	Reporting New Tasks.....	4-4
4.2.2.4	Updating and Resolving Tasks.....	4-4
4.2.3	How to Add a Task Repository .....	4-5
4.2.4	How to Add Support for JIRA .....	4-5
4.3	Working with the Tasks Window.....	4-6
4.3.1	How to View Tasks.....	4-6
4.3.2	How to Organize Tasks.....	4-7
4.3.3	How to Configure Tasks Window Settings .....	4-8

## **5 Working with NetBeans Modules**

5.1	About NetBeans Modules.....	5-1
5.2	About the NetBeans Platform .....	5-2
5.3	Working with NetBeans Modules .....	5-2
5.4	Working with Rich-Client Applications .....	5-3
5.5	Module and Rich-Client Application Tasks: Quick Reference.....	5-6
5.6	Setting Up Modules .....	5-7
5.6.1	About Module Project Templates.....	5-8
5.6.2	How to Create a Module Project .....	5-9
5.6.3	How to Create a Library Wrapper Module Project .....	5-10
5.6.4	How to Create a Module Suite Project .....	5-11
5.6.5	How to Create a NetBeans Platform Application Project.....	5-11
5.7	Using the NetBeans APIs .....	5-12
5.7.1	Generating Skeleton API Implementations .....	5-12
5.7.1.1	About Actions .....	5-14
5.7.1.2	How to Create an Action .....	5-14
5.7.1.3	About Code Generators.....	5-16
5.7.1.4	How to Create a Code Generator .....	5-16
5.7.1.5	About File Types.....	5-16
5.7.1.6	How to Create a File Type.....	5-17
5.7.1.7	About Module Installers.....	5-18
5.7.1.8	Installing Modules.....	5-18
5.7.1.9	How to Create a Module Installer/Activator.....	5-18
5.7.1.10	About Options Panels .....	5-19
5.7.1.11	How to Create an Options Panel.....	5-19
5.7.1.12	About Quick Search Providers .....	5-19
5.7.1.13	How to Create Quick Search Providers.....	5-19
5.7.1.14	About Windows.....	5-20
5.7.1.15	How to Create a Window .....	5-20
5.7.1.16	About Wizards .....	5-21
5.7.1.17	How to Create A Wizard .....	5-22

5.7.2	Extending Skeleton API Implementations .....	5-24
5.7.3	How to Register the NetBeans Sources and Javadoc.....	5-25
5.7.4	How to Use the NetBeans Sources and Javadoc .....	5-25
5.7.5	How to Search for NetBeans APIs.....	5-25
5.7.6	Code Templates for NetBeans APIs.....	5-26
5.8	Bundling Supporting Items .....	5-29
5.8.1	How to Bundle a Library .....	5-30
5.8.2	How to Bundle a Project Template or Sample.....	5-31
5.8.3	How to Bundle an Update Center's URL .....	5-31
5.8.4	How to Bundle a JavaHelp Help Set.....	5-32
5.8.5	How to Bundle a License .....	5-33
5.9	Registering Modules.....	5-33
5.9.1	About the System Filesystem.....	5-33
5.9.2	About XML Layer Files.....	5-33
5.9.3	How to Edit an XML Layer File.....	5-34
5.9.4	How to View the System Filesystem .....	5-34
5.10	Communicating Between Modules .....	5-35
5.10.1	About Service Providers .....	5-35
5.11	Building Modules.....	5-36
5.11.1	About NBM Files .....	5-37
5.11.2	How to Build an NBM File.....	5-37
5.12	Trying Out a Module.....	5-37
5.12.1	Deploying Modules .....	5-38
5.13	About Distributing Modules .....	5-38
5.13.1	About Update Centers .....	5-38
5.13.2	How to Generate an Autoupdate Descriptor .....	5-39
5.13.3	How to Bundle an Update Center's URL .....	5-39
5.13.4	Manually Registering an Update Center URL .....	5-40
5.14	Branding a Rich-Client Application .....	5-40
5.14.1	How to Brand the Window System .....	5-41
5.14.2	How to Add a Splash Screen.....	5-41
5.14.3	How to Remove Unwanted Modules .....	5-41
5.15	Distributing Rich-Client Applications .....	5-42
5.15.1	How to Build a ZIP Distribution .....	5-42
5.15.2	How to Build a JNLP Application.....	5-42
5.15.3	How to Run a Rich-Client Application.....	5-43
5.15.4	How to Use the NetBeans Shared JNLP Repository.....	5-43

## 6 Creating Java Projects

6.1	About Creating Java Projects .....	6-1
6.2	Using Java Project Templates.....	6-2
6.2.1	Standard Project Templates.....	6-2
6.2.1.1	Source Folders.....	6-4
6.2.1.2	Project Settings.....	6-4
6.2.1.3	Switching a Java SE Project to a JavaFX Deployment Model.....	6-4
6.2.1.4	Adding a JavaFX Class to a Java SE Project.....	6-4
6.2.1.5	Project Folders .....	6-5

6.2.2	Free-Form Templates .....	6-5
6.2.2.1	Source Folders.....	6-6
6.2.2.2	Project Settings.....	6-6
6.2.2.3	IDE Commands and Ant Targets.....	6-6
6.2.2.3.1	Creating a Target to Compile a Single File .....	6-6
6.2.2.3.2	Writing a Target for the Apply Code Changes Command .....	6-7
6.2.3	Creating Standard Projects.....	6-8
6.2.3.1	Managing the Classpath.....	6-8
6.2.3.1.1	Classpath and Standard Projects.....	6-8
6.2.3.1.2	Classpath and Free-form Projects .....	6-9
6.2.3.1.3	Adding Annotation Processors to the Classpath.....	6-10
6.2.3.2	Creating Dependencies Between Projects.....	6-10
6.2.3.3	Editing IDE-Generated Ant Scripts.....	6-11
6.2.3.4	Customizing the IDE-Generated Ant Script.....	6-12
6.2.3.5	Working with a JavaFX Project.....	6-13
6.2.3.5.1	JavaFX Application and JavaFX FXML Applications .....	6-13
6.2.3.5.2	JavaFX Preloaders .....	6-13
6.2.3.5.3	Editing JavaFX Applications.....	6-13
6.2.3.5.4	Building and Running JavaFX Applications .....	6-13
6.2.3.5.5	Debugging JavaFX Applications .....	6-14
6.2.4	Creating Free-Form Projects.....	6-14
6.2.4.1	Source Folders.....	6-14
6.2.4.2	Project Settings .....	6-14
6.2.4.3	IDE Commands and Ant Targets .....	6-14
6.2.4.4	Adding a Source Directory to a Free-Form Project .....	6-15
6.2.4.5	Declaring the Classpath in a Free-Form Project.....	6-16
6.2.4.6	Mapping an Ant Target to an IDE Command.....	6-17
6.2.4.7	Debugging Free-Form Projects.....	6-20
6.2.4.8	Storing IDE Targets in a Separate Ant Script .....	6-20
6.2.4.9	Editing the project.xml File .....	6-21
6.2.4.9.1	Using Properties in the project.xml File .....	6-21
6.2.4.9.2	Validating the project.xml File .....	6-22
6.2.5	Setting Up a Java Project Based on Existing Sources.....	6-22
6.3	Importing an Eclipse or JBuilder Project .....	6-23
6.4	Setting the Main Project .....	6-27
6.5	Adding Multiple Sources Roots to a Project .....	6-27
6.6	Sharing a Library with Other Users .....	6-28
6.6.1	How to Share a Library .....	6-28
6.7	Adding a Javadoc to a Project .....	6-30
6.7.1	How to Add a Javadoc to a Project .....	6-30
6.8	Setting the Target JDK.....	6-31
6.8.1	How to Register a New Java Platform.....	6-31
6.8.2	How to Set the Target JDK .....	6-32
6.8.3	How to Upload Java SE Embedded JRE to a Remote Device.....	6-33
6.9	Moving, Copying, and Renaming a Project .....	6-33
6.9.1	How to Move, Copy, or Rename a Project.....	6-33
6.10	Deleting a Project .....	6-33

## **7 Working with Java Code**

7.1	About Working with Java Code .....	7-1
7.2	Editing Java Code .....	7-1
7.2.1	How to Identify Java Source Files .....	7-2
7.3	Navigating in Java Code .....	7-4
7.3.1	Browsing Java Files.....	7-4
7.3.2	Browsing XML Files .....	7-6
7.3.3	Browsing Ant Scripts.....	7-6
7.3.4	Browsing an XML File.....	7-6
7.3.5	How to Navigate Within Your Code .....	7-7
7.4	Finding and Replacing Text .....	7-8
7.4.1	How to Find and Replace Text .....	7-8
7.5	Using Regular Expressions.....	7-9
7.5.1	Regular Expression Constructs.....	7-9
7.5.1.1	Sample Regular Expressions.....	7-10
7.6	Using Special Code Templates.....	7-10
7.7	Using Java Editor Shortcuts.....	7-11

## **8 Building Java Projects**

8.1	About Building Java Projects .....	8-1
8.2	Working with Ant.....	8-2
8.2.1	Using Ant with the IDE .....	8-2
8.2.2	How to Edit an Ant Script .....	8-3
8.2.2.1	Writing Custom Ant Tasks.....	8-6
8.2.3	How to Run an Ant Script .....	8-6
8.2.4	How to Debug an Ant Script.....	8-7
8.2.5	How to Create a Shortcut to a Target .....	8-7
8.2.6	How to Configure Ant Settings .....	8-8
8.2.7	How to Switch Versions of Ant .....	8-8
8.2.8	Ant Classpaths and Custom Tasks.....	8-9
8.2.8.1	Adding Binaries to Ant's Classpath in the IDE .....	8-9
8.2.8.2	Build Scripts With an Explicit Classpath .....	8-9
8.2.9	How to Install Ant Documentation in the IDE.....	8-10
8.3	Working with Builds .....	8-10
8.4	Building a Java Project .....	8-10
8.4.1	How to Build a Java Project.....	8-11
8.5	Using a Build Server .....	8-12
8.5.1	Creating a Build .....	8-12
8.6	Compiling a Single Java File.....	8-13
8.7	Building a JAR File.....	8-13
8.8	Packaging an Application as a Native Installer.....	8-14
8.9	Preparing a JAR File for Deployment Outside the IDE.....	8-16
8.9.1	Running an Application JAR Outside of the IDE .....	8-16
8.10	Using the Output Window .....	8-17
8.11	Refactoring Java Projects.....	8-17
8.11.1	How to Undo Refactoring Changes .....	8-18

8.11.2	How to Find Class, Methods, and Field Usages .....	8-18
8.11.2.1	Classes and Interfaces .....	8-18
8.11.2.2	Methods .....	8-19
8.11.2.3	Fields .....	8-19
8.11.2.4	Additional Find Mechanisms .....	8-19
8.11.3	How to Rename a Class or Interface.....	8-19
8.11.4	How to Rename a Field or Method.....	8-20
8.11.5	How to Move a Class to Another Java Package.....	8-21
8.11.6	How to Move an Inner Class One Level Up .....	8-22
8.11.7	How to Move a Class Member to a Superclass .....	8-22
8.11.8	How to Move a Class Member to a Subclass .....	8-23
8.11.9	How to Copy a Class.....	8-24
8.11.10	How to Encapsulate a Field.....	8-24
8.11.11	How to Change a Method's Signature.....	8-25
8.11.12	How to Invert a Boolean Method or Variable .....	8-27
8.11.13	Replacing a Constructor .....	8-27
8.11.14	How to Introduce a Variable, Constant, Field, or Method .....	8-28
8.11.15	How to Inline a Variable, Method, or Constant .....	8-29
8.11.16	How to Extract a Superclass.....	8-29
8.11.17	How to Extract an Interface.....	8-30
8.11.18	How to Use a Supertype Where Possible.....	8-31
8.11.19	How to Convert an Anonymous Inner Class to a Regular Inner Class .....	8-31
8.11.20	How to Safely Delete Java Code.....	8-32
8.11.20.1	Handling Deletions When The Code Element is Referenced .....	8-32
8.11.21	Using Hints in Source Code Analysis and Refactoring .....	8-33
8.11.22	How to Refactor an Enterprise Bean.....	8-34
8.12	Working with Maven in the IDE .....	8-34
8.12.1	How to Create a New Maven Project .....	8-35
8.12.2	How to Configure Maven Settings.....	8-36
8.12.2.1	Configuring Maven Settings for the IDE .....	8-36
8.12.2.2	Configuring Maven Settings for Projects .....	8-36
8.12.2.3	Using Project Configurations.....	8-36
8.12.2.4	Binding IDE Actions to Maven Goals.....	8-37
8.12.2.5	Creating Custom Actions .....	8-37
8.12.3	How to Work with the Maven POM.....	8-38
8.12.4	How to Manage Maven Project Dependencies .....	8-39
8.12.5	How to Work with Maven Artifacts .....	8-39
8.12.5.1	Visualizing Dependencies .....	8-40
8.12.6	How to Build a Maven Project.....	8-41
8.12.6.1	Executing Maven Goals in the IDE .....	8-42
8.12.6.2	Customizing the Build Process.....	8-42
8.12.6.3	Executing Individual Goals.....	8-42
8.13	Working with Maven Repositories .....	8-42
8.13.1	How to Work with Maven Repositories.....	8-43

## 9 Testing and Profiling Java Application Projects

9.1	About Testing and Profiling Java Application Projects .....	9-2
-----	---	-----

9.2	Testing Java Application Projects with Unit Tests.....	9-2
9.2.1	Test Types in the IDE .....	9-2
9.2.2	Unit Test Structure.....	9-3
9.3	Creating a Unit Test.....	9-3
9.3.1	Changing the JUnit Version .....	9-3
9.3.2	How to Create a Unit Test .....	9-3
9.4	Running a Unit Test.....	9-5
9.4.1	How to Run a Unit Test .....	9-5
9.4.2	Working with Unit Test Output .....	9-6
9.5	Debugging a Unit Test .....	9-6
9.6	Configuring Unit Test Settings .....	9-6
9.6.1	How to Edit Unit Test Settings .....	9-6
9.6.2	How to Edit the Classpath for Compiling or Running Tests .....	9-7
9.7	Creating a Selenium Test .....	9-7
9.8	Configuring Selenium Server Settings.....	9-7
9.9	Starting a Profiling Session.....	9-8
9.9.1	How to Profile a Project .....	9-8
9.9.2	How to Calibrate the Profiler.....	9-9
9.9.3	Understanding the Toolbar Icons.....	9-10
9.10	Selecting a Profiling Task.....	9-11
9.10.1	How to Select a Profiling Task.....	9-11
9.10.2	Using a Load Generator Script .....	9-11
9.11	Attaching the Profiler .....	9-12
9.11.1	How to Configure the Attach Settings.....	9-12
9.11.2	How to Attach the Profiler to a Local Application .....	9-13
9.12	Attaching the Profiler to a Remote Application .....	9-14
9.12.1	How to Attach to a Remote Application .....	9-14
9.12.2	Attaching to a Remote Server .....	9-15
9.13	Profiling a Free-form Project .....	9-16
9.13.1	Profiling Free-form Web Projects .....	9-17
9.13.1.1	A Typical Free-Form Project Profile Target .....	9-18
9.13.1.2	Writing a Target to Profile a Selected File.....	9-19
9.14	Taking and Accessing Snapshots of Profiling Data .....	9-20
9.14.1	Taking Snapshots During a Profiling Session .....	9-20
9.14.2	Taking Snapshots at the End of a Profiling Session.....	9-22
9.14.3	Starting and Stopping the Application Finished Dialog.....	9-22
9.14.4	Accessing Snapshots.....	9-22
9.15	Taking a Heap Dump .....	9-22
9.15.1	How to Take a Heap Dump .....	9-22
9.15.2	How to Analyze a Heap Dump Using Object Query Language (OQL) .....	9-23
9.15.2.1	OQL Examples .....	9-24
9.15.2.2	OQL built-in objects and functions.....	9-24
9.15.2.3	Selecting Multiple Values.....	9-28
9.15.2.4	Other Examples.....	9-31
9.16	Setting a Profiling Point .....	9-32
9.16.1	How to Set Profiling Points .....	9-33
9.16.2	How to Reset Profiling Results .....	9-33

9.16.3	How to Set a Stopwatch Profiling Point .....	9-34
9.17	Profiling Telemetry .....	9-34
9.18	Profiling Methods .....	9-35
9.18.1	Basic Methods Profiling Mode.....	9-36
9.18.1.1	Selecting Threads.....	9-37
9.18.1.2	Searching And Filtering Results.....	9-37
9.18.2	Advanced Methods Profiling Mode .....	9-37
9.18.3	Selecting Classes And Methods For Profiling .....	9-38
9.18.4	Configuring Additional Options .....	9-38
9.19	Profiling Objects .....	9-39
9.19.1	Basic Profiling.....	9-39
9.19.1.1	Results View .....	9-39
9.19.1.2	Searching And Filtering Results.....	9-40
9.19.2	Advanced Profiling .....	9-40
9.19.2.1	Objects Profiling Modes.....	9-40
9.19.2.2	Selecting Classes For Profiling.....	9-41
9.19.2.3	Configuring Additional Options.....	9-41
9.19.2.4	Objects Views .....	9-41
9.19.2.5	Expert Mode of Objects Profiling .....	9-42
9.20	Profiling Threads.....	9-42
9.21	Profiling Locks.....	9-43
9.22	SQL Queries Profiling .....	9-44
9.23	Additional Functions when Running a Profiling Session.....	9-46

## **10 Running and Debugging Java Application Projects**

10.1	About Running Java Application Projects .....	10-1
10.1.1	Running Standard Projects.....	10-2
10.1.2	Running Free-form Projects .....	10-2
10.2	Working with Project Execution.....	10-2
10.3	Running an Application.....	10-2
10.4	Running a Single File.....	10-3
10.4.1	Writing a Target to Run/Debug/Test a Single File.....	10-3
10.4.1.1	Running the Selected File .....	10-4
10.4.1.2	Getting a Reference to the Currently Selected File in the IDE .....	10-4
10.4.1.3	Debugging the Selected File.....	10-5
10.5	Setting the Runtime Classpath.....	10-6
10.6	Setting the Main Class and Runtime Arguments.....	10-7
10.7	Setting JVM Arguments .....	10-8
10.8	Debugging Applications .....	10-8
10.8.1	Debugging Free-form Projects .....	10-9
10.8.1.1	How to Create a Debug Target for a Free-form Java Project .....	10-9
10.8.1.1.1	A Typical Free-form Project Debug Target.....	10-10
10.8.1.1.2	Manually Mapping a Target to a Menu Item.....	10-12
10.8.1.1.3	Troubleshooting.....	10-12
10.8.1.2	How to Create a Debug Target for a Free-form Web Project.....	10-12
10.8.1.2.1	Using the Debug Target .....	10-15
10.8.1.2.2	Troubleshooting the Debug Target.....	10-16

10.8.2	Debugging GUI Projects .....	10-18
10.8.2.1	GUI Snapshots.....	10-18
10.8.2.2	Working with the Visual Debugger.....	10-19
10.8.2.2.1	Locating the Source Code for Components.....	10-19
10.8.2.2.2	Exploring Component Events .....	10-20
10.8.2.3	How to Configure Java Debugger Options .....	10-20
10.9	Using the Debugger Windows.....	10-21
10.9.1	Customizing a Debugger Window .....	10-21
10.9.2	Choosing Current Context in the Debugger.....	10-22
10.9.2.1	Debugger Windows and Context.....	10-22
10.9.2.2	The Source Editor and Context.....	10-22
10.9.3	Attaching Source Code to a JAR File .....	10-22
10.9.4	Managing Breakpoints .....	10-24
10.9.4.1	How to Set a Java Breakpoint .....	10-25
10.9.4.2	How to Set a Conditional Breakpoint.....	10-26
10.9.4.3	How to Organize Breakpoints Into a Group .....	10-27
10.9.4.4	About Source Maps Support.....	10-27
10.9.5	Managing Debugging Sessions.....	10-27
10.9.5.1	How to Manage a Local Debugging Session .....	10-28
10.9.5.1.1	Debugging the Main Project .....	10-28
10.9.5.2	How to Manage a Remote Debugging Session .....	10-29
10.9.5.3	How to Step Through Your Program .....	10-29
10.9.5.4	How to Fix and Continue in a Debugging Session.....	10-30
10.9.5.5	How to Finish a Debugging Session.....	10-31
10.9.6	Viewing Program Information When Debugging .....	10-31
10.9.6.1	Using the Variables Window .....	10-31
10.9.6.2	Using the Loaded Classes Window .....	10-32
10.9.6.3	Using the Events Window.....	10-32
10.9.6.4	Evaluating Variables in the Source Editor .....	10-32
10.9.6.5	How to Create a Watch.....	10-33
10.9.6.6	How to Create a Fixed Watch.....	10-33
10.9.6.7	How to Pin a Watch .....	10-34
10.9.6.8	Debugging Threads in the IDE .....	10-34
10.9.6.8.1	Changing the Current Thread .....	10-34
10.9.6.8.2	Suspending and Resuming Threads.....	10-34
10.9.6.8.3	Editor window icons.....	10-35
10.9.6.8.4	Multi-threaded Applications .....	10-35
10.9.6.8.5	Viewing Source Code for a Thread.....	10-35
10.9.6.9	Using the Call Stack .....	10-35
10.9.6.9.1	Changing the Current Call.....	10-36
10.9.6.9.2	Popping a Call From the Call Stack .....	10-36
10.9.6.10	How to Evaluate Code.....	10-36
10.9.6.11	How to Step Through an Expression.....	10-37

## 11 Implementing Java GUIs

11.1	About Implementing Java GUIs .....	11-1
11.1.1	The IDE's Java GUI Tools .....	11-1

11.2	Working with the GUI Builder .....	11-2
11.2.1	How to Create a New Form .....	11-3
11.2.2	How to Work with Containers .....	11-4
11.2.2.1	Controlling View Focus .....	11-4
11.2.2.2	Reordering Components Within a Container .....	11-4
11.2.3	How to Add a Component to a Form.....	11-5
11.2.4	How to Select Components in a Form .....	11-6
11.2.5	Controlling Selection Depth.....	11-6
11.2.6	How to Align Components .....	11-7
11.2.7	How to Size Components .....	11-8
11.2.8	How to Edit Component Properties .....	11-8
11.2.9	How to Set Events with the Connection Wizard .....	11-9
11.2.10	How to Manage Component Events.....	11-10
11.2.10.1	Defining Event Handlers.....	11-10
11.2.11	How to Modify GUI Source Code.....	11-11
11.2.11.1	Modifying Code Generation for Form Components.....	11-12
11.2.11.2	Setting Variable Modifiers for a Java Component.....	11-12
11.2.11.3	Modifying Code Generation for a Property .....	11-12
11.2.11.4	Modifying GUI Form Code Outside of the IDE.....	11-13
11.2.12	How to Create a Multiple Document Interface (MDI) Application .....	11-14
11.2.13	How to Create Accessible Forms.....	11-15
11.2.14	How to Preview a Form.....	11-15
11.3	Working with Layout Managers.....	11-15
11.3.1	How to Set the Layout Manager.....	11-17
11.3.2	How to Use the GridBag Customizer .....	11-18
11.3.3	How to Use a Custom Layout Manager.....	11-19
11.3.4	How to Set Layout Properties .....	11-20
11.4	Adding a Bean to the Window.....	11-20
11.5	Working with Database Applications and Beans Binding .....	11-21
11.5.1	How to Bind Two Bean Properties.....	11-22
11.5.2	How to Bind Data to a Swing Components.....	11-22
11.5.3	How to Use Special Binding Properties (Java Desktop Applications).....	11-23
11.5.4	How to Convert Values Between Source and Target Properties (Java Desktop Applications) .....	11-24
11.5.5	How to Validate Target Value Changes in Bindings (Java Desktop Applications).....	
	11-25	
11.6	Deploying GUI Applications.....	11-26
11.6.1	Preparing a GUI Application for Distribution .....	11-26
11.6.2	Running a Standalone GUI Application.....	11-27
11.7	Configuring the GUI Builder.....	11-27

## **12 Developing Web Applications**

12.1	About Developing Web Applications.....	12-1
12.2	Creating Web Application Projects .....	12-2
12.2.1	How to Create a Web Application Project .....	12-3
12.2.2	Setting Up a Web Project Based on Existing Sources .....	12-3
12.3	Working with JSP Files.....	12-4

12.3.1	How to Create a JSP File .....	12-5
12.3.2	How to Set Character Encoding .....	12-5
12.3.3	How to Edit a JSP File .....	12-9
12.3.4	How to Access a Custom Tag from a JSP Page .....	12-9
12.3.5	How to Access an Applet from a JSP Page .....	12-10
12.3.6	How to Compile a JSP File .....	12-10
12.3.7	How to View a JSP File's Servlet.....	12-11
12.3.8	How to Pass Request Parameters .....	12-12
12.3.9	How to Run a JSP File .....	12-12
12.4	Working with Tag Libraries .....	12-13
12.4.1	How to use Tag Libraries .....	12-13
12.4.2	How to Create a Tag Library Descriptor.....	12-13
12.4.3	How to Edit a Tag Library Descriptor .....	12-14
12.4.4	How to Create a Tag File .....	12-15
12.4.5	How to Edit a Tag File .....	12-16
12.4.6	How to Create a Tag Handler.....	12-17
12.4.7	How to Define TLD Information for a Tag Handler .....	12-17
12.5	Working with Applets.....	12-18
12.5.1	How to Create an Applet.....	12-18
12.5.2	How to Create an Applet that is Called from a JNLP File.....	12-19
12.5.3	How to Run an Applet.....	12-19
12.5.4	How to Generate an Applet Policy File.....	12-20
12.6	Working with Servlets.....	12-21
12.6.1	How to Create a Servlet Source File.....	12-21
12.6.2	How to Edit a Servlet Source File .....	12-22
12.6.3	How to View the Servlet Generated from a JSP File .....	12-22
12.6.4	How to Specify Parameters for a JSP Page.....	12-23
12.6.5	How to Run a Servlet .....	12-23
12.7	Using Filters.....	12-24
12.7.1	How to Create a Filter .....	12-24
12.7.2	How to Register a Filter .....	12-25
12.8	Using Web Application Listeners.....	12-25
12.8.1	How to Create a Web Application Listener.....	12-26
12.8.2	How to Register a Web Application Listener .....	12-26
12.9	Using WebSocket Endpoints .....	12-27
12.9.1	How to Create a WebSocket Endpoint .....	12-27
12.9.2	How to Create a WebSocket Encoder or Decoder .....	12-28
12.10	Configuring a Web Application.....	12-28
12.10.1	How to Set Build Properties.....	12-29
12.10.2	How to Edit Deployment Descriptors .....	12-29
12.11	Deploying a Web Application.....	12-30
12.11.1	How to Deploy a Web Application.....	12-31
12.11.2	How to Change the Target Server .....	12-33
12.12	Debugging a Web Application.....	12-33
12.12.1	How to Debug a Web Application.....	12-33
12.12.2	How to Debug a JSP File.....	12-34
12.12.3	How to Debug a Servlet.....	12-34

12.13	Profiling a Web Application.....	12-34
12.13.1	How to Profile a Standalone Web Application .....	12-34
12.13.2	How to Profile an Enterprise Application.....	12-35

## **13 Using Web Application Frameworks**

13.1	About Using Web Application Frameworks .....	13-1
13.2	Working with the JavaServer Faces Framework.....	13-1
13.2.1	How to Create a New Project with JSF Support.....	13-3
13.2.2	How to Add Support for a JSF Component Suite.....	13-4
13.2.3	How to Add JSF Support to an Existing Web Application.....	13-5
13.2.4	How to Create a JSF Page .....	13-5
13.2.5	How to Edit a JSF page .....	13-5
13.2.6	How to Create a JSF Facelets Template .....	13-7
13.2.7	How to Create Composite Components .....	13-8
13.2.8	How to Create a JSF Managed Bean .....	13-8
13.2.9	How to Work with JSF Components in the Palette .....	13-9
13.2.10	How to Create a JSF Form for Entity Data.....	13-10
13.2.11	How to Generate a JSF Data Table from an Entity Class.....	13-11
13.2.12	How to Generate JSF Pages from Entity Classes.....	13-12
13.2.13	How to Generate a Validation Constraint.....	13-13
13.3	Working with the Spring Framework.....	13-13
13.3.1	How to Create a New Project with Spring MVC Framework Support .....	13-15
13.3.2	How to Add Spring MVC Framework Support to an Existing Application .....	13-15
13.3.3	How to Create a Spring Configuration File .....	13-16
13.3.4	How to Organize Spring Configuration Files.....	13-16
13.3.4.1	Accessing the Spring Configuration Group Panel.....	13-16
13.3.4.2	Specifying Spring Configuration Files .....	13-17
13.3.4.3	Organizing Spring Configuration Files into Groups.....	13-17
13.4	Working with the Struts Framework .....	13-17
13.4.1	How to Create a New Project with Struts Framework Support .....	13-18
13.4.2	How to Add Struts Framework Support to an Existing Application .....	13-18
13.5	Working with the Hibernate Framework.....	13-18
13.5.1	How to Create a New Project with Hibernate Support .....	13-19
13.5.2	How to Add Hibernate Support to an Existing Application.....	13-19
13.5.3	How to Create the Hibernate Configuration File.....	13-20
13.5.4	How to Edit the Hibernate Configuration File.....	13-20
13.5.5	How to Create Hibernate Mapping Files .....	13-21
13.5.6	How to Generate Hibernate Mapping Files and POJOs from a Database .....	13-21
13.5.7	How to Create a Hibernate Reverse Engineering File.....	13-22
13.5.8	How to Create a Hibernate Utility Helper File .....	13-23
13.5.9	How to Create and Execute a HQL Statement or Script .....	13-23
13.6	Working with the Grails Framework.....	13-24
13.6.1	How to Create a New Project with Grails Framework Support.....	13-24
13.6.2	How to Install a Grails Plugin into a Grails Application.....	13-25

## **14 Developing Enterprise Applications**

14.1	About Developing Enterprise Applications.....	14-1
------	---	------

14.2	Adding Modules to the Project.....	14-2
14.2.1	How to Add a Module to an Existing Project.....	14-3
14.3	Adding Resources to the Project.....	14-3
14.3.1	How to Add an External Resource to an EAR File .....	14-3
14.4	Editing Deployment Descriptors .....	14-3
14.4.1	How to Edit an Enterprise Application's Deployment Descriptors.....	14-4
14.5	Building Enterprise Applications .....	14-4
14.5.1	How to Build an Enterprise Application Project and Sub-Projects.....	14-4
14.6	Verifying Enterprise Applications.....	14-5
14.6.1	How to Verify a Project's Deployment Descriptors.....	14-5
14.7	Deploying Enterprise Applications.....	14-6
14.7.1	How to Run an Enterprise Application.....	14-6
14.7.2	How to Deploy an Enterprise Application .....	14-6
14.7.3	How to Undeploy an Enterprise Application .....	14-7
14.8	Packaging Enterprise Applications .....	14-7
14.8.1	How to Redeploy a Project to a Different Server .....	14-7
14.9	About Docker Platform Support.....	14-7
14.9.1	How to Register a Docker Instance .....	14-7
14.9.2	How to Manage Docker Images .....	14-8
14.9.3	How to Manage a Container .....	14-8
14.9.4	How to Run a Docker Container .....	14-9
14.9.5	How to Create a Dockerfile .....	14-9
14.9.6	How to Edit a Dockerfile .....	14-9
14.9.7	How to Build an Image from a Dockerfile .....	14-9

## 15 Developing Application Clients

15.1	About Developing Application Clients .....	15-1
15.2	Creating Application Clients.....	15-1
15.2.1	How to create an enterprise application client.....	15-2
15.2.2	How to edit the deployment descriptors of an enterprise application client .....	15-2
15.2.3	How to add a module to an enterprise application .....	15-3

## 16 Developing with Enterprise Beans

16.1	About Developing with Enterprise Beans.....	16-1
16.2	Creating an EJB Module Project.....	16-2
16.2.1	How to Create an EJB Module Project.....	16-3
16.3	Creating an Enterprise Bean.....	16-4
16.3.1	How to Create Enterprise Beans.....	16-4
16.3.2	How to Generate Session Beans for Entity Classes.....	16-5
16.3.3	How to Define a Business Method for an Enterprise Bean .....	16-6
16.3.4	How to Send JMS Messages.....	16-7
16.3.5	How to Use a Service Locator .....	16-8
16.3.6	How to Access a Connection Pool from a Java Class .....	16-9
16.3.7	How to Send an Email from a Java File.....	16-9
16.4	Calling an Enterprise Bean .....	16-10
16.4.1	How to call an Enterprise Bean .....	16-10

16.5	Building and Deploying an EJB Module .....	16-11
16.5.1	How to Build an EJB Module as Part of an Enterprise Application.....	16-11
16.5.2	How to Build a Stand-alone EJB Module .....	16-12
16.5.3	How to Edit the Deployment Descriptors of an EJB Module.....	16-13
16.5.4	How to Verify the Deployment Descriptors of an EJB Module .....	16-14
16.5.5	How to Deploy a Stand-alone EJB Module.....	16-15
16.5.6	How to Debug an EJB Module.....	16-15
16.5.7	How to Profile an EJB Module.....	16-16
16.5.8	How to Test an EJB Module .....	16-17
16.6	Developing EJB 2.1 Entity Beans .....	16-18

## **17 Developing with Java Persistence**

17.1	About Developing with Java Persistence .....	17-1
17.2	Creating a Persistence Unit.....	17-2
17.2.1	Scope of the Persistence Unit .....	17-2
17.2.2	Persistence Provider .....	17-3
17.2.3	Data Source.....	17-3
17.2.4	Transaction Types.....	17-3
17.2.5	How to Create a Persistence Unit.....	17-3
17.3	Creating an Entity Class.....	17-4
17.3.1	How to Create an Entity Class.....	17-4
17.3.2	How to Map Entity Classes .....	17-5
17.3.3	How to Generate Entity Classes from a Database .....	17-6
17.3.4	How to Obtain an Entity Manager.....	17-7
17.4	Generating JPA Controller Classes.....	17-7
17.4.1	How to Generate a JPA Controller Class from an Entity Class .....	17-8
17.5	Adding Support for Java Persistence .....	17-8
17.5.1	How to Add Support for Java Persistence to the Project.....	17-8

## **18 Developing Applications Using XML**

18.1	About Developing Applications Using XML .....	18-1
18.2	Creating and Editing XML Documents .....	18-1
18.2.1	How to Create a Well-formed XML Document .....	18-2
18.2.2	How to Create a DTD-constrained XML Document .....	18-2
18.2.3	How to Create an XML Schema-constrained Document.....	18-3
18.2.4	How to Edit an XML Document.....	18-3
18.2.5	How to Validate an XML Document .....	18-4
18.2.6	How to Check that an XML Document is Well-formed.....	18-4
18.3	Generating a DTD from an Existing XML File .....	18-5
18.4	Creating an Empty DTD .....	18-5
18.5	Generating Documentation for a DTD .....	18-5
18.6	Registering a Local DTD or XML Schema.....	18-6
18.7	Removing a DTD or XML Schema Entry from the User Catalog .....	18-6

## **19 Developing and Securing Web Services**

19.1	About Developing and Securing Web Services.....	19-1
------	---	------

19.2	Working with Web Services .....	19-2
19.3	Creating Web Services.....	19-3
19.3.1	How to Create SOAP (JAX-WS) Web Services.....	19-3
19.3.1.1	How to Add Operations to a JAX-WS Web Service .....	19-5
19.3.1.2	How to Use the JAX-WS Web Service Designer .....	19-6
19.3.2	How to Create RESTful Web Services .....	19-6
19.4	Configuring Web Services .....	19-7
19.4.1	Using Annotations and Deployment Descriptors.....	19-9
19.4.2	How to Configure a Web Service With Annotations .....	19-10
19.4.3	How to Configure a Web Service With Its Deployment Descriptor .....	19-11
19.5	Creating JAX-WS Web Service Clients .....	19-11
19.5.1	How to Work with JAX-WS Web Service Clients.....	19-12
19.5.2	How to Set a Proxy for Web Services and Clients .....	19-12
19.5.3	How to Generate a JAX-WS Web Service Client.....	19-15
19.5.4	How to Call a Web Service Operation.....	19-16
19.5.5	How to Asynchronously Call a Web Service Operation.....	19-16
19.5.6	How to Deploy a Web Service Client .....	19-17
19.6	Creating RESTful Web Service Clients .....	19-18
19.6.1	How to Work with RESTful Web Service Clients.....	19-18
19.6.2	How to Generate RESTful Web Service Java Clients .....	19-19
19.6.3	How to Generate RESTful Web Service JavaScript Clients .....	19-20
19.6.4	How to Use the Web Services Manager .....	19-21
19.7	Deploying and Testing Web Services and Clients .....	19-21
19.7.1	How to Test a JAX-WS Web Service .....	19-21
19.7.2	How to Test a RESTful Web Service .....	19-22
19.8	Creating Handlers.....	19-22
19.8.1	How to Create a Handler.....	19-23
19.8.2	How to Configure Handlers .....	19-23
19.8.3	Testing and Using Handlers .....	19-23
19.9	Using JAXB for Java-XML Binding .....	19-23
19.9.1	How to Generate Java Classes from XML Schema .....	19-24
19.9.2	How to Marshall XML Elements From Java Classes .....	19-25
19.9.3	How to Unmarshall XML Elements to Java Classes.....	19-25
19.10	Configuring Quality of Service .....	19-26
19.10.1	How to Configure Quality of Service .....	19-26
19.10.2	How to Set Port Binding .....	19-27
19.11	Securing an Operation.....	19-27
19.11.1	How to Secure an Operation's Input Messages .....	19-28
19.11.2	How to Secure a Message .....	19-28
19.11.3	How to Secure an Operation's Output Messages .....	19-29
19.11.4	How to Set Transport Options.....	19-30
19.11.5	How to Set Client Security .....	19-30

## 20 Developing HTML5/JavaScript Applications

20.1	About Developing HTML5 Applications.....	20-1
20.2	Working with HTML5/JavaScript Applications.....	20-2
20.3	Creating an HTML5/JavaScript Application Project .....	20-2

20.3.1	How to Create HTML5/JavaScript Applications for Mobile Platforms.....	20-2
20.3.2	How to Add Support for HTML5 Features to an Application.....	20-4
20.3.3	How to Create a Node.js Application.....	20-4
20.4	Running an HTML5 Application.....	20-5
20.4.1	How to Specify the Browser.....	20-5
20.4.2	How to Run an Application .....	20-6
20.4.3	How to Run a File .....	20-6
20.5	Integrating an HTML5 Project with a Browser .....	20-7
20.5.1	How to Install the Extension from the Chrome Web Store .....	20-7
20.5.2	How to Install the Extension Manually .....	20-7
20.6	Inspecting HTML5 Code with the Browser .....	20-8
20.7	Changing Browser Screen Sizes.....	20-9
20.7.1	How to Switch Between Screen Sizes .....	20-9
20.7.2	How to Create a Custom Screen Size.....	20-10
20.8	Creating HTML5 Templates.....	20-10
20.8.1	How to Create a Site Template .....	20-10
20.9	Creating Cascading Style Sheets.....	20-10
20.9.1	How to Create a Cascading Style Sheet.....	20-11
20.9.2	How to Add a CSS Rule to a Cascading Style Sheet.....	20-11
20.9.3	How to Add a Property to a CSS Rule.....	20-11
20.9.4	Creating a CSS Preprocessor File .....	20-12
20.9.5	How to Generate CSS Files Using a CSS Preprocessor .....	20-13
20.10	Creating JavaScript Files .....	20-14
20.10.1	How to Create a JavaScript File .....	20-14
20.10.2	How to Edit a JavaScript File .....	20-14
20.10.3	How to Debug a JavaScript File.....	20-15
20.10.4	How to Set JavaScript Breakpoints .....	20-16
20.10.5	How to Run Unit Tests on JavaScript Files .....	20-17
20.10.6	How to Add a JavaScript Library to a Project .....	20-20
20.11	Using Grunt and Gulp Build Tools .....	20-21
20.11.1	Installing Grunt.js .....	20-21
20.11.2	Installing Gulp.js.....	20-21
20.11.3	Running Grunt Tasks .....	20-22
20.11.4	Running Gulp Tasks.....	20-22

## 21 Developing PHP Applications

21.1	About Developing PHP Applications.....	21-1
21.1.1	Databases .....	21-1
21.1.2	Remote Development.....	21-1
21.1.3	Frameworks .....	21-2
21.2	Working with PHP Applications.....	21-2
21.2.1	How to Create a PHP Project .....	21-2
21.2.2	How to Create a Run Configuration for the Project .....	21-3
21.2.3	How to Synchronize Local and Remote Sources .....	21-4
21.2.4	How to Run a Project .....	21-4
21.2.5	How to Test a PHP Project .....	21-5
21.3	Editing PHP Files .....	21-5

21.3.1	How to Use Syntax Highlighting .....	21-5
21.3.2	How to Use Code Completion.....	21-6
21.3.2.1	Snippets.....	21-6
21.3.2.2	Context-Sensitive Proposals.....	21-6
21.3.2.3	Code Templates and Abbreviations .....	21-7
21.3.2.4	Code Completion for Constructors.....	21-7
21.3.2.5	SQL Code Completion.....	21-7
21.3.2.6	PHP Namespaces.....	21-7
21.3.2.7	Overridden and Implemented Methods .....	21-7
21.3.2.8	Annotations .....	21-7
21.3.3	How to Use Bracket Completion.....	21-8
21.3.4	How to Use Parameter Hints .....	21-8
21.3.5	How to Use the Go To Declaration Feature .....	21-8
21.3.6	How to Use Rename Refactoring and Instant Rename.....	21-9
21.3.7	How to Use Code Generators .....	21-9
21.3.7.1	Constructors .....	21-9
21.3.7.2	Getters and Setters.....	21-9
21.3.7.3	Overridden and Implemented Methods .....	21-10
21.3.8	How to Define Variable Type in Comments .....	21-10
21.3.9	About PHP 7 Support .....	21-10
21.4	Debugging PHP Applications.....	21-11
21.4.1	How to Debug a PHP Project.....	21-11
21.4.2	How to Set Up XDebug.....	21-11
21.4.2.1	Check If XDebug Is Installed .....	21-11
21.4.2.2	Install XDebug.....	21-12
21.4.2.3	Setting Up XDebug.....	21-12
21.4.2.4	Testing XDebug .....	21-12
21.4.2.5	Setting Up Debugging Options in NetBeans IDE .....	21-13
21.4.3	How to Set PHP Breakpoints in the IDE .....	21-13
21.4.3.1	Setting Breakpoints .....	21-13
21.4.4	How to Set the Current Context in the PHP Debugger .....	21-13
21.4.4.1	Debugger Windows and Context.....	21-14
21.4.4.2	The Source Editor and Context.....	21-14
21.4.5	How to Start a PHP Debugging Session.....	21-14
21.4.5.1	Starting a PHP Debugging Session.....	21-14
21.4.5.2	Remote PHP Debugging and Path Mapping .....	21-14
21.4.5.3	Stepping Through Your Program .....	21-15
21.4.5.4	Finishing a PHP Debugging Session .....	21-16
21.4.6	How to Use PHP Debugger Windows .....	21-16
21.4.6.1	Using the Breakpoints Window .....	21-16
21.4.6.1.1	Icons.....	21-17
21.4.6.1.2	Actions .....	21-17
21.4.6.1.3	Grouping Breakpoints .....	21-17
21.4.6.2	Using the Call Stack Window .....	21-17
21.4.6.2.1	Icons.....	21-18
21.4.6.3	Using the Threads Window .....	21-18
21.4.6.4	Using the Variables Window .....	21-18

21.4.6.4.1	Properties.....	21-18
21.4.6.5	Using the Watches Window.....	21-19
21.4.6.5.1	Properties.....	21-19
21.4.7	How to View Program Information.....	21-19
21.4.7.1	Viewing Variables When Debugging PHP Applications .....	21-19
21.4.7.2	Enabling and Creating Watches in PHP Code .....	21-20
21.5	Testing PHP Applications.....	21-21
21.5.1	How to Configure PHP Unit Test Frameworks .....	21-21
21.5.2	How to Run PHP Tests .....	21-21
21.5.3	How to Analyze PHP Code.....	21-22
21.6	Using a Continuous Build Server With PHP .....	21-22

## **22 Developing Java ME Applications**

22.1	About Developing Java ME Applications .....	22-1
22.1.1	About Java ME Embedded and CLDC Applications .....	22-3
22.2	Understanding Java Platform Manager.....	22-3
22.3	Creating Java ME Projects .....	22-4
22.3.1	How to Create a Java ME Embedded Application from a Template.....	22-4
22.3.1.1	Creating Embedded Application .....	22-5
22.3.2	How to Add a MIDlet to a Project Configuration.....	22-5
22.3.3	How to Add Push Registry Entries.....	22-5
22.3.4	Adding API Permissions .....	22-6
22.4	Customizing MIDlet Properties.....	22-7
22.4.1	How to Customize the Application Descriptor (JAD) Attributes .....	22-8
22.4.2	How to Customize MEEP Javadoc Settings.....	22-8
22.5	Building Java ME Embedded Applications .....	22-8
22.5.1	How to Customize Compilation Options .....	22-9
22.5.2	How to Add Libraries and Resources to a Java ME Embedded Project.....	22-10
22.5.3	How to Customize MEEP Build Filter Settings.....	22-11
22.5.4	How to Obfuscate a MIDlet Suite.....	22-11
22.5.5	How to Customize the JAR and JAD Files in a Build.....	22-12
22.6	Working with Security and MIDlet Signing .....	22-12
22.6.1	How to Set Security through MIDlet Signing.....	22-13
22.6.2	How to Add or Creating a Keystore .....	22-13
22.6.3	How to Create a New Key Pair.....	22-14
22.6.4	How to Export a Key to an Emulator Platform .....	22-15
22.7	Working with Java ME Embedded Project Configurations.....	22-15
22.7.1	How to Work with Java ME Embedded Project Configurations.....	22-16
22.7.2	How to Customize Java ME Embedded Project Configurations.....	22-16
22.7.2.1	Customizing Project Configurations .....	22-16
22.7.2.2	Customizing Platform Properties.....	22-17
22.7.2.3	Customizing the Application Descriptor (JAD) Attributes.....	22-18
22.7.2.4	Adding MIDlets to a Project Configuration .....	22-18
22.7.2.5	Customizing Compilation Options.....	22-19
22.7.2.6	Adding Libraries and Resources to a Java ME Embedded Project .....	22-19
22.8	Running Java ME Embedded Applications .....	22-20
22.8.1	How to Customize Java ME Embedded Project Run Options .....	22-20

22.8.2	How to Run a MIDlet Suite with OTA Provisioning .....	22-20
22.8.3	How to Run Headless Builds .....	22-21
22.9	Using Java ME Emulator Platforms .....	22-21
22.9.1	How to Add MEEP Emulator Platforms .....	22-21
22.9.2	How to Customize Java ME Platform Properties .....	22-22
22.9.3	How to Use the Java ME SDK.....	22-23

## **23 Working with Web and Application Servers**

23.1	About Working with Web and Application Servers .....	23-1
23.1.1	Web Browsers.....	23-1
23.1.2	Application Servers .....	23-2
23.1.3	Understanding Connection Pools .....	23-3
23.1.4	Common Application Server Tasks .....	23-3
23.1.4.1	Setting Context Paths .....	23-4
23.1.4.2	Starting and Stopping Servers .....	23-4
23.1.4.3	Removing Applications from Servers .....	23-5
23.1.5	Developing on the Cloud.....	23-5
23.1.6	HTTP Server-Side Monitor.....	23-6
23.2	Working with Web Browsers .....	23-6
23.2.1	How to Configure a Web Browser .....	23-6
23.2.2	How to Change the Default Web Browser.....	23-6
23.2.3	How to Open a Web Browser from the IDE .....	23-7
23.2.4	How to Set a Proxy .....	23-7
23.3	Working with Glassfish Application Servers .....	23-8
23.3.1	How to Register a Server Instance .....	23-9
23.3.2	How to View the Server in the IDE.....	23-11
23.3.3	How to Start and Stop the Server .....	23-12
23.3.4	How to View the Server Log File .....	23-12
23.3.5	How to Access the Server Admin Console .....	23-12
23.3.6	How to Modify Server Properties .....	23-13
23.3.7	How to Set up a Connection Pool .....	23-13
23.3.8	How to Set up a JDBC Resource.....	23-14
23.3.9	How to Set up a JMS Resource .....	23-15
23.3.10	How to Set Up a JavaMail Session .....	23-16
23.3.11	How to Register and Delete Resources.....	23-17
23.3.12	How to Manage Users.....	23-17
23.3.13	How to Configure Security Roles.....	23-18
23.3.14	How to Download and Install Server Updates .....	23-19
23.4	Working with Oracle WebLogic Application Servers .....	23-20
23.4.1	How to Register a Server Instance .....	23-20
23.4.2	How to View the Server in the IDE.....	23-21
23.4.3	How to Start and Stop the Server .....	23-22
23.5	Working with JBoss Application Servers .....	23-22
23.5.1	How to Register a Server Instance .....	23-22
23.5.2	How to View the Server in the IDE.....	23-23
23.5.3	How to Start and Stop the Server.....	23-23
23.5.4	How to Set up a Connection Pool .....	23-24

23.5.5	How to Access Application Server Utilities.....	23-25
23.6	Working with Tomcat Web Servers .....	23-25
23.6.1	How to Register a Server Instance .....	23-26
23.6.2	How to View the Server in the IDE.....	23-27
23.6.3	How to Start and Stop the Server .....	23-28
23.6.4	How to Configure the Server Properties .....	23-29
23.6.5	How to Edit the Configuration File.....	23-31
23.6.6	How to Authenticate the Server .....	23-32
23.6.7	How to Set up a Connection Pool.....	23-32
23.7	Working with Web Applications on the Cloud.....	23-34
23.7.1	About Web Applications on the Cloud .....	23-35
23.7.2	How to Register a Cloud Account in the IDE.....	23-35
23.7.3	How to Develop Cloud Applications Locally .....	23-36
23.7.4	How to Deploy Web Applications to the Cloud .....	23-36
23.8	Working with the HTTP Server-Side Monitor.....	23-37
23.8.1	How to Set up the HTTP Server-Side Monitor.....	23-37
23.8.2	How to Analyze Session Data.....	23-39
23.8.3	How to Save Session Data .....	23-40
23.8.4	How to Replay Session Data .....	23-41

## **24 Working and Connecting with Databases**

24.1	About Working and Connecting with Databases .....	24-1
24.2	Working with the Database Tools .....	24-1
24.2.1	How to Browse Database Structures .....	24-2
24.2.2	How to Create Database Tables.....	24-3
24.2.3	How to Create Database Views .....	24-4
24.2.4	How to Add Columns to a Database .....	24-4
24.2.5	How to Delete Database Objects .....	24-5
24.2.6	How to View and Modify Data in a Database.....	24-5
24.2.7	How to Execute SQL Statements .....	24-6
24.2.8	How to Obtain a Database Schema .....	24-7
24.2.9	How to Recapture a Schema from a Database .....	24-8
24.3	Setting up a Database Connection.....	24-8
24.3.1	How to Add a JDBC Driver.....	24-9
24.3.2	How to Connect to the Java DB Database.....	24-10
24.3.3	How to Connect to Oracle Database.....	24-12
24.3.4	How to Use MySQL Database with the IDE.....	24-13
24.3.5	How to Enable Debug Mode.....	24-14

---

---

# Preface

Welcome to the *Developing Applications with NetBeans IDE User's Guide*.

## Audience

This document is intended for all users of NetBeans IDE and provides detailed information on developing applications using the IDE.

## Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at  
<http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc>.

### Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit  
<http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info> or visit  
<http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs> if you are hearing impaired.

## Related Documents

For more information, see the following NetBeans IDE 8.2 documentation and resources:

- NetBeans IDE 8.2 Release Information
- NetBeans IDE 8.2 Installation Instructions
- NetBeans IDE 8.2 Release Notes
- NetBeans IDE Documentation, Training & Support
- NetBeans User FAQs

## Conventions

The following text conventions are used in this document:

Convention	Meaning
<b>boldface</b>	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.

Convention	Meaning
<i>italic</i>	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
monospace	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

---

---

## What's New in This Guide

The following topics introduce the new and changed features of NetBeans IDE 8.1 and other significant changes that are described in this guide.

### New and Changed Features for Release 8.2

NetBeans IDE 8.2 includes the following new and changed features for this document.

- Developing C/C++ and Fortran Applications
  - This document does not contain specific information about using the IDE for developing applications in C, C++, and Fortran. For information about developing applications in C, C++, and Fortran, see the "C/C++ Fortran Development" section in the IDE's integrated help viewer and search the help for C/C++ topics.
  - An additional source for information about developing applications in C, C++, and Fortran is the C/C++ Learning Trail on the netbeans.org site.  
<https://netbeans.org/kb/trails/cnd.html>
- Support for multiple carets in the IDE's Source Editor. See [Table 2-24, "Using Multiple Carets"](#).
- Profiler improvements. See [Chapter 9, "Testing and Profiling Java Application Projects"](#).
- Debugger enhancements. See [Section 10.9.4.4, "About Source Maps Support"](#) and [Section 10.9.6.7, "How to Pin a Watch"](#).
- Docker platform support. See [Section 14.9, "About Docker Platform Support"](#).
- Enhanced support for Grunt and Gulp. See [Section 20.11, "Using Grunt and Gulp Build Tools"](#).
- Support for ECMA Script 6. See [Section 20.10.2, "How to Edit a JavaScript File"](#).
- PHP 7 support. See [Section 21.3.9, "About PHP 7 Support"](#).



---

# Introduction to NetBeans IDE

This chapter provides an overview of NetBeans IDE.

This chapter contains the following sections:

- [About NetBeans IDE](#)
- [NetBeans IDE Developer Resources](#)

## 1.1 About NetBeans IDE

NetBeans IDE is a free, open source, integrated development environment (IDE) that enables you to develop desktop, mobile and web applications. The IDE supports application development in various languages, including Java, HTML5, PHP and C++. The IDE provides integrated support for the complete development cycle, from project creation through debugging, profiling and deployment. The IDE runs on Windows, Linux, Mac OS X, and other UNIX-based systems.

The IDE provides comprehensive support for JDK 8 technologies and the most recent Java enhancements. It is the first IDE that provides support for JDK 8, Java EE 7, and JavaFX 2. The IDE fully supports Java EE using the latest standards for Java, XML, Web services, and SQL and fully supports the GlassFish Server, the reference implementation of Java EE.

## 1.2 NetBeans IDE Developer Resources

This section provides resources designed to get you up and running quickly on NetBeans IDE. There are various channels for learning more about the IDE and for providing feedback to the development team.

**Table 1–1 NetBeans IDE Developer Resources**

Resource	Description
netbeans.org	The web site for NetBeans IDE contains information and news for the community of NetBeans users: <a href="https://www.netbeans.org">https://www.netbeans.org</a>
NetBeans IDE Release Documentation	The IDE release documentation provides details on system requirements, supported technologies, installation instructions and known issues for the release: <a href="https://netbeans.org/community/releases/80/">https://netbeans.org/community/releases/80/</a>

**Table 1–1 (Cont.) NetBeans IDE Developer Resources**

Resource	Description
NetBeans IDE Documentation and Tutorials	The Documentation section of the netbeans.org website contains screencasts and tutorials that provide step-by-step instructions for developing applications with NetBeans IDE. The list of current tutorials are located at: <a href="https://netbeans.org/kb/index.html">https://netbeans.org/kb/index.html</a>
NetBeans IDE wiki	The NetBeans wiki provides additional documentation for the NetBeans community that is not included in the NetBeans User's Guide: <a href="http://wiki.netbeans.org/Main_Page">http://wiki.netbeans.org/Main_Page</a> You can find information on installation, configuration, licenses and trouble-shooting at the NetBeans User FAQs: <a href="http://wiki.netbeans.org/NetBeansUserFAQ">http://wiki.netbeans.org/NetBeansUserFAQ</a> You can find information on plugin development and NetBeans Platform API documentation at the NetBeans Developer FAQs: <a href="http://wiki.netbeans.org/NetBeansDeveloperFAQ">http://wiki.netbeans.org/NetBeansDeveloperFAQ</a>

# 2

---

## Working with NetBeans IDE

This chapter describes the basics of using and configuring NetBeans IDE.

This chapter contains the following sections:

- [About Working with NetBeans IDE](#)
- [Working with the Options Window](#)
- [Managing IDE Windows](#)
- [Customizing Toolbars in the IDE](#)
- [Working with Keyboard Shortcuts](#)
- [Understanding the Source Editor Features](#)
- [Setting Startup Parameters](#)
- [Setting Fonts and Colors for IDE Windows](#)
- [Managing Plugins in the IDE](#)
- [Displaying IDE Help in a Web Browser](#)
- [Internationalizing Source Code](#)
- [Managing and Creating Projects](#)
- [Working with Source Files in the IDE](#)
- [Working with Resource Bundles](#)
- [Working with Javadoc Documentation](#)
- [Viewing IDE Notifications](#)

### 2.1 About Working with NetBeans IDE

The IDE enables you to configure options for its many areas of functionality as well as customize its workspace. The primary tools you can use to configure and update the IDE include the following:

- **Options Window.** The Options window enables you to set general IDE settings, edit any of the IDE's configurable settings and set keyboard shortcuts. You open the Options window by choosing **Tools > Options**. (On Mac OS X, choose NetBeans > Preferences.)
- **Plugins Manager.** The Plugins manager enables you to enable and disable installed plugins and add new or updated plugins to the IDE.

For more information, see [Section 2.9, "Managing Plugins in the IDE"](#).

You can also pass startup parameters to the IDE launcher in the *IDE-HOME/etc/netbeans.conf* file by using startup switches. For more information, see [Section 2.7, "Setting Startup Parameters"](#).

## 2.2 Working with the Options Window

The Options window enables you to change any of the IDE's configurable settings. You select a category at the top of the window to display its settings in the lower pane. Depending on the category, you can click on the tabs in the lower pane to access additional settings. The changes are applied when you click **Apply**.

### 2.2.1 How to Edit IDE Settings

You use the Options window to specify global settings for the IDE. Settings that you specify at the project level override the settings that you specify in the Options window.

**To edit IDE settings in the Options window:**

1. Open the Options window by choosing **Tools > Options** from the main menu. (If you are running on Mac OS X, choose **NetBeans > Preferences**.)
2. Select the category in the upper pane to display the configurable settings.
3. Modify the settings.
4. Click **Apply**.

### 2.2.2 How to Export IDE Settings as Zip Archive

You can export your IDE settings as an archive that you can use as a backup or that you can import into an installation of the IDE on another machine.

**To export IDE settings as a zip archive:**

1. Open the Options window by choosing **Tools > Options** from the main menu. (If you are running on Mac OS X, choose **NetBeans > Preferences**.)
2. Click **Export** at the bottom of the Options window.
3. Specify the location and name of the zip archive that you want to create to contain your settings.
4. Select the Options categories that you want to export. Click **OK**.

### 2.2.3 How to Import IDE Settings

When you launch a new version of the IDE for the first time, you are prompted with the option of importing settings from a previous version of the IDE. This prompt occurs only if you have a user directory on your system from the previous version of the IDE and the user directory is in the default location.

The settings that are imported include the following items:

- Keyboard shortcuts
- Most Source Editor font and color settings
- Java Platform Manager contents
- Library Manager contents

- Database drivers
- Servers

You can recreate the conditions of the first launch at any time by deleting your current userdir. When you restart the IDE you will be prompted with the option of importing settings from a previous version of the IDE.

For more details about locating your current userdir, see the following FAQ.

<http://wiki.netbeans.org/FaqWhatIsUserdir>

---

**Note:** If you delete your userdir you could lose other important settings. In addition, you might also lose any modules you have installed through the Update Center in the new version and you may need to reinstall or reactivate those modules.

---

If you do not import the settings at the first launch but later decide that you would like to import the settings from a previous installation, you can choose the settings that you would like to import in the Options window.

Importing settings may overwrite your existing settings. To prevent possible loss of your settings, use the Export settings function to create a backup of your IDE settings.

**To import IDE settings:**

1. Open the Options window by choosing **Tools > Options** from the main menu. (If you are running on Mac OS X, choose **NetBeans > Preferences**.)
2. Click **Import** at the bottom of the Options window.
3. Click **Browse** and locate either of the following:
  - the zip archive that contains your settings
  - the userdir of the IDE installation that has the settings that you want to import
4. Select the Options categories that you want to import. Click **OK**.

You will need to restart the IDE to apply the imported settings.

## 2.3 Managing IDE Windows

Each window in the IDE appears as a tab in the pane in which it resides. The IDE's windowing system enables you to arrange windows anywhere in the IDE by dragging and dropping. The IDE remembers the position of both manually and automatically closed windows the next time they are opened.

Some windows only appear when you are performing a task to which they are related. For example, the Debugger windows only appear when you are in a debugging session. You can manually open task-related windows so that they are always open.

To open a task-related window, simply choose the window from the Windows menu.

The following table lists some keyboard shortcuts that you can use to navigate, activate and select components in IDE windows.

**Table 2–1 Keyboard Shortcut for Managing Windows**

Keys	Action
Ctrl-0	Switch to Editor window

**Table 2–1 (Cont.) Keyboard Shortcut for Managing Windows**

<b>Keys</b>	<b>Action</b>
Ctrl-1	Switch to Project window
Ctrl-2	Switch to Files window
Ctrl-3	Switch to Favorites window
Ctrl-4	Switch to Output window
Ctrl-5	Switch to Services window
Ctrl-6	Switch to Action Items window
Ctrl-7	Switch to Navigator window
Ctrl-Shift-5	Switch to HTTP Monitor window
Ctrl-Shift-7	Switch to Properties window
Ctrl-Shift-1	Select file in Projects window
Ctrl-Shift-2	Select file in Files window
Ctrl-Shift-3	Select file in Favorites window
Ctrl-Shift-8	Opens the Palette
Ctrl-F4	Close window
Ctrl-Shift-F4	Close all documents
Shift-F4	Open Documents dialog box
Shift-Escape	Maximize window
Ctrl-Tab	Switch to recent window
Ctrl-W	Closes the current tab in the current window. If the window has no tabs, the whole window is closed.
Alt-right	Displays the next tab in the current window.
Alt-left	Displays the previous tab in the current window.
Up arrow	Moves keyboard focus to the previous item in a group of items. Navigates to the previous setting in a drop-down list in a property sheet.
Down arrow	Moves keyboard focus to the next item in a group of items. Navigates to the next setting in a drop-down list in a property sheet.
Left arrow	Closes a folder (node).
Right arrow	Opens a folder (node).
F1	Show Help for selected component

### 2.3.1 How to Move a Window

The windowing system in the IDE enables you to drag window components to any location in the IDE.

**To move a window:**

1. Click the window header and drag the window to the desired position. A red preview box indicates the new location of the window after you release it.
2. Drop the window.

### 2.3.2 How to Configure Window Behavior and Appearance

**How to configure window behavior and appearance**

1. Choose Tools > Options from the main menu.
2. Click the Windows tab in the Appearance category.

### 2.3.3 How to Simultaneously Display Multiple Files in the Editor

You can split the Source Editor, which enables you to work with multiple files or different areas of the same file simultaneously. The Source Editor can be split horizontally or vertically.

**To simultaneously display multiple files in the Source Editor:**

1. Open two or more files in the Source Editor.
2. Click the tab of one of the files and drag it to the edge of the Source Editor pane where you want the file to be placed. A red preview box indicates the new position of the window.
3. Release the mouse button to drop the window when the red preview box indicates the window is in the new desired position.

**To create a group of tabs:**

1. Open two or more files in the Source Editor.
2. Right-click the tab for one of the files and choose New Document Tab Group.

### 2.3.4 How to Clone the View of a Single File

You can create a clone of a tab if you want to open two tabs with the same file. If you drag one of the tabs to split the editor view or create a tab group you can view different parts of the same file simultaneously.

**How to clone a tab:**

1. Right-click the document tab in the Source Editor and choose Clone.
2. Click the tab of the cloned document and drag it to the part of the window where you want the copy to be placed.

### 2.3.5 How to Manage Open Files

**How to navigate between open files:**

1. Choose Window > Documents from the main menu.
2. Select a document from the list in the Documents window.
3. Click Switch to Document.

**How to close or save open files:**

1. Choose Window > Documents from the main menu.
2. Select a document or documents from the list in the Documents window.

You can use the ctrl and shift keys to select multiple documents.

3. Click **Close Document(s)** or **Save Document(s)**.

## 2.4 Customizing Toolbars in the IDE

The IDE provides comprehensive control over how toolbars are presented. You can add and remove toolbars and position and control the visibility and contents of toolbars. You can also create customized toolbar configurations.

Choose **View > Toolbars** from the main menu to see a list of the available toolbars. The submenu displays all the toolbars in the IDE.

### 2.4.1 How to Show or Hide a Toolbar

Toolbars that are currently visible are indicated by a check mark in the Toolbars submenu.

**To show or hide a toolbar:**

- Choose **View > Toolbars** from the main menu and choose a toolbar name in the submenu.  
Alternatively, right-click the empty space in the toolbar area and choose the toolbar name from the pop-up menu.

To find the name of a toolbar, rest the pointer on the textured drag area of the toolbar to display a tool tip with the toolbar name.

### 2.4.2 How to Move a Toolbar

**To move a toolbar:**

- Click in the textured drag area on the left side of the toolbar and drag the toolbar to the desired location.

### 2.4.3 How to Add a Toolbar Button to a Toolbar

You can add buttons for many IDE commands and actions to the toolbar of the IDE.

**To add a toolbar button to a toolbar:**

1. Choose **View > Toolbars > Customize** to open the Customize Toolbars window.
2. Select the button of the action you want to add to the IDE toolbar and drag and drop the button onto the toolbar in the main window.

### 2.4.4 How to Remove a Button from a Toolbar

The Customize Toolbars window must be open when you are removing toolbar buttons.

**To remove a toolbar button from a toolbar:**

1. Choose **View > Toolbars > Customize** to open the Customize Toolbars window.
2. In the main window, click and drag the toolbar button above the toolbar.  
Alternatively, you can drag the button from the toolbar into the Customize Toolbars window.

## 2.4.5 How to Reorder a Button in a Toolbar

You can customize the order that buttons appear in the toolbar.

**To reorder a button in a toolbar:**

1. Choose **View > Toolbars > Customize** to open the Customize Toolbars window.
2. In the main window, drag and drop the button to the desired position.

## 2.4.6 How to Change the Size of Toolbar Buttons

You can select either small or large icons for the toolbar buttons.

**To change the size of toolbar buttons:**

- Choose **View > Toolbars > Small Toolbar Icons**  
Alternately, right-click the empty space in the toolbar area and choose **Small Toolbar Icons** from the popup menu.

## 2.4.7 How to Add a New Toolbar to the Main Window

You can create custom toolbars to group sets of buttons.

**To add a new toolbar to the main window:**

1. From the main window, choose **View > Toolbars > Customize**.
2. In the Customize Toolbars window, click **New Toolbar**.
3. In the New Toolbar dialog box, type a name for the toolbar and click **OK**.
4. Drag icons from the Customize Toolbars window into the new toolbar.

The new toolbar must have at least one icon.

## 2.5 Working with Keyboard Shortcuts

You can edit keyboard settings for individual commands or switch between preconfigured sets of keyboard shortcuts. The shortcut sets are designed for users that are already used to the keyboard shortcuts of other editors and IDEs.

### 2.5.1 How to Add or Remove a Keyboard Shortcut for a Menu Command

You can use the Options window to customize the keyboard shortcuts for menu commands.

**To add or remove a keyboard shortcut for a menu command:**

1. From the main window, choose **Tools > Options**.
2. Click **Keymap** in the Options window.
3. Locate the command for which you want to change a keyboard shortcut.
4. Double-click in the text field in the Shortcut column of the command and type the new keyboard shortcut.

As you press the key sequence, the correct syntax for that sequence automatically appears in the text field. If you simultaneously hold down the Alt key, the Control key, and the J key, "Alt+Ctrl+J" appears. You can only specify a keyboard shortcut that is not being used by another command.

---

**Note:** To set keyboard shortcuts for recorded macros, go to **Editor > Macros** in the Options window, select the macro name and edit the keyboard shortcut.

---

## 2.5.2 How to Customize Keyboard Shortcuts

### To customize keyboard shortcuts:

1. Choose Tools > Options > Keypad.
2. Do either of the following:
  - Select a predefined set of keyboard shortcuts, which is called Profile.
  - Edit particular keyboard shortcuts.

You can save customized sets of your shortcuts as profiles. Then, you can switch from one profile to another to quickly change multiple settings.

### To create a custom profile of keyboard shortcuts:

1. In the **Options > Keypad** window, click **Manage** profiles.
2. Select the profile you want to use as a base for your new profile and click **Duplicate**.
3. Enter the new profile name and click **OK**.
4. Ensure that the new profile is selected and modify the shortcuts you need.  
To edit a shortcut, double-click in the **Shortcut** field or click the ellipsis button (...). As you press the sequence of keys, the syntax for them is added.  
If you want to add special characters, such as Tab, Escape, or Enter, click the ellipsis button (...) again and select the key from the pop-up window.
5. When finished editing, click **OK** in the **Options** window.

---

**Note:** To find a shortcut for a specific command, type the command name in the **Search** field. To find a command by a combination, insert the cursor in the **Search in Shortcuts** field and press the shortcut key combination.

---

## 2.5.3 How to Switch Between Keyboard Shortcut Sets

You can use the Options window to switch between any of the existing keyboard shortcuts sets. The IDE includes several sets of keyboard shortcuts with shortcut mappings that correspond to the shortcuts in other editors and IDEs.

### To switch between keyboard shortcut sets:

1. From the main window, choose **Tools > Options**.
2. Click **Keypad** in the Options window.
3. Choose an existing keyboard set from the **Profiles** drop-down list at the top of the dialog box.

## 2.5.4 How to Create a New Keyboard Shortcut Set

You can configure the IDE to have multiple sets of keyboard shortcuts. You can use the Options window to create custom sets of keyboard shortcuts.

**To create a new keyboard shortcut set:**

1. From the main window, choose **Tools > Options**.
2. Click **Keymap** in the Options window.
3. Choose an existing keyboard set from the Profiles drop-down list at the top of the dialog box.
4. Click **Duplicate** to duplicate the currently selected shortcut set. Then edit the shortcuts in the new copy.

## 2.5.5 How to Use the Default Menu Shortcuts

There are keyboard shortcuts available to activate the following menu commands and other general commands:

**Table 2–2 File Menu**

Keys	Command	Action
Ctrl-Shift-N	New	Creates a new project with the New Project wizard.
Ctrl-N	New	Creates a new file with the New File wizard.
Ctrl-Shift-O	Open File	Opens an existing project.
Ctrl-S	Save	Saves the current file.
Ctrl-Shift-S	Save All	Saves all files.
Ctrl-Alt-Shift-P	Print	Print the current file.

**Table 2–3 Edit Menu**

Keys	Command	Action
Ctrl-Z	Undo	Reverses (one at a time) a series of editor actions, except Save.
Ctrl-Y	Redo	Reverses (one at a time) a series of Undo commands.
Ctrl-X	Redo	Reverses (one at a time) a series of Undo commands.
Ctrl-C	Copy	Copies the current selection to the clipboard.
Ctrl-V	Paste	Pastes the contents of the clipboard into the insertion point.
Ctrl-Shift-V	Paste Formatted	Pastes the formatted contents of the clipboard into the insertion point.
Delete	Delete	Deletes the current selection.
Select All	Ctrl-A	Selects everything in the current document or window.
Select Identifier	Alt-Shift-J	Selects the current identifier.
Ctrl-F3	Find Selection	Finds instances of the current selection.
F3	Find Next	Finds next instance of found text.
Shift-F3	Find Previous	Finds previous instance of found text.
Ctrl-F	Find	Finds previous instance of found text.

**Table 2–3 (Cont.) Edit Menu**

<b>Keys</b>	<b>Command</b>	<b>Action</b>
Ctrl-H	Replace	Finds a string of text and replaces it with the string specified.
Alt-F7	Find Usages	Finds usages and subtypes of selected code.
Ctrl-Shift-F	Find in Projects	Finds specified text, object names, object types within projects.
Ctrl-Shift-H	Replace in Projects	Replaces text, object names, object types within projects.

**Table 2–4 View Menu**

<b>Keys</b>	<b>Command</b>	<b>Action</b>
Ctrl-Minus	Collapse Fold	If the insertion point is in a foldable section of text, collapses those lines into one line.
Ctrl-Plus	Expand Fold	If the currently selected line in the Source Editor represents several folded lines, expands the fold to show all of the lines.
Ctrl-Shift-Minus	Collapse All	Collapses all foldable sections of text in the Source Editor.
Ctrl-Shift-Plus	Expand All	Expands all foldable sections of text in the Source Editor.
Alt-Shift-Enter	Full Screen	Expand window to full length and breadth of screen.

**Table 2–5 Navigate Menu**

<b>Keys</b>	<b>Command</b>	<b>Action</b>
Alt-Shift-O	Go to File	Find and open a specific file.
Ctrl-O	Go to Type	Find and open a specific class or interface.
Ctrl-Alt-Shift-O	Go to Symbol	Find and open a specific symbol.
Alt-Shift-B	Go to Spring Bean	Find and open a specific Spring bean.
Ctrl-Shift-T	Go to Test	Find and open a specific test.
Ctrl-Back Quote	Go to Previous Document	Open the document last opened before the current one.
Ctrl-Shift-B	Go to Source	Displays the source file containing the definition of the selected class.
Ctrl-B	Go to Declaration	Jump to the declaration of the item under the cursor.
Ctrl-Shift-P	Go to Super Implementation	Jump to the super implementation of the item under the cursor.
Ctrl-Q	Last Edit Location	Scroll the editor to the last place where editing occurred.
Alt-Left	Back	Navigate back
Alt-Right	Forward	Navigate forward
Ctrl-G	Go to Line	Jump to the specified line.
Ctrl-Shift-M	Toggle Bookmark	Set a bookmark on a line of code.
Ctrl-Shift-Period	Next Bookmark	Cycle forward through the bookmarks.
Ctrl-Shift-Comm a	Previous Bookmark	Cycle backwards through the bookmarks.

**Table 2–5 (Cont.) Navigate Menu**

<b>Keys</b>	<b>Command</b>	<b>Action</b>
Ctrl-Period	Next Error	Scrolls the Source Editor to the line that contains the next build error.
Ctrl-Comma	Previous Error	Scrolls the Source Editor to the line that contains the previous build error.
Ctrl-Shift-1	Select in Projects	Opens Projects window and selects current document within it.
Ctrl-Shift-2	Select in Files	Opens Files window and selects current document within it.
Ctrl-Shift-3	Select in Favorites	Opens Favorites window and selects current document within it.

**Table 2–6 Source Menu**

<b>Keys</b>	<b>Command</b>	<b>Action</b>
Alt-Shift-F	Format	Formats the selected code or the entire file if nothing is selected.
Alt-Shift-Left	Shift Left	Moves the selected line or lines one tab to the left.
Alt-Shift-Right	Shift Right	Moves the selected line or lines one tab to the right.
Alt-Shift-Up	Move Up	Moves the selected line or lines one line up.
Alt-Shift-Down	Move Down	Moves the selected line or lines one line down.
Ctrl-Shift-Up	Duplicate Up	Copy the selected line or lines one line up.
Ctrl-Shift-Down	Duplicate Down	Copy the selected line or lines one line down.
Ctrl-Slash or Ctrl-Shift-C	Toggle Comment	Toggles the commenting out of the current line or selected lines.
Ctrl-Space	Complete Code	Shows the code completion box.
Alt-Insert	Insert Code	Pops up a context aware menu that you can use to generate common structures such as constructors, getters, and setters.
Alt-Enter	Fix Code	Display editor hints. The IDE informs you when a hint is available when the light bulb is displayed.
Ctrl-Shift-I	Fix Imports	Generates the import statements required by the classes specified in the file.
Ctrl-P	Show Method Parameters	Selects the next parameter. You must have a parameter selected (highlighted) for this shortcut to work.
Ctrl-Shift-Space	Show Documentation	Show documentation for item under the cursor.
Ctrl-Shift-K	Insert Next Matching Word	Generates the next word used elsewhere in your code as you type its beginning characters.
Ctrl-K	Insert Previous Matching Word	Generates the previous word used elsewhere in your code as you type its beginning characters.

**Table 2–7 Refactor Menu**

<b>Keys</b>	<b>Command</b>	<b>Action</b>
Ctrl-R	Rename	Inplace rename.

**Table 2–7 (Cont.) Refactor Menu**

<b>Keys</b>	<b>Command</b>	<b>Action</b>
Ctrl-M	Move	Inplace move.
Alt-Delete	Safely Delete	Before deleting, display references.

**Table 2–8 Run Menu**

<b>Keys</b>	<b>Command</b>	<b>Action</b>
F6	Run Project	Runs the main project or the currently selected project if no main project is set.
Alt-F6	Test Project	Starts unit test for project.
F11	Build Project	Builds the main project or the currently selected project if no main project is set.
Shift-F11	Clean and Build Project	Deletes all previously compiled files and distributable outputs and builds the main project or the currently selected project if no main project is set.
Shift-F6	Run File	Runs the currently selected file.
Ctrl-F6	Test File	Starts unit test for current file.
F9	Compile File	Compiles the file. If you select a folder, the IDE compiles only the files that are new or have changed since the last compile.
Alt-F9	Check File	Checks file dependencies in the currently selected project when building.
Alt-Shift-F9	Validate File	Validates file dependencies in the currently selected project when building.
Ctrl-F11	Repeat Build/Run	Runs/builds the currently selected project once again.

**Table 2–9 Debug Menu**

<b>Keys</b>	<b>Command</b>	<b>Action</b>
Ctrl-F5	Debug Project	Debugs the main project or the currently selected project if no main project is set.
Ctrl-Shift-F5	Debug File	Starts debugging session for currently selected file.
Ctrl-Shift-F6	Debug Test File	Starts debugging test for file.
Shift-F5	Finish Debugger Session	Ends the debugging session.
F5	Continue	Resumes debugging until the next breakpoint or the end of the program is reached.
F8	Step Over	Executes one source line of a program. If the line is a method call, executes the entire method then stops.
Shift-F8	Step Over Expression	Steps over the expression and then stops the debugging.
F7	Step Into	Executes one source line of a program. If the line is a method call, executes the program up to the method's first statement and stops.
Shift-F7	Step into Next Method	Runs the current project to the specified method and then steps into the method.

**Table 2–9 (Cont.) Debug Menu**

<b>Keys</b>	<b>Command</b>	<b>Action</b>
Ctrl-F7	Step Out	Executes one source line of a program. If the line is a method call, executes the methods and returns control to the caller.
F4	Run to Cursor	Runs the current project to the cursor's location in the file and stop program execution.
Ctrl-F8	Toggle Line Breakpoint	Adds a line breakpoint or removes the breakpoint at the cursor location in the program.
Ctrl-Shift-F8	New Breakpoint	Sets a new breakpoint at the specified line, exception, or method.
Ctrl-Shift-F7	New Watch	Adds the specified variable to watch.
Ctrl-F9	Evaluate Expression	Opens the Evaluate Expression dialog box.
Ctrl-Alt-Up	Make Callee Current	Makes the method being called the current call. Only available when a call is selected in the Call Stack window.
Ctrl-Alt-Down	Make Caller Current	Makes the calling method the current call. Only available when a call is selected in the Call Stack window.

**Table 2–10 Profile Menu**

<b>Keys</b>	<b>Command</b>	<b>Action</b>
Alt-F2	Profile Project	Profiles the main project or the currently selected project if no main project is set.
Ctrl-Shift-F2	Rerun Profiling Session	Enables you to start a new profiling session that uses the settings from the previous profiling session. The shortcut for this action is active only when no profiling session is in progress.
Alt-Shift-F2	Modify Profiling Session	Opens the Select Profiling Task when a profiling session is in progress. You can modify the profiling task or criteria without stopping the application that you are profiling. The shortcut invokes this action only when no profiling session is in progress.
Ctrl-F2	Take Snapshot of Collected Results	Takes a snapshot of the collected profiling results and displays the results in a new window.

**Table 2–11 Tools Menu**

<b>Keys</b>	<b>Command</b>	<b>Action</b>
Ctrl-Shift-J	Insert Internationalized String	Enables you to add an internationalization string as you create the source.

**Table 2–12 Window Menu**

<b>Keys</b>	<b>Command</b>	<b>Action</b>
Ctrl-0	Editor	Switches to the Source Editor.
Ctrl-1/Ctrl-Shift-1	Projects	Opens the Projects window.

**Table 2–12 (Cont.) Window Menu**

<b>Keys</b>	<b>Command</b>	<b>Action</b>
Ctrl-2/Ctrl-Shift- -2	Files	Opens the Files window.
Ctrl-3/Ctrl-Shift- -3	Favorites	Opens the Favorites window.
Ctrl-4	Output Window	Opens the Output window.
Ctrl-5	Services	Opens the Services window.
Ctrl-Shift-6	Tasks	Opens the Tasks window.
Ctrl-6	Action Items	Opens the Action Items window.
Ctrl-7	Navigator	Opens the Navigator.
Alt-Shift-1	Debugging > Variables	Opens the Variables debugger window.
Alt-Shift-2	Debugging > Watches	Opens the Watches debugger window.
Alt-Shift-3	Debugging > Call Stack	Opens the Call Stack debugger window.
Alt-Shift-4	Debugging > Loaded Classes	Opens the Loaded Classes debugger window.
Alt-Shift-5	Debugging > Breakpoints	Opens the Breakpoints debugger window.
Alt-Shift-6	Debugging > Sessions	Opens the Sessions debugger window.
Alt-Shift-7	Debugging > Threads	Opens the Threads debugger window.
Alt-Shift-8	Debugging > Sources	Opens the Sources debugger window.
Ctrl-W	Close Window	Closes the current tab in the current window. If the window has no tabs, the whole window is closed.
Shift-Escape	Maximize Window	Maximizes the Source Editor or the present window.
Alt-Shift-D	Configure Window > Dock	Pins a detached window to the IDE.
Ctrl-Shift-W	Close All Documents	Closes all open documents in the Source Editor.
Shift-F4	Documents	Opens the Documents dialog box, in which you can save and close groups of open documents.
Ctrl-Tab (Ctrl-`)	Switch to Recent Window	Toggles through the open windows in the order that they were last used. The dialog box displays all open windows and each of the open documents in the Source Editor.

**Table 2–13 Help Menu**

<b>Keys</b>	<b>Command</b>	<b>Action</b>
F1	Help	Displays the help topic for the current UI element in the JavaHelp viewer.

**Table 2–13 (Cont.) Help Menu**

Keys	Command	Action
Shift-F1	Javadoc Index Search	Lets you search the Javadoc index.

## 2.5.6 How to Customize Keyboard Shortcuts

NetBeans IDE supports customizable shortcuts to improve your productivity but and increase your overall typing speed.

### To customize keyboard shortcuts:

1. Choose **Tools > Options > Keymap**.
2. Do either:
  - Select a predefined set of keyboard shortcuts, which is called Profile.
  - Edit particular keyboard shortcuts.

You can save customized sets of your shortcuts as profiles. Then, you can switch from one profile to another to quickly change multiple settings.

### To create a custom profile of keyboard shortcuts:

1. In the **Options > Keymap** window, click **Manage profiles**.
  2. Select the profile you want to use as a base for your new profile and click **Duplicate**.
  3. Enter the new profile name and click **OK**.
  4. Ensure that the new profile is selected and modify the shortcuts you need.
- To edit a shortcut, double-click in the **Shortcut** field or click the ellipsis button (...). As you press the sequence of keys, the syntax for them is added.

If you want to add special characters, such as Tab, Escape, or Enter, click the ellipsis button (...) again and select the key from the pop-up window.

5. When finished editing, click **OK** in the **Options** window.

To find a shortcut for a specific command, type the command name in the **Search** field. To find a command by a combination, insert the cursor in the **Search in Shortcuts** field and press the shortcut key combination.

## 2.6 Understanding the Source Editor Features

The Source Editor is a full-featured text editor that is integrated with the GUI Builder, the compiler, the debugger, and other parts of the IDE. You can generally open the Source Editor by double-clicking a node in the Projects window, Files window, or Navigator window.

The top of the Source Editor has a tab for each open document. Each tab shows the name of the document. The name of the document is displayed in bold if the document has been modified and has not been saved. You can right-click the tab to open a popup menu that lists commands that can be invoked on the tab.

If multiple files are open, you can split the Source Editor view by clicking and dragging the tab. As you drag to different parts of the editing area, you see a red outline that shows you the location of the new window. When one of these boxes

appears, you can drop the document and split the pane. You must close the split file if you want the editor to return to its unsplit state.

### 2.6.1 How to Use the Toolbar

The editor toolbar is at the top of the Source Editor window. The toolbar has buttons for various navigating and editing shortcuts, which vary according to the type of file you are editing. Hold the cursor over a button to display a description of the command.

#### How to toggle the editor toolbar:

- Choose **View > Show Editor Toolbar** to hide or display the editor toolbar.

The following table provides descriptions of icons that are used in the source editor toolbar.

**Table 2–14 Icons in Source Editor Toolbar**

Icon	Description
Source/History	Toggle between source code and local history views. Local history includes diffs between all save points.
 Last Edit (Ctrl+Q)	Moves to the last edit you made.
 Back (Alt+LEFT)	Moves back to previously opened document.
 Forward (Alt+RIGHT)	Moves forward to next opened document.
 Find Selection (Ctrl+F3)	Finds the item in which the cursor is currently inserted.
 Find Previous Occurrence (Shift+F3)	Moves the insertion point to the previous found occurrence of the text that you previously searched for.
 Find Next Occurrence (F3)	Moves the insertion point to the next found occurrence of the text that you previously searched for.
 Toggle Highlight Search (Alt+Shift+H)	Turns off or turns on the highlighting of search text.
 Toggle Rectangular Selection (Ctrl+Shift+R)	Turns off or turns on the rectangular block selection.
 Previous Bookmark	Moves the insertion point to the previous bookmark in the file.
 Next Bookmark	Moves the insertion point to the next bookmark in the file.
 Toggle Bookmark	Inserts a bookmark on the current line or removes a bookmark on the current line.
 Shift Line Left (Alt+Shift+LEFT)	Reduces the indentation by one tab stop.
 Shift Line Right (Alt+Shift+RIGHT)	Increases the indentation by one tab stop.
 Start Macro Recording	Starts recording a macro containing keystrokes and cursor movements.

**Table 2-14 (Cont.) Icons in Source Editor Toolbar**

Icon	Description
	Stops macro recording.
	Check XML document is well-formed.
	Validate XML document against DTD or schema specified in document.
	Run XSL transformation on open XSLT document.
	Completes the word you are typing to match the next word in the file that matches the word you are typing.
	Completes the word you are typing to match the closest previous word in the file that matches the word you are typing.
	Comments out the selected lines
	Removes the comment marks from the selected lines

## 2.6.2 How to Use the Left Margin

The left margin displays annotation glyphs that indicate line status, such as breakpoints, the program counter, or build errors. You can right-click the left margin to display a pop-up menu with a list of commands and options.

If you click the left margin of a Java, JavaScript or PHP file, a breakpoint is set on the corresponding line. You can click the left margin of a line to remove a breakpoint.

For more information about setting breakpoints and other glyphs used by the debugger, see [Chapter 10.9.4, "Managing Breakpoints."](#)

The following table shows some of the annotation glyphs that can appear in the left margin of the Source Editor.

**Table 2-15 Table of Common Editor Glyphs and Descriptions**

Glyph	Description
	The line contains an error. You can place the mouse pointer over the glyph to display a tooltip with information on the error.
	The line contains a method that overrides a method from another class. Place your mouse pointer over the glyph to display a tooltip with name of the overridden method's class. This glyph only applies to Java classes.
	The line contains a method that implements a method from an interface or an abstract class. Place your mouse pointer over the glyph to display a tooltip with name of the implemented method's class. This glyph only applies to Java classes.
	The line is bookmarked. You can cycle forward and backward through your Source Editor bookmarks by pressing Ctrl-Shift-Period/Comma. You can add or remove a bookmark by pressing Ctrl-Shift-M.

**Table 2–15 (Cont.) Table of Common Editor Glyphs and Descriptions**

Glyph	Description
	The line contains a debugging line breakpoint. You can remove the breakpoint by clicking the glyph. You can set another breakpoint by clicking in the left margin next to the line where you want the breakpoint to appear.
	The line contains multiple annotation glyphs. Click the glyph to cycle through the annotations.

### 2.6.3 How to Use the Error Stripe

The error stripe is the strip to the right of the right scroll bar and contains marks for various things in your file, such as errors, bookmarks, and comments for the To Do list. The error stripe represents the whole file, not just the lines currently displayed. You can immediately identify whether your file has any errors without having to scroll through the entire file. You can double-click a mark in the error stripe to jump to the line that the mark refers to.

### 2.6.4 How to Use the Status Line

The Source Editor status line is beneath the horizontal scroll bar. The first area of the status line shows the current line number and row number in the form line:row. The second area of the status line indicates the insertion mode (INS or OVR). The text area on the right is used for status messages.

### 2.6.5 How to Use Syntax Coloring and Highlighting

Source code displayed in the Source Editor is syntactically colored. For example, all Java keywords are shown in blue and all Java comments in light gray. Guarded blocks of Java code generated by the GUI Builder have a light blue background and cannot be edited.

**To customize semantic coloring settings for the Java Editor:**

- Choose Tools > Options > Fonts & Colors.

The IDE provides several preset coloring schemes, which are called profiles. You can create new profiles with custom colors and quickly switch between them.

**To save custom colors in a new profile:**

1. In the Options > Fonts & Colors window, click Duplicate next to the Profile drop-down list.
2. Enter the new profile name and click **OK**.
3. Ensure that the new profile is currently selected and choose Java from the Language drop-down list.
4. Select a category and change the font, font color (Foreground), background color, and effects for this category.

Use the **Preview** window to view the results.

5. Click **OK**.

---

**Note:** All NetBeans IDE settings and profiles are stored in the NetBeans userdir (refer to the FAQ on how to locate the userdir for your operating system). When upgrading to newer versions of NetBeans, you can export old settings and import them to the newer version.

---

In addition to coloring, the Source Editor highlights similar elements with a particular background color. Thus, you can think of the highlighting feature as an alternative to the Search command, because in combination with error stripes, it gives you a quick overview of where the highlighted places are located within a file.

The IDE highlights usages of the same element, matching braces, method exit points, and exception throwing points.

If you place the cursor in an element, such as a field or a variable, all usages of this element are highlighted. Note that error stripes in the Editor's right margin indicate the usages of this element in the entire source file. Click the error stripe to quickly navigate to the desired usage location.

**To rename all the highlighted instances:**

- Choose Refactor > Rename (or Ctrl-R).

## 2.6.6 How to Use Insert Mode and Overwrite Mode

When the Source Editor is in insert mode, the default insertion point is a vertical bar, and text that you type is inserted. In overwrite mode, the default insertion point is a solid block, and text that you type replaces the existing text.

Use the Insert key to toggle between the two modes.

Whenever the insertion point in the Source Editor is located immediately after a brace, bracket, or parenthesis, the matching brace, bracket, or parenthesis is highlighted.

If the Source Editor beeps when you try to enter new text, the file is a read-only file.

## 2.6.7 How to Set Editor Options

You can set options for the source editor in the Options window. The Editor category in the Options window contains tabs where you can set options for editor features, including code completion, formatting and folding.

You can set options for the colors of text and background colors in the Fonts and Colors category. The IDE includes several fonts and colors settings that are saved as profiles. You can choose to use one of the default settings profiles or create a custom profile that saves your font and color preferences.

**How to set global editor options:**

1. Choose Tools > Options from the main menu.
2. Click the **Editor** category and then click a tab to edit the settings for that feature.

**How to specify global fonts and colors in the editor:**

1. Choose Tools > Options from the main menu.
2. Click the **Fonts & Colors** category and then click a tab to edit the settings for that feature.

3. Select an item in the Category or Highlighting pane and then modify the color or other properties for that element in the right pane of the Options window.

**How to enable a default profile:**

1. Choose **Tools > Options** from the main menu.
2. Click the **FONT & Colors** category.
3. Select a profile from the Profiles drop-down list.

When you select a profile you can see an example of the new settings in the Preview pane.

**How to create a custom profile for fonts and colors**

1. Choose **Tools > Options** from the main menu.
  2. Click the **FONT & Colors** category.
  3. Select a profile from the Profile drop-down list to use as a base for the custom profile.
  4. Click **Duplicate** and type a name for the new profile.
  5. Modify the font and color options as desired.
- When you modify the new profile you can see an example of the new settings in the Preview pane.
6. Click **Apply** in the Options window.

**How to set formatting options for a project:**

1. Right-click the project node in the Projects window and choose **Properties**.
2. Select the **Formatting** category in the Project Properties window.
3. Select **Use project specific options** in the Formatting panel and specify the options.

## 2.6.8 How to Insert and Highlight Occurrences in the Editor

By default, the IDE automatically inserts matching pairs of braces, brackets, and quotes. When you type an opening curly brace and then press Enter, the closing brace is added automatically. For {, [, ", and ', the editor inserts a matching pair.

**To enable inserting matching pairs:**

1. Choose **Tools > Options > Editor > Code Completion**.
2. Select the **Insert Closing Brackets Automatically** checkbox.

By default the editor highlights the occurrences of various code symbols and keywords when the insert cursor is located in one of the occurrences. You can disable the highlighting completely or limit the highlighting to specific elements.

**How to select types of occurrences to highlight in the editor:**

1. Choose **Tools > Options** from the main menu, and then click the **Highlighting** tab in the **Editor** category.
2. Select a language in the drop-down list.
3. Deselect the types of elements that you do not want to be highlighted.

You can disable the highlighting of all types of elements for the specified language by deselecting **Mark Occurrences Of Symbol Under Caret**.

## 2.6.9 How to Record Macros

You can use macros in the source editor that can generate common source code snippets or perform common editor tasks. After you record a macro you can assign a keyboard shortcut to run the macro on the current file.

### How to record a macro:

1. Click the **Start Macro Recording** icon in the toolbar of the source editor.
2. Perform the tasks or type the code that you want to record.
3. Click the **Stop Macro Recording** icon in the toolbar.

When you click **Stop Macro Recording** the IDE opens the New Macro Dialog.

4. Type a name for the macro. Click **OK**.
5. Select the new macro in the Editor Macros dialog box and click Set Shortcut.
6. Type a shortcut in the Add Shortcut dialog box. Click **OK**.
7. Click **OK** in the **Editor Macros** dialog box.

The new macro will not be saved if you click **Cancel** in the Editor Macros dialog box.

To view or modify your saved macros, choose **Tools > Options** in the main menu and click the **Macros** tab in the **Editor** category.

A special macro syntax is used to define these macros. For example, if you want to clear the current line in the editor from the cursor, your macro definition would be as follows: `selection-end-line remove-selection`.

Then you can assign "Ctrl+L" as the keyboard shortcut for this macro. Whenever you press that key combination, the whole line, from the position of the cursor, will be deleted.

### 2.6.9.1 Macro Keywords for NetBeans Java Editor

To get a complete list of all macro keywords, you create a NetBeans API action, which will get EditorKit from the JEditorPane in an opened editor, call `EK.getActions()`, and dump `Action.NAME` property of each action to `System.out` (together with `Action.SHORT_DESCRIPTION`).

A short list of macro keywords is as follows:

Macro	Description
abbrev-debug-line	Debug Filename and Line Number
adjust-caret-center	Move Insertion Point to Center
adjust-window-center	Scroll Insertion Point to Center
completion-show	Show Code Completion Popup
fix-imports	Fix Imports
make-getter	Replace Variable With its Getter
remove-line	Delete Line

Macro	Description
select-next-parameter	Select Next Parameter
toggle-toolbar	Toggle Toolbar
word-match-next	Next Matching Word

For a complete list of macro keywords, see the following document.

<http://wiki.netbeans.org/FaqEditorMacros>

## 2.6.10 How to Modify Source Editor Code Templates

Code Templates are abbreviations that you type into the code and that the editor expands into code blocks. The templates can include placeholder values. After you expand a code template, you go through the expanded block and replace the placeholder values.

The IDE includes default code templates for various languages. You can use the Options window to create, remove and modify code templates.

### To modify an existing code template:

1. Choose **Tools > Options** from the main menu, and then click **Code Templates** tab in the **Editor** category.
2. Select a language from the drop-down list to see the code templates available for that language.
3. Select a code template from the list.
4. Type in the **Expanded Text** tab to modify the expanded code that the template produces.
5. Type in the **Description** tab to modify the description of the code template. Click **OK** in the **Options** window.

### To remove an existing code template:

1. Choose **Tools > Options** from the main menu, and then click the **Editor** category.
2. Select a language from the drop-down list to see the code templates available for that language.
3. Select a code template from the list.
4. Click **Remove**.

### To create a code template:

1. Choose **Tools > Options** from the main menu, and then click the **Editor** category.
2. Select a language from the drop-down list.
3. Click **New**.
4. Type an abbreviation for the new code template in the dialog box.
5. Type the expanded code in the **Expanded Text** tab.
6. Type a description for the code in the **Description** tab.
7. Click **OK** in the **Options** window.

## 2.6.11 How to Use Code Completion

The NetBeans IDE's Source Editor helps you quickly complete and generate code through the "smart" code completion feature. In a general sense, code completion is very useful when you want to fill in the missing code, look at the options available in the context of your application, and generate blocks of code when needed.

Use code completion to:

- generate code from the code completion window
- complete keywords in your code. The editor analyzes the context and suggests the most relevant keywords
- choose a name that matches the type of a new field or a variable that you are adding
- complete Javadoc tags after you type the "@" symbol
- suggest parameters for variables, methods, or fields
- quickly fill in the most commonly used prefixes and single suggestions

### To invoke code completion:

- Press Ctrl-Space (or choose **Source > Complete Code** from the main menu) to open the code completion box.

While you are typing, the list of suggestions shortens. The suggestions listed include those imported in your source file and symbols from the `java.lang` package.

The suggestions that are the most relevant for the context of your code are displayed at the top, above the black line in the code completion window.

If the "smart" suggestions are not the ones you want to use, press Ctrl-Space again to see a complete list.

### To invoke tooltips with method parameters:

- Press Ctrl-P (or choose **Source > Show Method Parameters** from the main menu).

The editor guesses on the parameters for variables, methods, or fields and displays the suggestions in a pop-up box.

For example, when you select a method from the code completion window which has one or more arguments, the Editor highlights the first argument and displays a tooltip suggesting the format for this argument. To move to the next argument, press the Tab or Enter keys.

### To fill in the most commonly used prefixes and single suggestions:

1. Type a name or part of a name (for example, `System.out.p`) and wait for code completion to show all fields and methods that start with "p." All the suggestions will be related to "print."
2. Press the Tab key and the editor automatically fills in the "print".

You can continue and type "l" and, after pressing Tab, the "println" will be added.

### To customize the code completion settings:

- Select **Tools > Options > Editor > Code Completion**.

**To invoke the code completion window automatically when you are typing certain characters:**

1. Select Tools > Options > Editor > Code Completion.
2. On the Code Completion tab, select the Auto Popup Completion Window checkbox.

The default character is ".", but you can add your own characters.

**To add characters that invoke the code completion window:**

1. Select Java from the Language drop-down list.
2. Type your characters in the Auto Popup Triggers for Java field.

The code completion window will pop up every time you type the specified characters.

---

**Note:** When the Auto Popup Completion Window checkbox is disabled, you need to press Ctrl-Space each time you want to use code completion.

---

In the code completion window, the following icons are used to distinguish different members of the Java language.

**Table 2–16 Icons in the Code Completion Window**

Icon	Meaning	Variants (if any)	Meaning
	Annotation type		
	Class		
	Package		
	Enum type		
	Code Template		
	Constructor	   	New constructor (generate) Protected constructor Private constructor Package private constructor
	Field	  	Protected field Private field Package private field
	Static field	  	Protected static field Private static field Package private static field
	Interface		

**Table 2–16 (Cont.) Icons in the Code Completion Window**

Icon	Meaning	Variants (if any)	Meaning
	Java keyword		
	Method		Protected method
			Private method
			Package private method
	Static method		Protected static method
			Private static method
			Package private static method
	Local variable		
	Attribute		

## 2.6.12 How to Use Hints

While you are typing, the Source Editor checks your code and provides suggestions of how you can fix errors and navigate through code.

For the most common coding mistakes, you can see hints in the left-hand margin of the Editor. The hints are shown for many types of errors, such as missing field and variable definitions, problems with imports, braces, and other. Click the hint icon and select the fix to add.

Hints are displayed automatically by default.

### To view all hints:

- Choose Source > Fix Code (or press Alt-Enter).

### To limit the number of categories for which hints are displayed:

1. Choose Tools > Options > Editor > Hints.
2. From the **Language** drop-down list, select the language and view a list of elements for which hints are displayed (their checkboxes are selected).
3. To disable hints for some categories, clear the appropriate checkboxes.

---

**Note:** On the **Hints** tab, you can also disable or limit the scope of dependency scans (**Dependency Scanning** option). These steps can significantly improve the performance of the IDE.

The IDE detects compilation errors in your sources by locating and recompiling classes that depend on the file that you are modifying (even if these dependencies are in the files that are not opened in the editor). When a compilation error is found, red badges are added to source file, package, or project nodes. Dependency scanning within projects can be resource consuming and degrade performance, especially if you are working with large projects.

To improve IDE's performance, you can do one of the following:

- Limit the scope of dependency scans to the Source Root (search for dependencies only in the source root where the modified class is located) or current Project.
  - Disable dependency scanning (choose **Project Properties > Build > Compiling** and deselect the **Track Java Dependencies** option). In this case, the IDE does not scan for dependencies or updates the error badges when you modify a file.
- 

#### **To surround pieces of your code with various statements:**

- Select a block in your code that you want to surround with a statement and click the bulb icon in the left-hand margin (or press Alt-Enter). The editor displays a list of suggestions from which you select the statement you need.

You can surround pieces of your code with various statements, such as `for`, `while`, `if`, `try/catch`, and other.

### **2.6.13 How to Navigate Through Code**

Use the **Go To..** commands located under the **Navigate** menu item to quickly jump to target locations.

#### **To go to declaration:**

1. Hold down the Ctrl key and click the usage of a class, method, or field to jump to its declaration. You can also place the cursor on the member (a class, method, or field).
2. Choose **Navigate > Go To Declaration**.

#### **To go to source:**

1. Hold down the Ctrl key and click a class, method, or field to jump to the source code, if the source is available. You can also place the cursor on the member (a class, method, or field).
2. Choose **Navigate > Go To Source** in the main menu.

#### **To go to type, file, or symbol:**

- If you know the name of the type (class, interface, annotation or enum), file, or symbol to where you want to jump, choose **Navigate > Go to Type (Go to File, or Go to Symbol)** and type the name in the new window.

---

**Note:** You can use prefixes, camel case, and wildcards.

---

**To go to line:**

1. Choose **Navigate > Go To Source** in the main menu.
2. Enter the line number to which you want to jump.

**To quickly return to your last edit:**

- Even if your last edit is in another file or project, press Ctrl-Q or use the button in the top left corner of the Source Editor toolbar.

The last edited document opens, and the cursor is at the position, which you edited last.

**To switch between open files:**

- To go to the previously edited file or move forward, choose **Navigate > Back** or **Navigate > Forward**, or press the corresponding buttons on the editor toolbar.  
The file opens and the cursor is placed at the location of your last edit. When you click one of the buttons, you can expand the list of the recent files and click to navigate to any of them.
- To toggle between files, press Ctrl-Tab to display all open files in a pop-up window, hold down the Ctrl key and press several times the Tab key to choose the file you would like to open.

**To go to bookmarks:**

- Choose **Navigate > Bookmark History Popup Next** in the main menu, to go to your next bookmark.
- Choose **Navigate > Bookmark History Popup Previous** in the main menu, to go to the previous bookmark.

### 2.6.14 How to Work with Import Statements

There are several ways of how you can manage import statements in NetBeans IDE. The IDE's Java Editor constantly checks your code for the correct use of import statements and immediately warns you when non-imported classes or unused import statements are detected.

When a non-imported class is found, the  error mark appears in the IDE's left-hand margin (this margin is also called the *glyph margin*). Click the error mark and choose whether to add the missing import or create this class in the current package.

---

**Note:** When you select a class from the code completion window, the Editor automatically adds an import statement for it.

---

**To add all missing import statements at once:**

- Choose **Source > Fix Imports** from the menu (or press Ctrl-Shift-I) while you are typing.

---

**Note:** To quickly see if your code contains unused or missing imports, watch the error stripes in the right-hand margin: orange stripes mark missing or unused imports.

---

**To add an import only for the type at which the cursor is located:**

- Press Alt-Shift-I.

**To remove one unused import or all unused imports:**

- If there are unused import statements in your code, press the  warning mark in the Editor left-hand margin and choose either to remove one unused import or all unused imports.

---

**Note:** In the Editor, unused imports are underlined.

---

## 2.6.15 How to Generate Code

When working in the Source Editor, you can automatically generate pieces of code in one of the two ways: by using code completion or from the **Code Generation** dialog box.

**To insert code from the Code Generation dialog box:**

- Choose Source > Insert Code (or Press Alt-Insert anywhere in the Editor).

The suggested list is adjusted to the current context.

In the IDE's Source Editor, you can automatically generate various constructs and whole methods, override and delegate methods, add properties and more.

**To generate code from the Code Completion window:**

- Press Ctrl-Space to open the code completion window and choose the most appropriate item.

The Editor generates a piece of code appropriate for the current context.

---

**Note:** In the code completion window, the constructors that can be automatically generated are marked with the  icon and the *generate note*.

---

## 2.6.16 Using General Editor Shortcuts

The IDE includes keyboard shortcuts that you can use to activate many Source Editor operations.

In the following table, multikey shortcuts are written in the following format: Ctrl-U, T. To use this shortcut, hold down the Ctrl key and press U, then release both keys and press T.

**Table 2-17 Scrolling and Selecting**

Keys	Action
Ctrl-down arrow	Scrolls the window up without moving the insertion point.
Ctrl-up arrow	Scrolls the window down without moving the insertion point.

**Table 2–17 (Cont.) Scrolling and Selecting**

Keys	Action
Ctrl-[	Moves the insertion point to the highlighted matching bracket. This shortcut only works when the insertion point is immediately after the opening or closing bracket.
Ctrl-Shift-[	Selects the block between a pair of brackets. This shortcut only works when the insertion point is immediately after either the opening or closing bracket.
Ctrl-G	Jumps to any specified line.
Ctrl-A	Selects all text in the file.

**Table 2–18 Modifying Text**

Keys	Action
INSERT	Switches between insert text and overwrite text mode.
Ctrl-Shift-J	Opens the Internationalize dialog box that you can use to insert an internationalized string at the insertion point.
Ctrl-U, U	Makes the selected characters or the character to the right of the insertion point uppercase.
Ctrl-U, L	Makes the selected characters or the character to the right of the insertion point lowercase.
Ctrl-U, S	Reverses the case of the selected characters or the character to the right of the insertion point.

**Table 2–19 Code Folding**

Keys	Action
Ctrl-Minus (-)	Collapses the block of code the insertion point is on.
Ctrl-Plus (+)	Expands the block of code the insertion point is next to.
Ctrl-Shift-Minus (-)	Collapses all blocks of code.
Ctrl-Shift-Plus (+)	Expands all blocks of code.

**Table 2–20 Cutting, Copying, Pasting, and Deleting Text**

Keys	Action
Ctrl-Z	(Undo) Reverses (one at a time) a series of editor actions, except Save.
Ctrl-Y	(Redo) Reverses (one at a time) a series of Undo commands.
Ctrl-X	(Cut) Deletes the current selection and places it into the clipboard.
Shift-Delete	(Cut) Deletes the current selection and places it into the clipboard.
Ctrl-C	(Copy) Copies the current selection to the clipboard.
Ctrl-Insert	(Copy) Copies the current selection to the clipboard.
Ctrl-V	(Paste) Pastes the contents of the clipboard at the insert point.
Delete	(Delete) Deletes the current selection.
Ctrl-E	Deletes the current line.

**Table 2–20 (Cont.) Cutting, Copying, Pasting, and Deleting Text**

Keys	Action
Ctrl-U	Deletes text in the following sequence: <ul style="list-style-type: none"> <li>■ text preceding insertion point on same line</li> <li>■ indentation on same line</li> <li>■ line break</li> <li>■ text on previous line</li> </ul>
Ctrl-Backspace	Removes the text in the current word preceding the insertion point.

**Table 2–21 Searching for Text**

Keys	Action
Ctrl-F3	Searches for the word the insertion point is on and highlights all occurrences of that word.
F3	Selects the next occurrence of the word in your current search.
Shift-F3	Selects the previous occurrence of the word in your current search.
Alt-Shift-H	Switches highlighting of search results on or off.
Ctrl-F	Opens the Find dialog box.
Ctrl-H	Opens the Find and Replace dialog box.

**Table 2–22 Setting Tabs**

Keys	Action
Tab	Shifts all text to right of insertion point to the right.
Alt-Shift-Right	Shifts text in line containing the insertion point to the right.
Alt-Shift-Left	Shifts text in line containing the insertion point to the left.

**Table 2–23 Using Bookmarks**

Keys	Action
Ctrl-Shift-M	Sets or unsets a bookmark at current line.
Ctrl-Shift-Period/Comma	Goes to next/previous bookmark.

**Table 2–24 Using Multiple Carets**

Keys (Windows/Linux)	Keys (Mac OS)	Keys (Eclipse/Idea)	Description
Ctrl-Shift-Click	Cmd-Shift-Click	Ctrl-Shift-Click	Add or remove a caret.
Ctrl-J	Cmd-J	Alt-J	Add a caret for the next occurrence.
Ctrl-Alt-Shift-J	Ctrl-Cmd-Shift-J	Ctrl-Alt-Shift-J	Add a caret for all occurrences.
Alt-Shift-[	Ctrl-Shift-[ or Alt-Cmd-Up	Alt-Shift-[	Add a caret in the previous line.

**Table 2–24 (Cont.) Using Multiple Carets**

<b>Keys (Windows/Linux)</b>	<b>Keys (Mac OS)</b>	<b>Keys (Eclipse/Idea)</b>	<b>Description</b>
Alt-Shift-]	Ctrl-Shift-] or Alt-Cmd-Down	Alt-Shift-]	Add a caret in the next line.
Alt-Shift-J	Ctrl-Shift-J	Alt-Shift-J	Remove the last added caret.
Ctrl-Shift-L	Cmd-Shift-L or Alt-Cmd-V	Ctrl-Shift-L	Paste clipboard content as lines over the multiple carets.
Escape	Escape	Escape	Remove all extra carets and return to the regular mode.

## 2.7 Setting Startup Parameters

You can pass startup parameters to the IDE launcher using the *IDE-HOME/etc/netbeans.conf* file or on the command line. The launcher reads the *netbeans.conf* file prior to parsing the command-line options, including any parameters you have added. The *netbeans.conf* file contains details on some of the available startup switches, including switches to modify the heap size.

**Table 2–25 Description of Common Switches**

<b>Switch</b>	<b>Description</b>
--help (or -h)	Prints descriptions of common startup parameters.
--jdkhome <i>jdk-home-dir</i>	Uses the specified version of the JDK instead of the default JDK. By default on Microsoft Windows systems, the IDE's launcher looks into the registry and uses the latest JDK available.
	You should back up your user directory before you upgrade the JDK that the IDE uses. If you later need to revert to the previous JDK, switch to the backed up user directory to ensure that you do not lose any settings.
	To switch the IDE's user directory, use the --userdir switch that is detailed below.
--cp:p <i>additional-classpath</i>	Prefixes the specified class path to the IDE's class path.
--cp:a <i>additional-classpath</i>	Appends the specified class path to the IDE's class path.
--open <i>file</i>	Opens the file in the Source Editor.
--open <i>file:line number</i>	Opens the file in the Source Editor at the specified line.
--laf <i>UI-class-name</i>	Selects the given class as the IDE's look and feel. The following are two examples of look and feel classes. <ul style="list-style-type: none"> <li>▪ com.sun.java.swing.plaf.motif.MotifLookAndFeel</li> <li>▪ javax.swing.plaf.metal.MetalLookAndFeel</li> </ul>
--fontsize <i>size</i>	Sets the font size, expressed in points, in the IDE's user interface. If this option is not used, the font size is 11 points.
--locale <i>language[:country[:variant]]</i>	Activates the specified locale.

**Table 2–25 (Cont.) Description of Common Switches**

<b>Switch</b>	<b>Description</b>
--userdir <i>userdir</i>	Explicitly specifies the user directory, which is the location where user settings are stored. If this option is not used in UNIX environments, the user directory is set in the <i>HOME</i> directory by default. If this option is not set on Microsoft Windows systems, the user directory is the one you specified when you first launched the IDE. You can determine the current user directory in the About dialog box.
--cachedirpath	Specifies the directory to store the user cache.
-J <i>jvm-flags</i>	Passes the specified flags directly to the JVM software.
-J-Dsun.java2d.noddraw=true	Prevents the use of DirectX for rendering. This switch might prevent problems that occur on some Microsoft Windows systems with faulty graphics cards.

---

**Note:** When adding startup switches, you can break options into multiple lines.

---

For information about your userdir and cachedir, see the following document.

<http://wiki.netbeans.org/FaqWhatIsUserdir>

For information about performance-related startup switches, see the following document.

<http://performance.netbeans.org/howto/jvmswitches/>

For information about modifying the JVM heap size, see the following FAQ on the NetBeans wiki.

<http://wiki.netbeans.org/FaqSettingHeapSize>

## 2.8 Setting Fonts and Colors for IDE Windows

You can customize the font and color properties that are used in various windows in the IDE. Font properties include the font family, style, variant, weight, and size. Color properties include the color of text, the background color of text, and the color of borders.

### 2.8.1 How to Set Fonts and Colors for the Help Viewer

You can customize the font and color properties that are used to display the pages in the help viewer. Font properties include the font family, style, variant, weight, and size. Color properties include the color of text, the background color of text, and the color of borders.

#### To set fonts and colors for displaying help pages:

1. From the main window, choose File > Open File.
2. In the Open dialog box, navigate to *IDE-install-directory\ide\docs\org\.netbeans\modules\usersguide* and open the file *ide.css*.

The `ide.css` file is the style sheet that describes how to present the fonts and colors in the help pages.

3. Edit `ide.css` to your preferences.
4. Choose File > Save.
5. Open the help viewer and verify your changes.

### 2.8.2 How to Set Fonts and Colors for the Output Window

You can customize the font and color properties that are used to display the pages in the help viewer. Font properties include the font family, style, variant, weight, and size. Color properties include the color of text, the background color of text, and the color of borders.

**To set fonts and colors for the Output window:**

1. Choose Tools > Options from the main menu.
2. Click the **Output** tab in the **Miscellaneous** category.

### 2.8.3 How to Set Fonts and Colors for the Terminal Window

You can customize the font and color properties that are used to display the pages in the help viewer. Font properties include the font family, style, variant, weight, and size. Color properties include the color of text, the background color of text, and the color of borders.

**To set fonts and colors for the Terminal:**

1. Choose Tools > Options from the main menu.
2. Click the **Terminal** tab in the **Miscellaneous** category.

## 2.9 Managing Plugins in the IDE

The IDE's Plugins manager enables you to update and manage your IDE's plugins dynamically. You can minimize startup time and save memory by deactivating the plugins that you do not need. Deactivated plugins are not deleted from your installation directory, but are simply ignored by the IDE. You can re-activate them at any time.

You can also choose to uninstall plugins from the IDE. Uninstalled plugins are removed from your installation directory. To use an uninstalled plugin, you need to install it again.

You use the Plugins manager to connect to the update centers to check if there are new plugins or new versions of already installed plugins available. If new or updated plugins are available, you can select, download, and install them using the Plugins manager.

A plugin generally consists of a group of dependent modules. Some IDE plugins are grouped together and referred to as a Feature. Each Feature generally corresponds to a technology and can contain a number of plugins that support that technology. When you activate or deactivate a Feature, the IDE activates or deactivates the corresponding plugins.

When you update a plugin, the update usually consists of updating one or more of the modules that make up the plugin. Some plugins may be dependent on modules in other plugins in order for the functionality to be implemented. The Plugins manager

warns you when this is the case. Deactivating a plugin usually consists of deactivating the individual modules that make up the plugin.

## 2.9.1 How to Update the IDE from the Update Center

When you use the Plugins manager to update the IDE, the IDE checks the registered update centers to see if there are new plugins or new versions of already installed plugins available. If new or updated plugins are available, you can select, download, and install the plugins using the Plugins manager. You can set the frequency that the IDE checks for updates in the Settings tab of the Plugins manager.

Alternatively, you can choose **Help > Check For Updates** from the main menu to open the Plugin Installer. The Plugin Installer will check for updates of installed in plugins. If updates are available, you can step through the installer to install the updates.

In addition to the default IDE Update Center, you can choose from several update centers that offer different types of plugins, such as experimental new plugins or old plugins that are no longer in regular distribution.

### To update installed plugins from the Update Center:

1. Choose **Tools > Plugins** from the main menu to open the Plugins manager.
2. Click the **Updates** tab to display available updates of installed plugins.
3. In the left pane, select the plugins you wish to update and click **Update**.
4. Complete the pages in the installer to download and install the update.

The left pane of the Updates tab displays the installed plugins which have updates available from the update centers. By default, the IDE regularly checks the registered update centers for available updates of installed plugins. If no plugins are displayed in the left pane, it means that no updates were available the last time the IDE checked the update center.

### To add new plugins from the Update Center:

1. Choose **Tools > Plugins** from the main menu to open the Plugins manager.
2. Click the **Available Plugins** tab to display plugins that are available but not installed.
3. In the left pane, select the plugins you wish to add and click **Install**.
4. Complete the pages in the installer to download and install the plugin.

---

**Note:** Some plugins may require you to restart the IDE to complete the update process or to activate or deactivate them.

---

## 2.9.2 How to Install Downloaded Plugins

If you have already downloaded a plugin's .nbm file, you can manually install it without having to connect to an update center.

### To install a downloaded plugin:

1. Choose **Tools > Plugins** from the main menu and select the **Downloaded** tab.
2. Click **Add Plugins** and browse to the location of the downloaded .nbm file. Select the file and click **Open**.
3. Repeat the previous step for each plugin that you wish to add.

4. Click **Install** and complete the pages of the installer to install the plugin.

If the plugin that you have selected in the file chooser does not appear in the Downloaded tab of the Plugins manager, you probably have the same or a newer version of that plugin already installed. You can check the Installed tab to verify that the plugin is installed and activated.

### 2.9.3 How to Activate and Deactivate Plugins

The IDE enables you to deactivate installed plugins that you do not need to minimize startup time and save memory. Deactivated plugins are not deleted from your installation directory, but are simply ignored by the IDE. You can re-activate them at any time. You do not need to download the plugins again.

**To activate or deactivate an installed plugin:**

1. Choose **Tools > Plugins** from the main menu and then click the **Installed** tab in the Plugins manager.
2. In the left pane, select the plugin you wish to activate or deactivate.  
The icon in the Active column indicates the status of the plugin.
3. Click **Activate** or **Deactivate** to activate or deactivate the plugin.  
The icon for the plugin in the left pane reflects the new status of the plugin.
4. Click **Close** to exit the Plugins manager.

If you want to completely remove a plugin from your local system, select the checkbox for the plugin in the left pane and then click **Uninstall**.

The following table defines the icons used in the Installed tab of the Plugins manager to indicate the status of plugins.

**Table 2–26 Plugin Status Icons**

Icon	Description
	The plugin is installed and activated.
	The plugin will be fully deactivated after you restart the IDE.
	The plugin is installed but deactivated.

### 2.9.4 How to Globally Install Plugins

If you are using a multi-user installation of the IDE, you can install a plugin globally so that the plugin is available to all users. When you install or update a plugin using the Plugins manager, the IDE places the plugin JAR and docs in your user directory. The IDE places JAR and docs for globally installed plugins in your installation directory instead of an individual user directory.

**To globally install a plugin:**

1. Choose **Tools > Plugins** to open the Plugins manager.
2. Click the **Settings** tab and then select **Force install into shared directories** in the Plugin Install Location drop-down list.
3. Click the **Available Plugins** tab, select the **Install** checkbox for the plugin and click **Install**.

You can also install manually downloaded plugins in the Downloaded tab.

4. Follow the wizard instructions to complete the installation of the plugin.
5. Restart the IDE to activate the new plugin, if necessary.

To install a plugin globally, you must have write access for the IDE installation directory.

### 2.9.5 How to Add an Update Center

The IDE's Plugins manager enables you to update your IDE's installed plugins from registered update centers. You can specify the update centers you want the IDE to check and how often the IDE checks the update centers.

You can manage the registered update centers in the Settings tab of the Plugins manager. In addition to the default IDE Update center, you can add other update centers that offer different types of plugins, such as experimental new plugins or old plugins that are no longer in regular distribution.

**To add an update center:**

1. Choose **Tools > Plugins** from the main menu to open the Plugins manager.
2. Click the **Settings** tab to display the registered update centers.
3. Click **Add** to open the Update Center Customizer dialog box.
4. Type the name of the update center and the URL and click **OK**.

In the Settings tab of the Plugins manager you can also do the following:

- Deactivate an update center by deselecting the Active checkbox for the update center you wish to deactivate. You can reactivate the update center later if you wish.
- Edit the details of an update center by selecting the update center in the left pane and clicking **Edit** in the right pane.
- Remove an update center by selecting the update center in the left pane and clicking **Remove** in the right pane.

### 2.9.6 How to Schedule Update Checks

By default, the IDE periodically checks the update centers for new updates. You can set the interval or specify that the IDE should not automatically check for updates.

**To schedule update checks:**

1. In the Settings tab of the Plugins manager, choose a frequency from the Check Interval drop-down list. Choose Never if you do not want the IDE to automatically check for updates.
2. Click **Proxy Settings** if you need to set any proxy settings needed to enable the IDE to access the update centers.

## 2.10 Displaying IDE Help in a Web Browser

The online help is designed to be viewed in the JavaHelp viewer, which is integrated into the IDE. If you prefer, you can view the online help in the IDE's web browser or in another HTML browser.

### 2.10.1 How to Extract IDE Help from a JAR File

The online help is divided into several help sets. Most help sets are packaged in a JAR file, while a few help sets are packaged in a zip file. Help sets are typically stored in the ide/modules/docs directory in your installation directory.

**To extract the online help from a JAR file:**

1. In a command window, change to the *install-directory/ide/modules/docs* directory and list the files in that directory.
2. Use the jar command-line utility to unpack the JAR file in which you are interested. On Microsoft Windows systems and UNIX systems, the command is as follows:

```
jar xf jar-file
```

For *jar-file*, use the file name of the JAR file from which you want to extract files.

The Jar tool makes copies of the files and writes them to the current directory. The directory structure is reproduced according to the package structure in the archive.

### 2.10.2 How to View IDE Help in a Web Browser

**To view the online help in a web browser:**

1. From the IDE main window, choose **View > Web Browser**.
2. In the web browser use the File Browser and navigate to the files you just extracted.

## 2.11 Internationalizing Source Code

The IDE's internationalization tools let you easily insert internationalization strings while you write your code as well as internationalize files that are already written. You can also check resource bundles to make sure each key referenced by your code exists.

When you internationalize existing source files, the IDE searches for every customizable occurrence of a quoted text string. If you decide to internationalize the string, the IDE replaces it with a method call and adds the string to the appropriate resource bundle. You can select from several code formats for generating internationalized strings, or you can use your own custom format.

### 2.11.1 How to Enable Automatic Internationalization

As you design a form in the GUI Builder, you can have the code generated as internationalized code.

**To turn on automatic internationalization for a form:**

1. Open the form and make sure that the form appears in the Design view.
2. In the Navigator window, select the root node for the form.
3. In the Properties window, select the Automatic Internationalization checkbox.

## 2.11.2 How to Internationalize a Single File

The Internationalize dialog box enables you to replace hard-coded strings in a single file with internationalized strings. If you need to internationalize several Java sources into one or more resource bundles, use the Internationalization wizard.

### To internationalize source code:

1. In the Files or Projects window, right-click the class file you want to internationalize and choose **Tools > Internationalization > Internationalize**.

The Internationalize dialog box opens enabling you to edit each string in the file consecutively.

- If a resource bundle already exists for the file, the bundle properties file in which the strings are saved is displayed in the Bundle Name field. The Replace String field also displays a preview of the internationalized string.
- If no resource bundle exists for the file, the Bundle Name and Replace String fields are empty. You can click the ellipsis (...) button to specify a properties file or create a new one in which to save internationalized strings.

2. If you want to change the method used to generate the localized string, click Format.
3. If you want to add arguments to the method call, click Arguments. You can only add arguments to the method call if you use the `java.text.MessageFormat` or `org.openide.util.NbBundle.getMessage` formats.
4. Check that the key and value are correct. By default, the Internationalize dialog box gives the key the same name as the string being replaced.
5. Click Replace to generate the internationalized string.

The next string to be internationalized is then displayed in the Value text field.

6. Click Skip to ignore any strings you do not want to internationalize.

The IDE automatically dismisses the dialog box once the last string has been replaced.

To get information about the highlighted string in the Source Editor, click Info to see the name of the component containing the string and the property that the string is associated with.

## 2.11.3 How to Use the Internationalization Wizard

The Internationalization wizard enables you to replace hard-coded strings with internationalized strings in multiple files. If you are localizing the source into more than one language, the Internationalization wizard also lets you specify the localized string for multiple locales.

### To automatically internationalize strings in multiple source files:

1. Choose **Tools > Internationalization > Internationalization Wizard** from the main window.
2. Click **Add Source(s)** to add one or more source files to internationalize. Click **Next** to proceed.
3. Click **Select All** if you want a single resource bundle to contain the key and value pairs for all the listed sources. To select a resource bundle for specific source files, select the desired sources and click **Select Resource**. In the Select Resource dialog box, select the desired `.properties` file or create a new file. Click **Next** to proceed.

To modify key and value pairs:

1. From the Source drop-down list, select the source file in which you want to create the variable.
2. Select the Generate Field checkbox and set the variable's modifiers.
3. Type the name of the identifier in the Identifier text field.
4. The Init String field gives you a preview of the code that the variable will use to reference the resource bundle. To change this code, click Format and select the desired code format from the Init Code Format Editor.
5. Click Next to generate the field.
6. In the Modify Found Strings pane, set the key name and localized values for each of the strings you want to internationalize by entering a new value in the appropriate column. Use the Source drop-down list to switch between source files. Deselect the checkbox in the first column for any string you do not want to internationalize.
7. To change the code format used to generate the internationalized string or add arguments to the method call, click the ellipsis button (...) at the end of the string's row.
8. Click Finish to internationalize the strings for all of the selected source files.

#### 2.11.4 How to Insert an Internationalized String Into Source Code

The Insert Internationalized String command enables you to add internationalization strings one at a time as you create the source.

##### **To insert an internationalized string into your source code:**

1. In the Source Editor, put the insertion point at the location in the source file where you want to insert an internationalized string.
2. Right-click the desired location and choose **Tools > Internationalization > Insert Internationalized String** from the pop-up menu, or use the keyboard shortcut (Ctrl-Shift-J).

The Insert Internationalized String dialog box appears enabling you to edit each string in the file consecutively. To choose a different .properties file or create a new file, click the ellipsis (...) button.

3. Click Format if you want to change the method used to generate the localized string.
4. Click Arguments if you want to add arguments to the method call. You can only add arguments to the method call if you use the `java.text.MessageFormat` or `org.openide.util.NbBundle.getMessage` formats.
5. Type in values for the Key and Value properties and click OK.

#### 2.11.5 How to Internationalize a String With the GUI Builder

If you need to internationalize a GUI application, you can use the GUI Builder to replace a GUI component's hard-coded text with internationalized strings.

##### **To insert an internationalized string with the GUI Builder:**

1. Open the desired file in the GUI Builder by double-clicking its node in the Files or Projects window.

2. Select the appropriate GUI component in the Navigator window.
3. In the Properties window, select the property that you want to internationalize (for example, the `text` property of `jLabel`).
4. Select Resource Bundle from the drop-down list at the top of the dialog box.  
The property editor switches to resource bundle mode.  
If a resource bundle already exists for the source, the bundle properties file into which the internationalized strings will be saved is displayed in the Bundle Name field.
5. If no `.properties` file is listed in the Bundle Name field, click the ellipsis (...) button next to the field to open the Select Resource Bundle dialog box. In the dialog box, specify an existing `.properties` file or create a new file. Click OK to return to the property editor.
6. If you want to change the method used to generate the internationalized string, click the Format button.
7. If you want to add arguments to the method call, click Arguments. You can add arguments to the method call only if you use the `java.text.MessageFormat` or `org.openide.util.NbBundle.getMessage` formats.
8. Type in values for the Key and Value properties and click OK.

## 2.11.6 How to Test a Bundle for Internationalized Strings

You can check resource bundles for missing key and value pairs by using the Internationalization Test wizard. The wizard looks for all internationalized string keys (with a customizable meaning) in the source and checks for a corresponding key and value pair in the specified resource bundle. You can use the wizard to add any missing pairs.

### To test a bundle for internationalized strings:

1. Choose **Tools > Internationalization > Internationalization Test Wizard**.
2. Click **Add Source(s)** and select one or more source files to test in the Select Sources dialog that appears. Click **Next** to proceed.
3. Specify the resource bundles for the source files. Click **Select All** to check one resource bundle for all the listed sources. To select a specific resource bundle for one or more source files, select the desired sources and click **Select Resource**. Click **Next** to proceed.

The wizard lists all of the internationalized strings with missing key and value pairs in the selected file. The wizard automatically suggests key names and values for the string.

4. Use the Source drop-down list to switch between source files. Deselect the checkbox in the first column for any string you do not want to include in the resource bundle. Change the key name and value for any string by typing a new value in the appropriate column.
5. Click **Finish** to add the missing key and value pairs to the resource bundle.

## 2.11.7 How to Add Arguments for Message Formats

The `java.text.MessageFormat` code format lets you use strings that are constructed dynamically at runtime. The dynamically added elements are stored in an array of

objects, which is passed as a parameter to `java.text.MessageFormat`. You can use the Message Format Arguments dialog box to specify the values of these objects.

You can add arguments when you use the `org.openide.util.NbBundle.getMessage` format. This format is part of the NetBeans APIs and is used to build modules for the IDE. Consult the NetBeans API documentation for more information on this format.

**To enter substitution parameters for a message format:**

1. In the Internationalize dialog box, click the **Format** button.
2. Select the `java.text.MessageFormat` format or the `org.openide.util.NbBundle.getMessage` format from the Replace Code Format drop-down list. Then click OK.
3. In the Internationalize dialog box, click the **Arguments** button.
4. Click the **Add** button next to the Arguments text field.  
A series of parameters, beginning with 0, is added to the text field.
5. Type a value for the parameter in the Code field.
6. Use the Add and Remove buttons to add or remove message parameters. When you are done, click OK to close the dialog box.
7. Finish entering values for the key and value fields as normal.

The arguments you supply are substituted for the `{arguments}` wild card in the following format:

```
java.text.MessageFormat(java.util.ResourceBundle.getBundle("bundle
name").getString("key"), new Object[] {arg1, arg2, arg3})
```

## 2.12 Managing and Creating Projects

A NetBeans project is a group of source files and the settings with which you build, run, and debug those source files. In the IDE, all development has to take place within a project. For applications that involve large code bases, it is often advantageous to split your application source code into several projects.

### 2.12.1 How to Create a Project

NetBeans IDE provides wizards that enable you to create projects that are based on project templates. The IDE includes several project templates designed to support different types of development including general Java applications, Java web and enterprise applications, HTML5 applications and PHP applications.

**To create a project:**

1. Choose **File > New Project** (Ctrl-Shift-N) from the main menu.
2. Select the appropriate project template in the New Project wizard.
3. Follow the steps in the remainder of the wizard.

For information on creating a Java Application Project, see [Chapter 6, "Creating Java Projects."](#)

For information on creating a Java Web Application Project, see [Chapter 12, "Developing Web Applications."](#)

For information on creating a Java EE Project, see [Chapter 14, "Developing Enterprise Applications."](#)

For information on creating an HTML5 Application Project, see [Chapter 20, "Developing HTML5/JavaScript Applications."](#)

For information on creating a PHP Application Project, see [Chapter 21, "Developing PHP Applications."](#)

For information on creating a Java ME Application Project, see [Chapter 22, "Developing Java ME Applications."](#)

## 2.12.2 How to Work with Character Encodings for a Project

By default, newly-created projects in the IDE use UTF-8 character encoding. This encoding determines how the IDE interprets characters beyond the ASCII character set. The IDE displays and saves any new files you create using the encoding set by the project in which they reside. If you want to change encoding properties, the IDE provides you with the ability to do so manually.

The IDE implements the FileEncodingQuery (FEQ) layer model. FEQ is an interface for obtaining information about which encoding can be used for reading from/writing to a particular file. The layer model prioritizes encoding based on the following hierarchy:

1. **File FEQ.** The encoding value declared within a file.
2. **Project FEQ.** The value of the current global project encoding in a session.
3. **Fallback FEQ.** The encoding of the locale in which the IDE is running.

Project created in NetBeans IDE 5.x and older did not implement the FEQ and will be opened using the fallback FEQ, the default locale set by your system's environment.

---

**WARNING:** If you change the project encoding property on a project that already contains files that were created using a specific character encoding, there is a risk that compiling and running the project may not succeed. This is due to the fact that the programming compiler needs to be passed an encoding value, and there can only be one such value. Neither the IDE nor the programming language performs automatic encoding detection of files.

---

### 2.12.2.1 Changing the Character Encoding of a Project

When you change the encoding for a project, all new files are created using the new project encoding.

#### To change the character encoding for a project:

1. Right-click the project node in the Projects window and choose **Properties**.
2. In the left column under Categories, select **Sources**.
3. In the Encoding drop-down list, select the character encoding that you want to be applied to the project. Click OK. The new encoding is applied to the project you are working in.

---

**Note:** The new value that you set for project encoding is retained as the global project encoding value for new projects. Therefore, when you create a new project, the IDE uses the encoding value of the previously created project for the new project.

---

### 2.12.3 How to Organize Projects into Groups

You can create groupings of projects so that you can open and close several projects at once. In addition, each group can have a different main project. After you create a project group, that project group is available in the Project Group menu. When you select a project group from that menu, all other open projects are closed. If you want to close all open projects, choose **File > Project Group > (none)**.

**To create a project group:**

1. Choose **File > Project Groups** from the main menu.
2. Click **New Group** in the Manage Groups dialog box.
3. Type the name of the new group in the Create New Group dialog box.
4. Select the options for the group. Click **Create Group**.

If you select the **Automatically Save Project List** option in the dialog box, projects that you open or close will automatically be added to or removed from the group.

**To create a project group that contains a specific set of projects:**

1. Close any projects that you do not want to include in the group.
2. Choose **File > Project Groups** to open the Manage Groups dialog box.
3. Click **New Group**.
4. Select **Free Group** in the Create New Group dialog box.
5. Select **Use Currently Open Projects**.

If you created a free group and deselected the **Automatically Save Project List** option when you created the group, you can perform the following steps to modify the contents of the group.

**To modify the contents of a free group:**

1. Open the project group that you want to modify.
2. Open any projects that you want to add to the group.
3. Close any projects that you want to remove from the group.
4. Choose **File > Project Groups**.
5. Select the group or groups in the Manage Groups dialog box. Click **Properties**.
6. Select **Automatically Save Project List** in the Project Group Properties dialog box. Click **OK**.

## 2.13 Working with Source Files in the IDE

The IDE provides tools that enable you to find, compare and move the files in your projects. You can also modify the properties of files and create file templates that you can use when you create files.

For information on editing Java source files, see [Chapter 7, "Working with Java Code."](#)

### 2.13.1 How to Find Files in Projects

You can use the Find in Projects command to locate the occurrences of specific text strings in files in your project. You can invoke the command from the popup menu in the Projects, Files and Favorites windows.

The results of the search are displayed in the Search Results window. The Search Results window displays a node for each file that contains the search string. When a file contains more than one occurrence of the string you can expand the node to view a list of each occurrence in the file. You can double-click an occurrence in the Search Results window to open the file in the editor at the line containing the search string.

**To find a string in project files:**

1. Right-click the project or folder that you want to search and choose **Find** in the popup menu to open the Find in Projects dialog.

Alternatively, you can select an element in the Projects, Files or Favorites windows and choose **Edit > Find** in the main menu.

2. Type the search string in the Containing Text field.

3. Specify any additional options that you want to apply to the search string.

4. Specify the Scope of the search.

By default the scope is the folder that you selected, but you can use the drop-down menu to modify the scope.

5. Specify any File Name Patterns to limit the search to certain types of files.

6. Click **Find**.

## 2.13.2 How to Specify Editor Formatting Options

You can specify how the source editor formats source code for various languages. You can specify formatting options globally to apply to all files in that language and also at the project level to specify the options for a specific project.

**To specify global editor formatting options:**

1. Choose **Tools > Options** in the main menu to open the Options window.

2. Click the **Formatting** tab in the **Editor** category.

3. Select a language in the Languages drop-down list.

4. Select a formatting category in the Category drop-down list.

5. Specify the formatting options for the category. Click **Apply** in the Options window.

In addition to global settings, you can specify the editor formatting options at the project level that will only apply to the current project. You can also use the formatting options specified for an existing project and apply the options to the current project.

**To specify editor formatting options for a project:**

1. Right-click the project node in the Projects window and choose **Properties** in the popup menu.

2. Select **Formatting** in the Project Properties window.

3. Select **Use project-specific options**.

4. Select a language in the Languages drop-down list.

Alternatively, click **Load from other project** to use the formatting options from an existing project.

5. Select a formatting category in the Category drop-down list.

6. Specify the formatting options for the category. Click OK.

### 2.13.3 How to Compare Two Files

You can use the diff viewer included in the IDE to compare source files. You can export a diff patch file directly from the diff viewer window.

The IDE can display diffs in either of two modes:

- **Graphical Diff Viewer.** (Default) The graphical diff viewer displays the two files side by side and uses background highlight colors to display the differences between the two files. By default, changed lines are highlighted in blue, added lines are highlighted in green, and removed lines are highlighted in red.
- **Textual Diff Viewer.** The textual diff viewer shows you the diff output in text so that you can copy and paste the output into a file or into an email.

You can use either the built-in diff engine or a command-line diff engine.

- **Built-in Diff Engine.** (Default) The built-in diff engine enables you to compare two files without your needing a command-line diff executable installed on your system.
- **Command-line Diff Engine.** Before you can use the command-line diff engine on a Microsoft Windows system, you must have a command-line diff executable (for example, from a Cygwin distribution) installed on your system. For the UNIX environment, there is a diff executable installed and available to you by default.

You can specify the diff engine in the Diff tab in the Miscellaneous category of the Options window. You can open the Diff tab by clicking Options in the diff viewer window or by choosing **Tools > Options** in the main menu.

#### To compare two files:

1. Select two files in the IDE.
2. Right-click and choose **Tools > Diff** from the pop-up menu to run the diff command.

By default, the IDE opens the diff viewer window in graphical mode. You can click the Textual tab to view the diff in the textual diff viewer.

You can click Options in the diff viewer window to open the Options window and select the diff engine that is used to find differences between files.

3. Click Export in the diff viewer window to save the diff as a diff patch file.

### 2.13.4 How to Apply a Diff Patch to a File

A patch file enables you to modify or patch a file based on the differences between the two versions of the file. The differences between the versions are contained in a diff patch file. Patch files enable software developers who are not sharing a common repository to distribute and integrate changes that have been made to the code. The IDE enables you to create and apply patches that update copies of source files so that you do not have to incorporate the changes manually.

You can apply a patch to an individual file or a folder. Patches that are applied to folders use relative paths to the files within them. Folder patches must be applied on the same folder node that the patch was created on to ensure that the patch is applied properly. If you are uncertain to which file or folder the patch should be applied, you can find the context information in the patch file itself. Patch files generated by the IDE

contain the context in which the patch was originally created in the first few lines of the file.

**How to apply a patch file to a file:**

1. In the Projects window, right-click the file or folder to which you want to apply the diff patch and choose **Tools > Apply Diff Patch** in the popup menu.
2. Locate the patch file in the file browser and click Patch.

When you apply the patch, the IDE applies the changes contained in the selected patch to the chosen file or folder and informs you that patch is successfully applied. In the Question dialog box, click Yes to view applied changes in the Diff Viewer or No to close the dialog box.

For details on applying patches to files that are under version control, see [Section 3.3.16, "How to Create and Apply a Patch \(Subversion\)"](#), and [Section 3.5.17, "How to Create and Apply a Patch."](#)

## 2.13.5 How to Access Files Outside of a Project

When developing applications, you must create a project for your source code and related files. If you need to access a file outside of your projects, you can do either of the following things:

- Choose **File > Open File** in the main menu and navigate to the location of the file on your system.
- Use the Favorites window to access the file.

**To use the Favorites window:**

1. Choose **Window > Favorites** (Ctrl-3) in the main menu.
2. Right-click in the Favorites window and choose Add to Favorites.
3. Select the folder that you want to access and click OK.

Project-level commands, such as Run Project, are not available from the Favorites window. For such commands to be available, you need to have a project open in the Projects window.

## 2.13.6 How to Create a File Template

You can save a file as a template and add it to the types of templates available in the template chooser in the New File wizard.

**To create a new template:**

1. In the Projects window, right-click the file you would like to turn into a template and choose **Save As Template**.
2. In the Save As Template dialog box, select a category for the template and click OK.

**To modify a an existing template:**

1. Choose **Tools > Templates** from the main menu.
2. In the Template Manager, expand the appropriate category node and select the template.
3. Click **Open in Editor**.

4. In the Source Editor, make the desired changes to the template.
5. Choose **File > Save** from the main menu.

In the Template Manager, you can click **Duplicate** to create new templates based on existing templates. You can create new folders and move templates between folders.

---

**Note:** Templates are not project-specific. If you modify a template that is used by a wizard in the IDE the modified template will be used by that wizard for generating files in all projects.

---

### 2.13.7 How to Work with Unknown File Extensions

If you want to work with a file of a type that the IDE does not recognize, you can associate it with one of the file types that the IDE is familiar with.

**To register a file extension as belonging to a specific file type:**

1. Choose **Tools > Options** from the main menu.
2. Click the **Files** tab in the **Miscellaneous** category.
3. Click **New** to open the **Enter New File Extension** dialog box.
4. Enter the file extension that you want to register and click **OK**.
5. In the **Associated File Type (MIME)** field, select the file and MIME type that you want to associate with your new file extension.

---

**Note:** For some object types, such as Java objects, it is not possible to add an extension.

---

### 2.13.8 How to Specify Files to Ignore

If you have files that you do not want to be recognized by the IDE, you can configure the IDE to ignore these files.

---

**Note:** If you set the IDE to ignore **.class** files, these files are not visible in the **Files** window under the **build** folder.

---

**To specify files to be ignored:**

1. From the main window, choose **Tools > Options** and click **Miscellaneous**.
2. Click the **Files** tab and use the **Ignored Files Pattern** field to enter a POSIX-style regular expression to specify the files you want ignored.

**Table 2–27 Table of useful characters for regular expressions**

Character	Description
	Or.
^	Matches all file or directory names beginning with the subsequent characters.
\$	Matches all file or directory names ending with the preceding characters.
\	Escape character. Necessary if you want to match to a period (.) or other special character.

**Table 2–27 (Cont.) Table of useful characters for regular expressions**

Character	Description
.*	Wildcard.

### 2.13.9 How to Work with Character Encodings for Files

When creating a file for a project, the files are created using the specified project encoding. Some file types declare the encoding in the file (for example, HTML, JSP, and XML files). Therefore, when the IDE creates these files, it automatically includes character encoding declarations in the file template.

When determining the character encoding for a file, the IDE applies the FileEncodingQuery (FEQ) layer model. It first determines whether the encoding is declared in the file. If no encoding declaration can be found, the IDE presumes the file encoding is specified by the project's encoding property. If the project encoding is not specified (for example, imported or older projects), the IDE applies the encoding set by the environment in which it runs.

To change the encoding for a file (for example, HTML, XML, or JSP files) you need to change the encoding and charset specified in the corresponding tag in the file. When an encoding is specified in a file, this setting overrides the encoding set in the project.

**To manually change the character encoding for a file:**

1. Open the file in the Source editor.
2. Modify the encoding and charset tags for the file, if available.

---

**WARNING:** The IDE does not convert characters when the encoding of a file changes. If you manually change the encoding declaration within the file, and that encoding does not match the project encoding, you may encounter problems when compiling and running the project. The encoding tag affects how file contents are viewed internally - not only during runtime, but also during the design phase such as when you add content in the Source Editor.

---



---

**WARNING:** Manually changing the character encoding declaration within a file changes how the IDE reads and displays that file. When you change the file encoding, the IDE does not convert the existing contents of the file to the new encoding. Care should be taken when changing the encoding of a file because the file may contain characters that cannot be saved or that may not display properly in the new encoding.

---

### 2.13.10 How to Specify Action Items

You can create a list of things that you need to resolve in your various projects and view the list as entries in the Action Items window. The list of action items can include compilation errors that are identified by the IDE, issues in an issue tracker and tasks that are identified by specific ToDo character patterns in project files. The IDE automatically scans your projects for action items and displays the items in the Action Items window.

Action items are created by some types of errors and by the occurrence of specific character patterns in the file. The list of action items can include the following types of items.

- Compiler errors
- Hudson tasks
- Issues
- Maven POM problems
- TODO
- Whitelist violations

By default, you can create action items by adding comments that contain the following patterns to a file.

- TODO
- XXX
- FIXME
- Javadoc comments that contain the @todo keyword (Java projects)

**To create a custom pattern:**

- Choose **Tools > Options** from the main menu.
- Click the **Action Items** tab in the Team category.
- Click **Add** and type the new pattern in the table. Click OK.

**To edit or remove a pattern:**

- Choose **Tools > Options** from the main menu.
- Click the **Action Items** tab in the Team category.
- Select a pattern in the table and click **Edit** to edit the selected pattern or **Remove** to delete the selected pattern. Click OK.

The IDE includes the following patterns as action items by default.

- The pattern <<<<<, which denotes merge conflicts in CVS and other version control systems
- Lines where compiler errors are registered (Java projects)

**To open the Action Items window:**

- Choose **Window > Action Items**.

You can double-click an entry in the Action Items window to jump to the line of the file where the action item needs to be resolved or open the issue in the IDE.

**To create a filter for items in the Action Items window:**

1. Choose **Window > Action Items** to open the Action Items window.
2. Click the Filter button in the Action Items window toolbar to open the Action Items Filter dialog box.
3. Click **New** to create a new filter and type a name for the filter.
4. Select the types of action items that you want to be displayed.

5. (Optional) Click the Keywords tab and specify any additional criteria that needs to be met.
6. Click OK.

To apply a filter to the list of action items, click the arrow on the Filter button and select a filter from the drop-down list.

### 2.13.11 How to Use Bookmarks in Files

Bookmarks enable you to quickly navigate to a specific line in a file. After you set a bookmark in a file you can use keyboard shortcuts to jump to the bookmark or between bookmarks in the file. You can view a list of your bookmarks in the Bookmarks window. When you double-click a bookmark in the Bookmarks window the IDE opens the file in the source editor at the line that contains the bookmark.

#### How to toggle a bookmark in a file:

1. Place the insert cursor in the line in the file.
2. Type **Ctrl-Shift-M**.

#### How to move between bookmarks in a file:

1. Type **Ctrl-Shift-Period** or **Ctrl-Shift-Comma** to open the Bookmarks popup menu.
2. Use the up and down arrows to navigate in the popup menu. Select a bookmark in the menu to open the file at the line containing the bookmark or select <Bookmarks> in the popup menu to open the Bookmarks window.

#### How to view a list of all bookmarks:

1. Choose **Windows > IDE Tools > Bookmarks** from in the main menu.

## 2.14 Working with Resource Bundles

Resource bundles store selected characteristics of an object as key and value pairs in a set of .properties files. You can store a variety of characteristics in resource bundles, like localized strings used to internationalize your code or properties for an Ant script. The IDE displays resource bundles as properties object nodes (  ) that contain nodes for each of its locales (  ) and keys (  ).

When used for internationalizing source code, the IDE stores each locale's characteristics in a .properties file which is contained within the resource bundle. Each key corresponds to the name of a property and must be the same for each locale. The key value is the localized string displayed by the object at runtime and can vary for each locale.

The IDE also enables you to automate the process of internationalizing your source code. You can use the IDE's internationalization features to:

- Automatically replace hard-coded strings with internationalized ones
- Insert new internationalized strings as you write code
- Check your resource bundles to ensure they contain all of the necessary keys

---

**Note:** Though resource bundles and locales are both .properties files, the IDE displays resource bundles and the locales they contain using different icons in the Files window to avoid confusion.

---

## 2.14.1 How to Create and Delete Resource Bundles

The IDE enables you to manage resource bundles and the various key and locale pairs stored within them. When you use the New wizard to create a new properties file, the IDE automatically creates a `.properties` file for the default locale to which you can then add specific properties and additional locales.

A properties file is created with a `.properties` suffix. The Files window displays a properties object node () for the resource bundle with one locale subnode () for the default locale.

### To create a resource bundle:

1. Choose File > New to open the New wizard.
2. Select the project in which you want to create the resource bundle.
3. Expand the Other node in the Categories pane, select Properties File in the File Types pane, and click Next.
4. Type the file name in the File Name field.
5. Enter the location where you want to create the resource bundle. Click Finish.

### To delete a resource bundle:

1. Right-click the properties file in the Files window and choose Delete.
2. In the verification dialog box, click Yes to remove the file.

## 2.14.2 How to Edit a Resource Bundle

There are two ways of editing the properties files contained in your resource bundles:

- In the Source Editor as text files. The Source Editor displays one text file for each of the resource bundle's locales. Each locale file lists all the key and value pairs for that locale
- In the Properties Editor, a special editor that is displayed inside the Source Editor. The Properties Editor displays key and value pairs for all locales of your resource bundle in table layout.

### To edit a properties file in the Properties Editor:

1. In the Files window, right-click the properties object node () and choose Open. The Properties Editor is displayed showing all existing keys and their values for each locale.
2. If you want to change a property's key, type the new name in the Key column.
3. If you want to change a property's value for a locale, type the new value in the locale's Value column.
4. If you want to add or remove keys, use the New Property and Remove Property keys.

### To edit a property file as a text file:

1. Right-click the properties object node and choose Edit to edit the default locale.
2. If you want to edit a different locale, expand the properties object node, right-click the node for the locale you want to modify and choose Edit.

### 2.14.3 How to Add and Remove a Property

The IDE's Properties Editor enables you to manage the properties in all of the locales in your resource bundle. You can also add a property to or remove a property from only one locale.

#### To add a new property to all locales in a resource bundle:

1. In the Files window, right-click the properties object node ( ) and choose Open.  
The Properties Editor is displayed showing all existing keys and their values for each locale.
2. In the Properties Editor, click the New Property button.
3. Enter a key and its default value for the new property.

You can also add a comment along with the key and value. The comment is displayed when you click the value for that locale's key.

4. Click OK.  
The property appears in the Properties Editor with the default value entered for each existing locale.
5. If you want to set a different value for each locale, enter the new values in each locale's column.
6. Choose File > Save to save the file.

#### To add a new property to a specific locale:

1. In the Files window, expand the properties object node for the resource bundle that contains the locale.
2. Right-click the node of the locale to which you want to add a property and choose Add Property.
3. Enter a key and a value for the new property and click OK.
4. Choose File > Save to save the file.

#### To remove a property from all locales in a resource bundle:

1. In the Files window, right-click the properties object node and choose Open.
2. In the Properties Editor, select the key for the property you want to remove.
3. Click the Remove Property button.
4. Click Yes in the verification dialog box.

#### To remove a property from a specific locale:

1. In the Files window, right-click the properties object node and choose Delete.
2. Click Yes in the verification dialog box

You can also delete an entire resource bundle and all of the .properties files contained within it by right-clicking its properties object node ( ) and choosing Delete.

## 2.14.4 How to Add and Remove a Locale

The New Locale dialog box provides an extensive list of predefined locales. It also lets you define your own locale by choosing the appropriate language code, country code, and language variant.

### To add a locale:

1. Right-click the properties object node (  ) in the Files window and choose Add Locale.
2. Select a language code, country code, and variant from the drop-down lists. You can also select from the list of predefined locales in the bottom of the dialog box.
3. Click OK to add the locale.

### To remove a locale:

1. Expand the resource bundle's object node in the Files window.
2. Right-click the locale and choose Delete.

You can also delete an entire resource bundle and all of the `.properties` files contained within it by right-clicking its properties object node (  ) and choosing Delete.

## 2.14.5 How to Edit a Locale

The localized strings for each locale are stored in that locale's `.properties` file. You can modify a locale in the Source Editor as a text file or in the Properties Editor. You can also edit a specific locale using its customizer.

### To edit a locale in the Properties Editor:

1. Right-click the properties object node (  ) in the Files window and choose Open. The Properties Editor is displayed showing all existing keys and their values for each locale.
2. If you want to change a property's key, type the new name in the Key column.
3. If you want to change a property's value for a locale, type the new value in the locale's Value column.
4. If you want to add or remove keys, use the New Property and Remove Property keys.

### To edit a locale as a text file in the Source Editor:

1. Right-click the properties object node and choose Edit to edit the default locale.
2. If you want to edit a different locale, expand the properties object node, right-click the node for the locale you want to modify and choose Edit.

### To edit a locale using the customizer:

1. Expand the properties object node in the Files window.
2. Right-click the desired locale and choose Customize. The Locale customizer opens listing the locale name and each existing key.
3. If you want to switch locales, click the ellipsis (...) button. This option is not available for the default locale.

4. If you want to add a new property to the locale, click Add Key and enter the key name and value.
5. If you want to remove a property from the locale, select the property and click Remove Key.

---

**Note:** You cannot edit the existing values of properties with the customizer.

---

## 2.15 Working with Javadoc Documentation

Javadoc is a tool for generating API documentation in HTML format from doc comments in source code.

NetBeans supports the Javadoc standard for Java documentation: both viewing and generating Javadoc. It provides a solid documentation tool when working with code.

### 2.15.1 How to Add Javadoc to a Project

You can make Javadoc documentation for a JAR file's class available in the IDE by associating that documentation with the JAR file.

When you add a required project to a project's classpath, the required project's Javadoc and sources are automatically added to the project as well.

When you create a Java class library for a single JAR file, you can simply add the JAR file to the project's classpath to make the associated Javadoc and source code available. If your Java library contains multiple JAR files, however, you must add the library itself to the classpath. Adding the library to the classpath also makes it easier to share the project with other developers.

#### To add Javadoc for a JAR file:

1. Choose **Tools > Libraries** from the main menu.
2. In the left pane of the Ant Library Manager, select the project library within which the JAR file you want to add Javadoc documentation to is located.

---

**Note:** Only libraries already registered with the IDE are listed in the Ant Library Manager's Class Libraries list.

---

3. If the JAR file for which you want to add Javadoc documentation has not already been added to a registered library, create a new empty library using the New Library button. Next, in the Classpath tab click **Add JAR/Folder** and specify the location of the JAR file containing the compiled class files.

---

**Note:** You can also associate the Javadoc with a JAR file using the project's Project Properties window. However, doing so creates the association only for that project. Open the Project Properties dialog box by right-clicking the project node and choosing Properties. Select the Libraries node in the Categories pane. Then select the JAR with which you want to associate the Javadoc and click Edit. You can then specify the sources to be associated.

---

A class library can contain multiple JAR files as well as their Javadoc documentation and source code.

---

4. In the Javadoc tab, click **Add ZIP/Folder** and specify the location of the Javadoc files.
5. Click **OK** to exit the Ant Library Manager.

### 2.15.2 How to Add the JDK Javadoc to the IDE

The API documentation for your Java platform can be used for reference or as a learning tool.

**To add JDK Javadoc to the IDE:**

1. Choose **Tools > Java Platforms** from the main window.
2. Select the platform to which you want to add Javadoc in the left panel of the dialog box.
3. In the Javadoc tab, click **Add ZIP/Folder** and specify the location of the Javadoc files. Click **Close**.

---

**Note:** You can download and install Java SE Documentation files from the Oracle Technology Network (for example, JDK 7 documentation is available at <http://www.oracle.com/technetwork/java/javase/documentation/java-se-7-doc-download-435117.html>.)

---

### 2.15.3 How to View Javadoc Documentation

After you add a Javadoc library to the project, you can view the documentation for any of the library's classes in the Source Editor, browse the documentation in your external browser, and search the library using the Javadoc Index Search.

To view Javadoc documentation for any elements of code you are writing, you must add the Javadoc library containing the documentation to the project.

**To view Javadoc documentation:**

- In the Source Editor, place the pointer on the code element whose documentation you want to display and do one of the following:
  - Choose **Source > Show Documentation** (or press Ctrl-Shift-Space) to view Javadoc documentation for the selected element in a separate IDE window.
  - Press Alt-F1 to view Javadoc documentation in an external browser window.
  - Choose **Window > Other > Javadoc** to open the Javadoc page that will show Javadoc dynamically for the code elements you are editing.
- Choose **Help > Javadoc Index Search** (Shift-F1). The Javadoc Index Search window opens in an editor tab and displays the results of the search for the code element that is currently selected in the Source Editor. The HTML Viewer displays the Javadoc for the selected element in the search results. Click the Toggle the Display of the HTML Viewer button to hide or show the HTML viewer. You can double-click an element in the search results to open the Javadoc documentation for the element in a page in your browser.

### 2.15.4 How to Generate Javadoc Documentation

For each of your projects, you can produce a set of Javadoc HTML pages that describe the project's classes, inner classes, interfaces, constructors, methods, and fields. The

Javadoc is constructed from the structure of your code and the Javadoc comments embedded in your code. You can also configure how the IDE generates Javadoc documentation for each of your projects.

**To generate Javadoc documentation for a project:**

1. Select the project in the Projects window.
2. Choose **Run > Generate Javadoc for Project**.

The IDE generates the Javadoc to the `dist/javadoc` folder in your project directory and opens the index page in the IDE's designated web browser.

You can select multiple projects in the Projects window and generate Javadoc documentation for them at once by choosing **Run > Generate Javadoc (number of selected projects) Projects** from the main IDE's menu.

**To configure how the IDE generates Javadoc documentation:**

1. In the Projects window, right-click the project node and choose **Properties**.
2. Expand the Build node and select **Documenting** in the left pane of the dialog box.
3. Set the desired options and click OK.

---

**Note:** In Free-form projects the Generate Javadoc command is disabled by default. If your Ant script contains a target for generating Javadoc documentation, you can map the target to the Generate Javadoc command in the Project Properties dialog box's Build and Run panel.

---

**To create a Javadoc stub:**

- Place the cursor above a method or a class that has no Javadoc, type `/**`, and press Enter.

The IDE creates a skeletal structure for a Javadoc comment filled with some content. If you have a Javadoc window open, you will see the changes immediately while you are typing.

## 2.15.5 How to Enter Javadoc Comments in Source Code

You can use editor hints to automatically generate basic Javadoc comments for code elements in your source files. The generated comments include required tags for the particular code element. For example, if a method takes a parameter, a `@param` tag is inserted. You can also use hints to generate corrections to Javadoc comments, such as when the comments contain incorrect tags.

You are notified of an editor hint by a light bulb icon that appears in the left margin of the Source Editor. You can read the hint by clicking the light bulb icon or by pressing Alt-Enter. You can generate the code suggested by the hint by clicking the hint or by pressing Enter.

The hints for Javadoc generation are turned off by default.

**To turn on hints for Javadoc comments:**

1. Choose **Tools > Options > Editor > Hints**.
2. On the Hints tab, select Java from the Language drop-down list.
3. Select the Javadoc checkbox or expand it to fine-tune your choice.

By default, the Javadoc hints work for protected elements and public elements. If you would like to change the scope of the hints, expand the Javadoc node and select one of the subnodes. Then choose the radio button for the scope to which you would like the hints to apply.

For more information about Javadoc tags, see:

<http://java.sun.com/javase/6/docs/technotes/tools/solaris/javadoc.html>

For information on how to write Javadoc comments, see:

<http://java.sun.com/products/jdk/javadoc/writingdoccomments/index.html>

## 2.15.6 How To Analyze and Fix Javadoc Comments

To identify the places in your code that need Javadoc comments and quickly insert these comments, you can use the Javadoc Analyzer tool available in the Java Editor.

### To analyze and fix Javadoc comments:

1. Select a project, a package, or an individual file and choose **Tools > Analyze Javadoc** from the main menu.  
The Analyzer window displays suggestions for adding or fixing Javadoc comments, depending on the scope of your selection.
2. Select one or several checkboxes where you would like to fix Javadoc and click the **Fix Selected** button.
3. Click **Go Over Fixed Problems** and use the **Up** and **Down** arrows to actually add your comments. This might be helpful if you selected to fix several instances at once and now want to revisit the stubs.

## 2.16 Viewing IDE Notifications

The Notifications window displays a list of all notifications that occurred in the current IDE session. Notifications indicate the change in the status of various IDE processes, including available IDE updates, the status of builds and test results. A new notification is indicated by a notification icon in the status bar of the IDE. The notification icon is displayed in the status bar until you view the notification.

### How to view IDE notifications:

1. Perform either of the following to view the list of IDE notifications.
  - Click the Notifications icon in the status bar, if available.
  - Choose **Window > IDE Tools > Notifications** from the main menu.
2. Select a notification in the list to display details about the notification.



---

## Versioning Applications with Version Control

This chapter describes using popular version control packages with the NetBeans IDE.

This chapter contains the following sections:

- [About Versioning Applications with Version Control](#)
- [Versioning Applications with Git](#)
- [Versioning Applications with Subversion](#)
- [Versioning Applications with Mercurial](#)
- [Versioning Applications with CVS](#)
- [About Local History](#)

### 3.1 About Versioning Applications with Version Control

Versioning applications with version control is a method of coordinating the efforts of multiple team members, sometimes at different locations in an organization, in a way that avoids problems that can arise when different people may be working on the same application. To help prevent making conflicting changes to a source file, version control typically uses a workflow similar to this:

- Each user updates a file before working on it, to ensure starting with the latest changes and revisions
- Each user checks out (or locks) the file to prevent other team members from conflict
- After making changes (and verifying them in a local build or other method), the user checks in (or commits) the file to the team's shared repository

Although these techniques help prevent conflicting changes being made to the team's shared files, version control systems also contain tools for resolving conflicts and for reverting to previous versions in the case of problems with newly introduced material.

The rest of this chapter describes how each of the version control systems (Git, Subversion, and Mercurial) apply these principles to the practice of versioning applications, and also include information on available tools for working in a file's local history.

### 3.2 Versioning Applications with Git

Git is a free and open source, distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

The IDE's Git support enables you to perform versioning tasks directly from your project within the IDE. You can call Git commands on both files and directories in the Projects, Files and Favorites windows, in the IDE. The IDE also provides a graphical Diff Viewer, enabling you to compare file revisions, as well as supporting inline diff directly in the editor.

The advantages of a distributed revision control system like Git are:

- Better support for distributed teams by removing a centralized bottleneck.
- Better scalability with large numbers of concurrent users.
- After the initial clone, faster to work with, independent of a user's network infrastructure.

### 3.2.1 Git Visualization Features

The IDE provides several file status information tools that simplify the process of working with version-controlled files, including:

- **Color Coding.** Enables you to view the current status of version-controlled files.
- **Annotations.** Enables you to view revision and author information for each line of version-controlled files.

Since Git is a distributed revision control system, you typically begin by cloning an external repository to work with. This clone is a complete copy of the repository including the revision history. You can clone this local copy as often as you like, and when you want to you can push your changes back to the original repository provided you have permissions, or export your changes and send them to the owner if you do not.

For further documentation on the Git support and Git itself, see the following resources:

- NetBeans Git Home:  
[http://netbeans.org/projects/versioncontrol/pages/Git\\_main](http://netbeans.org/projects/versioncontrol/pages/Git_main)
- NetBeans Git User's Guide:  
<https://netbeans.org/kb/docs/ide/git.html>
- Git Home:  
<http://git-scm.com/>
- Git Documentation:  
<http://git-scm.com/documentation>

### 3.2.2 How to Initialize a Git Repository

The IDE enables you to initialize a Git repository from existing files that are not in source control yet.

#### To initialize a Git repository:

1. In the Projects window, select an unversioned project and choose either:
  - **Versioning > Initialize Git Repository** from the node's context menu.
  - **Team > Git > Initialize Repository** from the IDE's main menu.
2. In the Initialize a Git Repository dialog box, specify the path to the repository you are going to store your versioned files, or click Browse and browse for the directory required.

**3. Click OK.**

A .git subfolder is created in the folder you specified in step 2 (your NetBeans project folder by default), which is your Git repository where all the data of your project snapshots are stored. Git starts versioning all files in the folder you specified. You can view files being added to the repository from the Output window (**Window > Output**).

- All the project files are marked **Added** in your Working Tree.
- After you initialized the Git repository, you either add files or directly commit them to the Git repository.

### 3.2.3 How to Clone a Git Repository

The IDE enables you to clone an external Git Repository and to make it available within the IDE. You effectively create a copy or clone of the entire repository to work with in the IDE.

**To clone a Git Repository:**

1. Choose **Team > Git > Clone** from the IDE's main menu. The Clone Repository wizard displays.
  2. In the Remote Repository panel of the wizard, specify the URL that contains the connection protocol and the location of the Git repository, user name and password (you can save the latter for the future if required).
  3. (Optional) Press **Proxy Configuration** to display the Options dialog box and set the proxy server settings. Click **OK** when finished.
  4. Click **Next** to switch to the next step of the wizard.
  5. In the Remote Branches panel, select the repository branch(es) to be fetched to your local repository. Click **Next**.
  6. In the Destination Directory panel, specify the following:
    - In the Parent Directory field, the path to the directory intended for the cloned repository on your hard drive (alternatively, click the Browse button and navigate to the directory). The Parent Directory field is pre-filled with the path to the default NetBeansProjects directory where all NetBeans projects are stored.
    - In the Clone Name field, the name of the local folder where the original project will be cloned to. By default Clone Name is filled out with the actual Git repository name.
    - In the Checkout Branch field, selected the branch to be checked out into the working tree.
    - In the Remote Name field, the name that represents the original repository being cloned. origin is the default alias of the repository being cloned. It is a recommended value.
    - Leave the Scan for NetBeans Projects after Clone checkbox selected to activate after-scanning right after the clone finishes. (The plugin searches for NetBeans projects in the cloned resources and offers to open the found projects.)
  7. Click **Finish**.
- After a Git repository is cloned, the metadata .git folder is created inside the folder you selected in the wizard.

---

**Note:** Select the **Scan for NetBeans projects after clone** option if you intend to immediately work with the cloned sources.

If the sources contain projects, a dialog will appear prompting you to open them in the IDE.

If the sources do not contain a project, the dialog will appear prompting you to create a new project from the sources and then open them in the IDE. When you create a new project for such sources, select the appropriate project category in the Create Project wizard and then use the Create Project with Existing Sources template in that category.

---

The IDE supports the following Git URLs:

Protocol	Access Method	Example
file	Direct repository access (on local disk)	file:///path_to_repository or path_to_repository
http	Access using HTTP protocol	http://hostname/path_to_repository
https	Access using HTTP protocol with SSL encryption	https://hostname/path_to_repository
ssh	Access using SSH protocol	ssh://hostname/path_to_repository
sftp	Access using SFTP protocol	sftp://hostname/path_to_repository
git	Access using GIT protocol	git://hostname/path_to_repository

### 3.2.3.1 Cloning a Repository from GitHub using SSH protocol

To clone a repository from GitHub using the SSH protocol, proceed as follows:

---

**Note:** You need to have a GitHub account and be a project member in order to clone using SSH.

---

1. Choose **Team > Git > Clone** from the main menu. The Clone Repository wizard displays.
2. At the **Remote Repository** page of the Clone Repository wizard, specify the path to the repository required in the **Repository URL** field, for example, `git@github.com:tstupka/koliba.git`.
3. Verify `git` is specified in the **Username** text field.
4. Select the **Private/public key** option.
5. **(Skip if using SSH-agent or Pageant for automated SSH access to the Git server.)** Complete the following steps to access the Git server using your private SSH key and a passphrase:
  1. Specify the path to the key file, for example `C:\Users\key`.

---

**Note:** The OpenSSH private key format is required. Keys generated by PuTTYgen for Microsoft Windows must be converted to the OpenSSH format before using them in the IDE.

---

2. Enter the passphrase for the key file, for example, abcd.
3. (Optional) Select the **Save Passphrase** option if required.
6. (Applies if using SSH-agent or Pageant for automated SSH access to the Git server.) Leave the **Private Key File** and **Passphrase** fields empty to get authenticated access from the IDE to the Git server through correctly configured SSH-agent or Pageant.
7. (Optional) Click **Proxy Configuration** to display the **Options** dialog box and set the proxy server settings. Click **OK** when finished.
8. Click **Next**.
9. At the **Remote Branches** page, select the repository branch(es) to be fetched (downloaded) to your local repository, for example `master`.
10. Click **Next**.
11. At the **Destination Directory** page, specify the following:
  - In the **Parent Directory** field, the path to the directory intended for the cloned repository on your hard drive (alternatively, click the **Browse** button and navigate to the directory).  
The **Parent Directory** field is pre-filled with the path to the default `NetBeansProjects` directory where all NetBeans projects are stored.
  - In the **Clone Name** field, the name of the local folder where the original project will be cloned to.  
By default, **Clone Name** is filled out with the actual Git repository name.
  - In the **Checkout Branch** field, select the branch to be checked out into the working tree.
  - In the **Remote Name** field, the name that represents the original repository being cloned.  
`origin` is the default alias of the repository being cloned. It is a recommended value.
  - Leave the **Scan for NetBeans Projects after Clone** checkbox selected to activate after-scanning right after the clone finishes. (The plugin searches for NetBeans projects in the cloned resources and offers to open the found projects.)
12. Click **Finish**.  
After the repository is cloned, the **Clone Completed** message displays.
13. Choose the desired option.

### 3.2.4 How to Add Files to a Repository

The IDE enables you to track a new file and also to stage changes to an already tracked file in the Git repository. You effectively add your sources into a local Git repository. The repository files are placed under a `.git` directory under the project directory.

**To add files to a Git Repository:**

When adding files to a Git repository, the IDE composes and saves snapshots of your project first in the Index. After you perform the commit, the IDE saves those snapshots in the HEAD. The IDE allows you to choose between two workflows:

- Explicitly add new or modified files to the Index and then commit only those that are staged in the Index to the HEAD.
- Skip adding new or modified files to the Index and commit the required files directly to the HEAD.

**To add files to the index and then commit those that are staged:**

1. In the Projects window, right-click the file you want to add.
2. In the context menu, choose **Git > Add**. This adds the file contents to the Index before you commit it.
3. In the Projects window, right-click the file you want to commit.
4. In the **Commit** dialog box, select the **Changes between HEAD and Index** toggle button. This displays the list of files that are already staged.
5. Commit the file as described in [Section 3.2.5, "How to Commit Sources to a Git Repository."](#)

**To skip adding files to the Index and commit the files directly:**

1. In the Projects window, right-click the file you want to commit.
2. In the context menu, choose **Git > Commit**.
3. In the Commit dialog box, select the Select the Changes between Index and Working Tree toggle button. This displays the list of files that are not staged.
4. Commit the file as described in [Section 3.2.5, "How to Commit Sources to a Git Repository."](#)

### 3.2.5 How to Commit Sources to a Git Repository

The IDE enables you to commit files to a Git repository. The IDE executes the commit and stores your modifications to the repository upon a successful commit.

**To commit versioned files to a repository:**

1. Choose **Team > Commit** from the IDE's main menu. The Commit dialog box displays.
2. Type in a commit message in the Commit Message text area. Alternatively, you can do any of the following:
  - Click **Recent Messages** to view and select from a list of messages that you have previously used.
  - Click **Load Template** to select a message template.
3. Specify the Author's and Committer's names in the respective fields.
4. Select the files to be committed in the Files to Commit section.
5. In the Update Task section, specify and modify tasks related to the change being committed, if required.
6. After specifying actions for individual files, click **Commit**.

---

**Note:** The IDE's status bar, located in the bottom right of the interface, displays as the commit action takes place.

Upon a successful commit, versioning badges disappear in the Projects, Files and Favorites windows, and the color coding of committed files returns to black.

---

### 3.2.6 How to Revert Modifications

You can throw away local changes made to selected files in your Working Tree and replace those files with the ones in the Index or HEAD.

**To revert modifications:**

1. Select the versioned project, file or folder for which you want to revert changes.
2. Choose **Team > Revert Modifications** from the IDE's main menu or **Git > Revert Modifications** from the selected item's context menu.
3. In the **Revert Modifications** dialog box, select any of the following options:
  - **Revert All Uncommitted Changes in Working Tree and Index.** Deletes all uncommitted changes and returns to the state of files in HEAD.
    - **Remove also New Files and Folders.** Deletes new files and folders that have not been committed to HEAD.
  - **Revert Uncommitted Changes in Working Tree to the State in Index.** Deletes uncommitted changes in Working Tree and returns to the current state of files in Index.
    - **Remove also New Files and Folders.** Deletes new files and folders that have been created in Working Tree but have not been added to Index.
  - **Revert only Uncommitted Changes in Index to HEAD.** Deletes uncommitted changes in Index and returns to the state of files in HEAD.
4. Click **Revert**. The IDE replaces the selected files with those you specified in the selected option.

### 3.2.7 How to Reset

You can cancel changes and bring your repository back to a particular commit.

**To reset:**

1. Select the versioned project, file or folder.
2. Select **Team > Revert/Recover > Reset** from the IDE's main menu or **Git > Revert/Recover > Reset** from the selected item's context menu.
3. In the **Git Reset** dialog box, specify any of the following options:
  - **Revision.** Specify the required revision by entering a commit ID, existing branch, or tag name in the Revision field or click **Select** to view the list of revisions maintained in the repository. The following fields display information specific to the selected revision:
    - **Commit ID.** A unique identifier of the specified revision.
    - **Author.** A unique identifier of the specified revision.
    - **Message.** A message specified during the commit of the revision.

- **Options.** Choose any of the reset modes below:
    - **Do Not Modify Index or Working Tree (--soft).** Select to move HEAD to the specified commit and leave all changes in Index and Working Tree.
    - **Modify Working Tree (--mixed).** Select to overwrite all changes in Index and leave changes in Working Tree.
    - **Update Index and Working Tree (--hard).** Select to overwrite all changes that are made in your Index and Working Tree.
4. Click **Reset**. The IDE discards changes in accordance with the selected option.

### 3.2.8 How to Create a Tag

You can create tags to refer to a particular commit. This can simplify searching for elements in a specific commit.

#### To create a tag:

1. Specify the following in the **Create Tag** dialog box:
  - **Tag Name.** Specify the name of the tag being created.
  - **Tag Message.** Enter a comment to be included with the tag.
  - **Force Update.** Select to displace an existing tag with an identical name.
  - **Revision.** Type a specific revision of the selected item by entering a commit ID, existing branch, or tag name. Alternatively, click **Select** to open the Select Revision dialog box where you can view the list of revisions maintained in the repository and choose the required one.
2. Review information pertinent to the specified revision in the following fields:
  - **Commit ID.** A unique identifier of the specified revision.
  - **Author.** Name of a person who committed the revision.
  - **Message.** A message specified during the commit of the revision.
3. Click **Create**.

The tag is added to the specified revision.

### 3.2.9 How to Compare File Revisions

The IDE's graphical Diff Viewer enables you to compare revisions of a file side by side using color coding to emphasize the differences between the files being compared.

To access the Diff Viewer, select a version-controlled file or folder (for example, from the Projects, Files or Favorites window) and choose either **Git > Diff > Diff To HEAD** from the context menu or **Team > Diff > Diff To HEAD** from the main menu.

The following table lists the Git commands available in the toolbar of the Diff Viewer:

Icon	Name	Function
	Changes between HEAD and Working Tree	Displays a list of files that are either already staged or only modified/created and not staged yet.
	Changes between HEAD and Index	Displays a list of files that are staged.

Icon	Name	Function
	Changes between Index and Working Tree	Displays files that have differences between their staged and working tree states
	Go to Next Difference	Displays next difference in the file.
	Go to Previous Difference	Displays previous difference in the file.
	Refresh Statuses	Refreshes the status of the selected files and folders. Files displayed in the Versioning window can be refreshed to reflect any changes that may have been made externally.
	Revert Modifications	Displays the Revert Modifications dialog box.
	Commit Changes	Displays the Commit dialog box.

The Diff Viewer provides the following UI components:

Blue	Indicates lines that have been changed since the earlier revision.
Green	Indicates lines that have been added since the earlier revision.
Red	Indicates lines that have been removed since the earlier revision.

The following icons allow you to make changes directly to your local working copy:

	Inserts the highlighted text into your Working Tree copy.
	Reverts the whole local Working Tree copy.
	Removes the highlighted text from the local Working Tree copy.

### 3.2.10 How to Work with Branches

The IDE enables you to maintain different versions of an entire code base using branches, which involves:

- creating a branch
- checking out a branch
- merging branches
- deleting branches

#### 3.2.10.1 Creating a Branch

Creating a branch enables you to work on a separate version of your file system for stabilization or experimentation purposes without disturbing the main trunk.

##### To create a local branch:

1. In the Projects or Files window, choose a project or folder from the repository in which you want to create the branch.

2. Choose **Team > Branch/Tag > Create Branch** from the main menu (alternatively, right-click the versioned project or folder and choose **Git > Branch/Tag > Create Branch** in the context menu). The Create Branch dialog box displays.
3. In the Branch Name field, enter the name of the branch being created.
4. Type a specific revision of the selected item by entering a commit ID, existing branch, or tag name in the Revision field or press Select to view the list of revisions maintained in the repository and choose a required one.
5. Review the Commit ID, Author, Message fields information specific to the revision being branched from and click **Create**. The branch is added to the Branches/Local folder of the Git repository.

### 3.2.10.2 Checking Out

To edit files on a branch that already exists, you can check out the branch to copy the files to your Working Tree.

#### To check out a revision:

1. Choose **Team > Checkout > Checkout Revision** from the main menu. The Checkout Selected Revision dialog box displays.
2. Specify the revision required by entering a commit ID, existing branch, or tag name in the Revision field or press Select to view the list of revisions maintained in the repository and specify a required one.

---

**Note:** Specify the revision required by entering a commit ID, existing branch, or tag name in the Revision field or press Select to view the list of revisions maintained in the repository and specify a required one.

---

3. Review the Commit ID, Author, Message fields information specific to the revision being checked out.
4. To create a new branch out of the checked out revision, choose the **Checkout as New Branch** option and enter the name in the Branch Name field.
5. Press **Checkout** to check out the revision. Files in the Working Tree and in the Index are updated to match the version in the specified revision.

#### To check out files:

1. Choose **Team > Checkout > Checkout Files** from the main menu. The Checkout Selected Paths dialog box displays.
2. (Optional) Select the **Update Index with Entries from the Selected Revision** option to update the Index with the state in the selected revision prior to the checkout itself.
3. (Enabled if the **Update Index with Entries from the Selected Revision** option is selected.) Specify the required revision by clicking **Select**.
4. Review the **Commit ID, Author, Message** fields information specific to the revision being checked out.
5. Click **Checkout** to complete checking out the files.

### 3.2.10.3 Merging

Merging enables you to port modifications from a repository revision to the Working Tree.

#### To merge:

1. Choose **Team > Branch/Tag > Merge Revision** from the main menu. The Merge Revision dialog box displays.
2. Specify the revision required by entering a commit ID, existing branch, or tag name in the Revision field or press **Select** to view the list of revisions maintained in the repository and specify a required one.
3. Review the Commit ID, Author, Message fields information specific to the revision being merged.
4. Click **Merge**. A three-way merge between the current branch, your Working Tree contents, and the specified branch is done.

---

**Note:** After merging, you must still commit the changes in order for them to be added to the HEAD.

---

### 3.2.10.4 Deleting

You can delete an unnecessary local branch.

#### To delete a branch:

1. Select **Team > Repository Browser** from the main menu.
2. In the Git Repository Browser, select the branch to be deleted.

---

**Note:** The branch must be inactive, that is, not currently checked out into the Working Tree.

---

3. Right-click the selected branch and select **Delete Branch** from the popup menu.
4. In the Delete Branch dialog box, click **OK** to confirm the branch deletion. The branch is removed from the local repository as well as the Git Repository Browser.

## 3.2.11 How to Work with Remote Repositories

The IDE enables you to work with remote repositories hosted on the Internet or network, which involves:

- fetching
- pushing
- pulling

### 3.2.11.1 Fetching

Fetching gets the changes from the original remote repository that you do not have yet. It never changes any of your local branches. Fetching gets all the branches from remote repositories, which you can merge into your branch or just inspect at any time.

#### To fetch updates from a remote repository:

1. Select **Team > Remote > Fetch**.

2. In the first panel of the wizard, select either the Configured repository (to use the path to the repository configured earlier) or Specify Git Repository Location option (to define the path to a remote repository that has not been accessed yet, its name, login, password, private key file, passphrase, and proxy configuration if required). Click **Next**.
3. In the second panel of the wizard, select the branches to fetch changes from. Click **Finish**. A local copy of a remote branch is created. The selected branches are updated in the **Branches > Remote** directory in the Git Repository Browser. Next the fetched updates can be merged into a local branch.

### 3.2.11.2 Pulling

When pulling some updates from a remote Git repository, the changes are fetched from it and merged into the current HEAD of your local repository.

#### To pull from a remote repository:

1. Select **Team > Remote > Pull**.
2. At the Remote Repository page of the wizard, select either the Configured repository (to use the path to the repository configured earlier) or Specify Git Repository Location option (to define the path to a remote repository that has not been accessed yet, its name, login, password, private key file, and passphrase, if required) and click **Next**.
3. At the Remote Branches page of the wizard, select the branches from which you wish to pull changes and then click **Finish**. Your local repository is synchronized with the origin repository.

### 3.2.11.3 Pushing

Pushing enables you to contribute changes from your local Git repository into a public Git repository.

#### To push to a remote repository:

1. Select **Team > Remote > Push**.
2. In the first panel of the wizard, select either the Configured repository (to use the path to the repository configured earlier) or Specify Git Repository Location option (to define the path to a remote repository that has not been accessed yet, its name, login, password, private key file, and passphrase, if required) and click **Next**.
3. In the second panel of the wizard, select the branch(es) to push your edits to. Click **Next**.
4. In the third panel of the wizard, select the branch(es) to be updated in the Remotes directory of your local repository and click **Finish**. The specified remote repository branch is updated with the latest state of your local branch.

## 3.2.12 How to Set Git Global Options

Git allows you to set global options, which affect output window tabs, new files during a commit, adding a signed-off-by line to the commit log message, and ignoring non-sharable folders.

#### To set Git global options:

1. Select **Tools > Options**. The IDE's Options window opens.

2. Select the **Team > Versioning** tabs, then select Git under Versioning Systems.

The following Git Global Options properties can be set:

---

Automatically open Output window tabs	If selected, the Output window displays when versioning operations that involve interaction with the Git repository take place (for example, clone, fetch, pull, push).
Exclude new files from commit automatically	Select to omit new files during the commit.
Add Signed-off-by line by the committer at the end of the commit log message	Select to add a committer's name as a signer in a log message during the commit.
Permanently ignore non-sharable folders	When selected, the IDE ignores files and folders not meant to commit (for example, build and dist for J2SE projects) and adds paths to them to the .gitignore file.

---

### 3.2.13 How to Shelve Changes (Git)

Shelving allows to make changes to a project without committing and pushing them to the Git versioning system.

**To temporarily set aside some not yet committed changes in a working directory as a patch file:**

1. Select a versioned project, file, or folder.
2. Select **Team > Shelve > Shelve Changes** from the main menu.
3. Specify the name for a patch to be shelved in the **Patch Name** field.
4. Specify the options for the patch in the **Options** pane of the dialog box.
5. Click **Shelve**.

The IDE stashes away the not yet committed changes contained in the selected project, file, or folder into a patch file and reverts the modified files in the working copy (the working copy gets clear of the local changes). The previously modified files become up to date.

### 3.2.14 How to Rebase (Git)

Rebasing applies changes from one line of work onto another in the order they were introduced.

**To forward-port local commits to another destination commit:**

1. Select a versioned project, file or folder.
  2. Select **Team > Branch/Tag > Rebase** from the main menu.
- Alternatively, right-click the selected item and select **Git > Branch/Tag > Rebase** from the context menu.
3. Set the required properties in the **Select Commits to Rebase** dialog.
  4. Click **Rebase** to complete rebasing according to the defined settings.

## 3.3 Versioning Applications with Subversion

Subversion is a type of version control system that aids developer groups working with shared source files in common repositories by managing file revision history information.

The IDE's Subversion support enables you to manage changes to version-controlled files as you work. In the IDE, you can call Subversion commands on both files and directories in the Projects, Files, Versioning, and Favorites windows. The IDE also provides a graphical Diff Viewer, enabling you to compare file revisions, as well as a History Viewer, allowing you to search a file or folder's history for versions based on specific criteria.

The IDE's Subversion support allows you to:

- Rename documents in your repository and keep the whole history of the document's revisions.
- Treat commits as whole entities. If you attempt to commit changes to several files and the commit fails, no files are changed in the repository

### 3.3.1 Subversion Visualization Features

The IDE provides several file status information tools that simplify the process of working with version-controlled files, including:

- Color Coding. Enables you to view the current status of version-controlled files.
- Annotations. Enables you to view revision and author information for each line of version-controlled files.

### 3.3.2 Working with Subversion

The following table outlines the basic workflow when working with Subversion in the IDE.

Task	Details
Set Up Subversion	See <a href="#">Section 3.3.5, "How to Set Up Subversion."</a>
Synchronize local files with repository	Check out files from a repository. See <a href="#">Section 3.3.6, "How to Check Out Files From a Remote Repository (Subversion)." </a> Import your project into a Subversion repository. See <a href="#">Section 3.3.7, "How to Place Projects Under Version Control (Subversion)." </a>
Edit Sources	Make changes to local copies of versioned files.  Diff file revisions between repository versions and your local working copies. See <a href="#">Section 3.3.9, "How to Compare File Revisions in Subversion."</a>  Merge changes in repository revisions with the local copies. See <a href="#">Section 3.3.14, "How to Merge File Revisions in Subversion."</a>
Update Local Versions	Update local versions of files with changes committed to the Subversion repository. See <a href="#">Section 3.3.8, "How To Update Files in a Local Working Directory (Subversion)." </a>
Resolve Conflicts	Resolve conflicts between local versions of files with changes committed to Subversion repository revisions. See <a href="#">Section 3.3.15, "Resolving Merge Conflicts in Subversion."</a>
Commit Sources	Commit local changes to files into the repository. See <a href="#">Section 3.3.10, "How to Commit Local Changes to a Remote Repository (Subversion)." </a>

### 3.3.3 How to View File Status Information

The IDE's Subversion support enables you to view and manage the evolution of changes in version-controlled files.

You can view version status information in many of the IDE's windows, including the Versioning, Projects, Files, and Favorites windows. The Versioning window, however, represents the primary place within which to manage version-controlled files by displaying a list of all of the new, modified, and removed files in the currently selected project or directory.

To open the Versioning window, select either:

- **Subversion > Show Changes** from the context menu of a version-controlled file or folder from the Projects, Files, or Favorites window.
- **Team > (Subversion >) Show Changes** from the main menu.
- **Window > Versioning > Subversion** from the main menu.

The IDE's Subversion support provides file status information in the following ways:

Task	Details
Status Labels	Textual indication of file status in the Versioning, Projects, Files, and Favorites windows. To display status labels, select <b>View &gt; Show Versioning Labels</b> from the main menu.
Status Color	Textual indication of file status in the Versioning, Projects, Files, and Favorites windows. To display status labels, select <b>View &gt; Show Versioning Labels</b> from the main menu.
Status Badges	Graphical indication of the status of files contained within your project, folder, and package nodes. Displayed in the Projects, Files, and Favorites windows.
Revision Annotations	Displays commit message, author, date, and revision number information in the left margin of files open in the Source Editor. To display annotations, select a versioned file and select <b>Show Annotations</b> (or <b>Subversion &gt; Show Annotations</b> ) from its context menu. Alternatively, select <b>Versioning &gt; Show Annotations</b> from the main menu.

The IDE displays version-controlled files using the following color coding and font styles:

Color Coding	Description
 Main.java	Indicates that the file is a new local file that does not yet exist in the repository.
 Main.java	Indicates that the file has been modified locally.
 Main.java	Indicates that the file contains conflicts. You must employ the Resolve Conflicts command ( <b>Subversion &gt; Resolve Conflicts</b> ) for such files.
 Main.java	Indicates that the file is ignored by Subversion and will not be included when calling versioning commands. In the Versioning window, grey text signifies deleted files.

Color Coding	Description
Main.java	Indicates that the file is excluded when calling the Commit command. All other Subversion commands, however, work as usual. Note that files displayed in the strike-through style only appear in the Versioning window and Commit dialog. They will not appear in Diff panes, nor will their parent folders (or packages) display badges if they are modified.

Current Subversion file status is indicated by adding the following badges to project, package and directory icons:

Badge	Description
	A blue badge on a folder or package node marks folders or packages that contain locally modified or new files. In the case of packages, this badge applies only to the package itself and not its subpackages. For folders, the badge indicates local modifications in that folder or any of its subfolders.
	A red badge on a folder or package node marks folders or packages that contain files for which the repository copy of the file contains changes which conflict with the local version. In case of packages, this badge applies only to the package itself and not its subpackages. For folders, the badge indicates local modifications in that folder or any of its subfolders.

**Note:** Parent folders (or packages) of files excluded from commits and displayed in the strike-through style will not display badges if they are modified.

### 3.3.4 How to Work with Version Histories

Searching the histories of files can be helpful when you need to find specific commits, for example when backporting bugs. The IDE's Search Histories window enables you to view a summary of a file's evolution over time by revision number.

You can view file histories in the IDE's Search History window. To do so, select a versioned file (for example, from the Projects, Files, or Favorites window) and select either **Subversion > Search History** from the context menu or **Team > (Subversion >) Search History** from the main menu. Alternatively, in the Versioning window you can select Search History from a file's context menu.

#### 3.3.4.1 Searching for Specific Revisions

Using the Search History window, you can search a file or folder's history for versions based on several criteria. These are:

Criteria	Description
Message	The description submitted with the commit.
Username	The author of the commit
From	The tag, revision number, or date from which the commit was made.
To	The tag, revision number, or date before which the commit was made.

#### To search for a specific revision:

1. Select the file or folder for which you want to find a specific revision.

2. Select **Team > (Subversion >) Search History** from the main menu. Alternatively, you can select **Subversion > Search History** from the selected file's context menu. The IDE displays the file's revision history in the Search History window.
3. Enter search criteria information in the Message, Username, From, and To fields. You can also use the Browse buttons to designate any tag, branch, revision, or date you want to limit your search to.
4. Click **Search**. The IDE displays the file's revision history in the Search History window.

---

**Note:** If you want to perform a diff, or revert (that is, 'rollback') changes from a specific commit, you can do so from the displayed revision history using the Diff and Revert links to the right of each revision.

---

### 3.3.4.2 Comparing Revisions in the Search History Window

You can compare a file with previous versions from within the Search Histories window using the **Diff** button.

**To view revision differences in the Search History window:**

1. Click the Diff button in the toolbar.
2. Select the revision against which you want to compare your local working copy. The IDE displays a diff of the file's revisions within the Search History window. Note that you can navigate between the differences using the Previous Difference (  ) and Next Difference (  ) buttons.

### 3.3.5 How to Set Up Subversion

Before you can take advantage of the IDE's Subversion support, you need to have Subversion client software installed on your system. The IDE supports Subversion client versions 1.3.x and higher. The IDE's Subversion support works by interacting with the Subversion client to carry out the commands.

You can download Subversion as a binary package from the following link:

<http://www.collab.net/downloads/subversion>

Though the Subversion site does not guarantee the quality of the binary downloads, you might find them easier to work with. For example, the executable binary for Microsoft Windows systems sets up the environment variable that enables your system and the IDE to recognize the Subversion installation. If you do not use an installer, follow the Subversion installation instructions closely to make sure that you set up everything correctly on your system.

If you are using Mac OS X, you may need to manually register the Subversion executable home folder in the IDE. See Specifying the Path to the Subversion Executable in the Guided Tour of Subversion for more details:

<http://netbeans.org/kb/docs/ide/subversion.html>

After the Subversion client is set up, you can run Subversion commands from the IDE's **Team > Subversion** menu. To check out files from a Subversion repository, select **Team > (Subversion >) Checkout**. In the process of checking out, the IDE automatically registers the working directory where your local copies of version-controlled files and their status information will be stored.

If you have already checked out files from a Subversion repository (using Subversion client 1.3.x or higher), the IDE automatically recognizes the files as versioned files if those files are part of an open IDE project or if they are added to the Favorites window. You can call Subversion commands on such files from the Subversion menu or by right-clicking a file or folder and choosing from the Subversion submenu.

### 3.3.6 How to Check Out Files From a Remote Repository (Subversion)

In order to work on shared files located in a remote repository, you need to copy the necessary files and directories to your local working directory. This is done by checking out the files from the repository.

#### To check out files from a remote repository:

1. Select **Team > (Subversion >) Checkout** from the main menu. The Subversion Checkout wizard opens.
2. In the first panel of the wizard, enter a URL that contains the connection protocol and the location of the repository.
  - If you enter a protocol such as `http://`, `https://`, or `svn://`, User and Password fields appear in the wizard. Fill in these fields as necessary.
  - If you enter `svn+ssh://`, you must supply the command to establish the external tunnel.
  - If you are using a proxy, click the Proxy Configuration button and enter the required information.

Click **Next**.

3. In the Folders to Checkout panel of the wizard, specify the folder that you want to check out in the Repository Folder field or click **Browse** to choose from a list of all folders in the repository.
  - To check out only the contents of the folder you are specifying (rather than the folder itself), select the **Skip "<selected\_folder>" and check out only its content** option.
4. If you need to specify a revision number, enter the information in the Repository Revision field or click the **Browse** button to view and select from a list of all revisions in the repository.
5. Specify the local folder into which you want to check out the selected folders. Alternatively, click the **Browse** button to navigate to the desired directory on your system.
6. Click **Finish** to check out the files. The IDE initiates the checkout action and the IDE's status bar indicates the progress of the files downloading from the repository to your local working directory. You can also view files being checked out from the Output window (**Ctrl+4**).

---

**Note:** Select the **Scan for NetBeans projects after Checkout** option if you intend to immediately work with the checked-out sources. If the sources contain projects, a dialog will appear prompting you to open them in the IDE. If the sources do not contain a project, the dialog will appear prompting you to create a new project from the sources and then open them in the IDE. When you create a new project for such sources, select the appropriate project category and then use the "With Existing Sources" template in that category.

---

### 3.3.7 How to Place Projects Under Version Control (Subversion)

The IDE enables you to place any project you are working on under version control. You effectively import your sources into the remote repository. To do so, you need to be able to access a Subversion repository for which you have write privileges.

#### To place an IDE project under version control:

1. In the Projects window, select an unversioned project and then select either:
  - **Versioning > Import into Subversion Repository** from the node's context menu.
  - **Team > Subversion > Import into Repository** from the IDE's main menu.
 The Subversion Import wizard opens.
2. In the Subversion Repository page of the Import wizard, specify the protocol and location of the Subversion repository as defined by the Subversion URL. Depending on your selection, you may require to specify further settings, such as repository username and password, or, in the case of `svn+ssh://`, you must specify the tunnel command to establish the external tunnel. Click **Next**.
3. In the Repository Folder panel, specify the repository folder in which you want to place the project in the repository. A folder containing the name of your project is suggested for you in the Repository Folder text field by default.
4. In the text area beneath **Specify the Message**, enter a description of the project you are importing into the repository.
5. Click **Finish** to initiate the import, or optionally, click **Next** to continue to a third panel that enables you to preview all files that are prepared for import. From this panel, you can choose to exclude individual files from import, or identify the MIME types of files before importing. Upon clicking **Finish**, the IDE uploads the project files to the repository and the Output window opens to display the progress.

The IDE supports the following Subversion protocol types:

Protocol	Access Method	Example
file	Direct repository access (on local disk)	<code>file:///repository_path[@REV]</code>
http	Access using WebDAV protocol to a Subversion-aware server	<code>http://hostname/repository_path[@REV]</code>
https	Access using HTTP protocol with SSL encryption	<code>https://hostname/repository_path[@REV]</code>
svn	Access using custom protocol to an svnserve server	<code>svn://hostname/repository_path[@REV]</code>
svn+ssh	Access using SVN protocol through an external SSH tunnel	<code>svn+ssh://hostname/repository_path[@REV]</code>

### 3.3.8 How To Update Files in a Local Working Directory (Subversion)

Updating files or folders enables you to incorporate any changes that other developers have committed to the repository since your previous checkout or update.

#### To update a local version of a file or folder:

- Select a versioned file (for example, in the Projects, Files, or Favorites window) and select **Subversion > Update**. Alternatively, select a file in the Versioning

window and select **Update** from the context menu. The IDE incorporates any changes existing in the repository version of the file.

---

**Note:** It is recommended that you perform an update on all files prior to committing them to the repository. Doing so allows you to become aware of any conflicts prior to performing the commit.

---

### 3.3.8.1 Updating Projects with Dependencies

If you are working on a project for which there are other required projects, you can update both the main project and all dependent projects using the IDE's Update with Dependencies command.

**To update an entire project and all dependent projects:**

1. Right-click the project node in the Projects window.
2. Select **Subversion > Update With Dependencies**. The IDE incorporates any changes existing in the repository version of all of the projects with your local working copy.

## 3.3.9 How to Compare File Revisions in Subversion

The Diff command compares different revisions of a file and displays the differences found graphically.

### 3.3.9.1 Comparing File Revisions Graphically

The IDE's graphical Diff Viewer enables you to compare different versions of a file side by side using color coding to emphasize the differences between the files being compared.

The previous ( ) and next ( ) difference buttons in the toolbar enable you to navigate among the differences in the file. You can also refresh the status and update files from within the Diff Viewer using the Refresh Diff ( ) and Update ( ) buttons. When you are finished comparing the files, you can commit your changes using the Diff Viewer's Commit button ( ).

**To generate a graphical diff comparing a repository revision to your working copy:**

- Right-click a versioned file node in the Projects, Files, or Versioning window and select **Diff** (or **Subversion > Diff**). The IDE displays the results in the Diff Viewer in a new tab of the main window.

---

**Note:** If you want to perform a diff on all files contained in a folder, select a folder and select **Subversion > Diff** from the context menu. All files that contain differences between your local version and the repository version will be listed in the upper pane of the Diff Viewer. You can then view individual diffs on files by clicking a file from the list.

---

**To make changes to a file while comparing it in the Diff Viewer:**

1. Navigate through differences between the two versions using the previous difference and next difference arrow icons.
2. Click any of the displayed icons to perform immediate changes to your local copy.

---

**Note:** Your local copy appears in the right pane of the Diff Viewer. You can also make changes to your local copy by typing directly into it in the Diff Viewer.

Any changes made in the Diff Viewer are automatically saved as they occur.

---

The following icons enable you to make changes directly within the Diff Viewer:

- **Replace.** Inserts the highlighted text from the previous revision into the current revision
- **Move All.** Reverts the file's current version to the state of the selected previous version.
- **Remove.** Removes the highlighted text from the current version so that it mirrors the previous version.

### 3.3.10 How to Commit Local Changes to a Remote Repository (Subversion)

Once your working copies of version-controlled files have been edited, you can then place changes into the repository using the Subversion Commit action.

---

**Note:** Subversion versioned files and folders must be recognized by the IDE as such in order to call Subversion actions on them. To do so, you must first check out sources from a Subversion repository.

---

#### To commit changes in local files to a remote repository:

1. Select a version-controlled file or folder (for example, from the Projects, Files, or Favorites window) and select **Subversion > Commit** from the context menu. The Commit Dialog opens, listing all files that contain local changes. If the files you want to commit do not already exist in the repository, the commit action will add them.
2. Enter a commit message in the Commit Message text area, indicating the purpose of the commit.
  - Click the Recent Messages icon in the upper right corner of the dialog to view recent commit messages.
3. Click **Commit**. The IDE executes the commit and sends your local changes to the repository.

---

**Note:** It is recommended that you perform an update (**Subversion > Update**) on any files prior to committing them to the repository. By doing so, you will be able to identify and handle any conflicts arising from repository changes prior to performing your commit.

---

When working in the Commit dialog, you can exclude individual files from a commit. To do so, click on the Commit Action column for the specific file and select **Exclude from Commit**. The file name responds by displaying in strike-through text.

### 3.3.10.1 Ignoring Files

If your local working directory includes files or directories that you do not want to place under version control, you can set the IDE to ignore them permanently using the Ignore command.

---

**Note:** You cannot employ the Ignore command on files that already exist in the repository.

---

#### To ignore local files in your working directory:

1. Select the file or directory you wish to ignore from the Projects, Files, Favorites, or Versioning window.
2. Select **Ignore** (or **Subversion > Ignore**) from the context menu of selected file or directory. The IDE ignores the file or directory whenever Subversion commands are called on it or on the directory within which it is stored.

---

**Note:** To change the status of ignored files so that they can be acted upon in Subversion, select the specific file and select **Subversion > Unignore**.

---

## 3.3.11 How to Work with Branches in Subversion

The IDE's Subversion support enables you to:

- checkout branches from a remote repository.
- create branches in the repository you are working from.
- switch to a branch in the repository you are working from.

For more information on working with branches, see the Subversion documentation at:

<http://svnbook.red-bean.com/>

### 3.3.11.1 Checking Out Branches

If you need to edit files on a branch folder that already exists, you can check out the branch to copy the files to a local working directory. You must however create a new local working directory within which to checkout the branch.

#### To checkout a branch to your local working directory:

1. Select **Team > (Subversion >) Checkout** from the main menu. The Subversion Checkout wizard opens.
2. In the first panel of the wizard, enter a URL that contains the connection protocol and location to the repository. Depending on your protocol selection and connection requirements, enter any required parameters, such as username, password, proxy configuration, etc. Click **Next**.
3. In the Folders to Checkout panel of the wizard, specify the folder that represents the branch you want to check out in the Repository Folder field. You can click **Browse** to choose from a list of all folders in the repository. Specify the revision number if you want to work from a specific revision.

- To check out only the contents of the folder you are specifying (that is, not the folder itself), select the **Skip "<selected\_folder>" and check out only its content** option.
4. Specify the local working directory into which you want to checkout the selected branch folder. Alternatively, click the **Browse** button to navigate to the desired directory on your system.
  5. Click **Finish** to check out the branch folder. The IDE initiates the checkout action and the IDE's status bar indicates the progress of the files downloading from the repository to your local working directory. You can also view files being checked out from the Output window (**Ctrl+4**).

### 3.3.11.2 Creating Branches

If you want to work on a separate version of your file system for stabilization or experimentation purposes, you can do so by creating a branch. To create a branch in Subversion, you are effectively copying a version-controlled project or folder and adding it to a new location within the repository. The IDE's Subversion support enables you to select a source from either your local working copy, or browse to a location in the repository you are working from. You can create branches in the IDE by choosing **Subversion > Copy To** from a versioned file or folder's context menu.

#### To create a branch:

1. Select the versioned project or folder (for example, in the Projects, Files, or Favorites window) that you want to be the root of your branch and select **Subversion > Copy To**. The Subversion Copy dialog box opens.
2. Under Source, select Local Folder if you want to create a branch from your local working copy, otherwise select Remote Folder to specify the version maintained in the repository.
  - You can specify a specific revision of the selected item by entering a revision number in the Revision text box. Clicking the Search button adjacent to the Revision text box allows you to view revisions maintained in the repository.
  - Click the **Skip selected Folder and copy only its Contents** option if you want to avoid including the selected folder when creating a copy.
3. Under Destination, select the target location for the new branch copy. In Repository Location, type in a path relative to the repository, otherwise click the Browse button to open a new dialog that aids in browsing the repository folders.
4. Enter a description for the new branch folder in the Copy Description text area. If you want to switch to the branch after creating it, select the **Switch to Copy** checkbox.
5. Click **Copy**. The IDE creates the branch by copying the folder contents to the specified location in the repository.

### 3.3.11.3 Switching to a Branch

If you want to switch your local working copy to point to a branch or other location in the repository, you can do so using the Subversion Switch to command.

#### To switch to a branch or other location in the repository:

1. Select a versioned file or folder (for example, from the Projects, Files, or Favorites window) and select either **Subversion > Switch to Copy** from the context menu or

**Team > (Subversion >) Switch to** from the IDE's main menu. The Subversion Switch dialog box opens.

2. For Repository Folder, enter the branch folder you want to switch to, or click Browse to view a list of all locations in the repository.
  - If you need to specify a previous revision, enter the revision number for the selected location. Leave blank if you require the most recent revision. To display a log of all revisions for the selected location, you can click the Search button.
3. Click Switch. The IDE updates your working copy to reflect the content maintained by the newly selected branch location in the repository. Note that any differences arising between your local working copy and the location that you are switching to will be overwritten by the new location.

### 3.3.12 How to Revert Modifications (Subversion)

Occasionally it may be desirable or necessary to revert changes made to files in your local working copy. You may need to either revert changes made locally, or retrieve revisions from the repository, and undo any changes to files caused by these revisions. The IDE enables you to do this using the Revert Modifications command.

#### To revert modifications:

1. Select the versioned file for which you want to revert changes (for example, from the Projects, Files, Favorites or Versioning window) and select **Revert Modifications** (or **Subversion > Revert Modifications**). The Revert Modifications dialog box opens, enabling you to specify criteria.  
If you have created files or folders since the point to which you are reverting, you can delete them by selecting the **Remove Newly Added Files and Folders** option.
2. In the Revert Modifications dialog box, select one of three options:
  - **Revert Local Changes.** Reverts any changes made in your local working copy to the previously committed or updated version.
  - **Revert Modifications from Single Commit.** Reverts the local working copy to the repository revision made prior to the previous commit. (Specify a revision number in the Revision text field.)
  - **Revert Modifications from Previous Commits.** Reverts the local working copy to a specified revision from the repository. (Specify starting and ending revision numbers in the corresponding text fields.)
3. Click **Revert**. The IDE reverts any changes in the local file to a state that corresponds with the repository version, according to the specified parameters.

### 3.3.13 How to Recover Deleted Files (Subversion)

When working with version-controlled sources, you may sometimes find it useful to be able to recover locally deleted files from the repository. The IDE's version control support enables you to recover versioned files by performing a revert delete action on files maintained in your local working copy.

#### To revert deletes made to files in your local working copy:

1. Right-click the project or folder that previously contained the deleted files (for example, from the Projects, Files, or Favorites window) and select **Subversion > Show Changes** from the context menu. The Versioning window opens in the

bottom panel of the IDE displaying file changes. The file names of recently deleted files display in grey text with status listed as Locally Deleted.

2. Right-click the file you want to recover and select **Revert Delete**.
3. In the Confirm Overwrite dialog that displays, click Yes to enable the IDE to recover the file from its local history repository. The file disappears from the Versioning window and is relisted in the project directory within your local working copy.

### 3.3.14 How to Merge File Revisions in Subversion

The Merge command is useful for porting changes from a specific repository revision to your local working copy of a file or folder. You can also port the changes made within a given range of revisions to your local working copy.

**To port changes from a revision or range of revisions to your local working copy:**

1. In the Projects, Files, or Favorites window, right-click the files or folders on which you want to perform the merge operation and select **Subversion > Merge Changes**.
2. In the dialog that appears, fill in the following fields:
  - **Merge From.** Select one of the following types of merge:
    - **One Repository Folder.** Port changes from one repository folder.
    - **Two Repository Folders.** Port changes from two repository folders.
    - **One Repository Folder Since Its Origin.** Port changes that have occurred between the time of the folder's creation and the revision number that you specify in the Ending Revision Field.
  - **Repository Folder.** The folder from which you want to port changes. (If you select Two Repository Folders in the Merge From field, this field is replaced by First Repository Folder and Second Repository Folder fields.)
  - **Starting Revision.** If merging from a single revision, enter the revision number. If merging changes that occurred within a range of revisions, enter the starting point of that range.
  - **Ending Revision.** If merging changes that occurred within a range of revisions, enter the ending point of that range.
3. Click Merge. The IDE incorporates any differences found in the selected revision to the local copy of the file. If merge conflicts occur, the file's status is updated to Merge Conflict to indicate this.

---

**Note:** After merging file changes to your local working directory, you must still commit changes using the Commit command in order for them to be added to the repository.

---

### 3.3.15 Resolving Merge Conflicts in Subversion

When merge conflicts occur, a Merge Conflict badge (  ) appears on the parent folder (or package) of file. Within the file itself, each conflict is marked with arrows followed by the lines from the two revisions that caused the conflict.

---

**Note:** Merge conflicts must be resolved prior to checking your local file into the repository.

---

**To resolve merge conflicts graphically with the Merge Conflicts Resolver:**

1. In the Projects, Files, Favorites, or Versioning window, select the file whose status indicates that there is a conflict and select Resolve Conflicts from the menu. The Merge Conflicts Resolver displays with merge conflicts highlighted in red.
2. Use the Next ( ) and Previous ( ) difference buttons in the upper-left corner to navigate to each conflict in the file.
3. For each conflict, click Accept above the pane containing the text that you wish to accept. Once you have chosen the correct text, it is highlighted in green and displayed in the Merge Result pane. The text you did not choose is highlighted in blue.
4. If neither pane contains the text you want, exit the Merge Conflict Resolver and edit your source file manually. When you are done making changes, right-click the file and select **Resolve Conflicts** (or **Subversion > Resolve Conflicts**) from the context menu. Then repeat the procedure, beginning with Step 1.
5. After resolving each conflict, click **OK** to exit the Merge Conflict Resolver. The IDE updates your local working copy with the desired changes.

---

**Note:** Once you have resolved each merge conflict, you still need to commit the file to add your changes to the repository copy.

---

### 3.3.16 How to Create and Apply a Patch (Subversion)

Patch files enable software developers who are not sharing a common repository to distribute and integrate changes that have been made to the code. The IDE enables you to create and apply patches that update copies of source files so that you do not have to incorporate the changes manually.

**To create a patch file:**

1. Select a versioned file (for example, in the Projects, Files, or Favorites window) for which you want to create a patch.
2. Select **Team > (Subversion >) Export Diff Patch** from the main menu. The Export Diff Patch dialog opens.
3. Select one of the following:
  - Save to File. When selected, enter a name for the patch file and specify the location where you want to save the patch.
  - Attach to Task. When selected, select the task repository and specify a task ID and description. The patch file is added as an attachment to the specified task.
4. Click **OK**. A patch file is created containing the differences between the source file versions.

---

**Note:** the Export Diff Patch command is only available on files and folders which contain local or remote changes that have not already been merged.

---

**To apply a patch to a local file or folder:**

1. Select a versioned file or folder (for example, in the Projects, Files, or Favorites window) on which you want to apply the patch.
2. Select **Team > (Subversion >) Apply Diff Patch** from the main menu. The Apply Diff Patch dialog displays.
3. In the dialog, type the path or navigate to the patch file you want to apply.
4. Click **Patch**. The patch is applied to the selected file and a dialog opens, confirming that the patch was applied successfully. Click **Yes** to view changes in the IDE's Diff Viewer.

---

**Note:** Because patches on folders use relative paths to the files within them, folder patches must be applied on the same folder node that the patch was created on to ensure that the patch is applied properly.

If you are uncertain to which file or directory the patch should be applied, you can find the context information in the patch file itself. Patch files generated by the IDE contain the context in which the patch was originally created in the first few lines of the file.

---

### 3.3.17 How to Shelve Changes (Subversion)

Shelving allows to make changes to a project without committing them to Subversion.

**To temporarily set aside some not yet committed changes in a working directory as a patch file:**

1. Select a versioned project, file, or folder.
2. Select **Team > Shelve > Shelve Changes** from the main menu.
3. Specify the name for a patch to be shelved in the **Patch Name** field.
4. Click **Shelve**.

The IDE stashes away the not yet committed changes contained in the selected project, file, or folder into a patch file and reverts the modified files in the working copy (the working copy gets clear of the local changes). The previously modified files become up to date.

## 3.4 Versioning Applications with Mercurial

Mercurial is a fast, lightweight Source Control Management system designed for efficient handling of very large distributed projects. Unlike Subversion, Mercurial works with distributed repositories which are commonly used in many open source projects today and support distributed development without any centralized control.

The IDE's Mercurial Plugin support enables you to manage changes to version-controlled files as you work. In the IDE, you can call Mercurial commands on both files and directories in the Projects, Files and Favorites windows. The IDE also provides a graphical Diff Viewer, enabling you to compare file revisions, as well as supporting inline diff directly in the editor.

The advantages of a distributed revision control system like Mercurial are:

- Better support for distributed teams by removing a centralized bottleneck
- Better scalability with large numbers of concurrent users

- After the initial clone, faster to work with, independent of a user's network infrastructure

### 3.4.1 About Mercurial Visualization Features

The IDE provides several file status information tools that simplify the process of working with version-controlled files, including:

- Color Coding. Enables you to view the current status of version-controlled files.
- Annotations. Enables you to view revision and author information for each line of version-controlled files.

The IDE's Mercurial support is similar in style to the IDE's Subversion support. The main difference is as Mercurial is a distributed revision control system, you typically begin by cloning an external repository to work with. This clone is a complete copy of the repository including the revision history. You can clone this local copy as often as you like, and when you want to you can push your changes back to the original repository provided you have permissions, or export your changes and send them to the owner if you do not.

For further documentation on the Mercurial Plugin support and Mercurial itself, see the following resources:

NetBeans Mercurial Home: <http://wiki.netbeans.org/MercurialVersionControl>

Mercurial Home: <http://mercurial.selenic.com/wiki/>

Understanding Mercurial:

<http://mercurial.selenic.com/wiki/UnderstandingMercurial>

Mercurial Man Pages:

<http://www.selenic.com/mercurial/wiki/index.cgi/ManPages>

### 3.4.2 How to View File Status Information in Mercurial

The IDE's Mercurial support enables you to view and manage the evolution of changes in version-controlled files.

#### 3.4.2.1 Viewing Revision Information

The IDE's Mercurial support enables you to view version status information in many of the IDE's windows, including the Status, Projects, Files, and Favorites windows. The Status Window, however, represents the primary place within which you manage version-controlled files by displaying a list of all of the new, modified, and removed files in the currently selected project or directory.

**To open the Status window, choose either:**

- **Mercurial > Status** from the context menu of a version-controlled file or folder from the Projects, Files, or Favorites window.
- **Team > Mercurial > Status** from the main menu.
- **Window > Versioning > Mercurial** from the main menu.

The IDE's Mercurial support provides file status information in the following ways:

- **Status Labels.** Textual indication of file status in the Status, Projects, Files, and Favorites windows. To display status labels, select **View > Show Versioning Labels** from the main menu.

- **Status Color.** Graphical indication of file status in the Status, Projects, Files, and Favorites windows.
- **Status Badges.** Graphical indication of the status of files contained within your project, folder, and package nodes. Displayed in the Projects, Files, and Favorites windows.
- **Revision Annotations.** Displays commit message, author, date, and revision number information in the left margin of files open in the Source Editor. To display annotations, select a versioned file and select **Show Annotations** (or **Mercurial > Show Annotations**) from its context menu. Alternatively, select **Versioning > Show Annotations** from the main menu.

The IDE displays version-controlled files using the following color coding and font styles:

- Green. Indicates that the file is a new local file that does not yet exist in the repository.
- Blue. Indicates that the file has been modified locally.
- Red. Indicates that the file contains conflicts. You must employ the Resolve Conflicts command (**Mercurial > Resolve Conflicts**) for such files.
- Grey. Indicates that the file is ignored by Mercurial and will not be included when calling versioning commands. In the Status window, grey text signifies deleted files.
- Strike-through. Indicates that the file is excluded when calling the Commit command. All other Mercurial commands, however, work as usual. Note that files displayed in the strike-through style only appear in the Status window and Commit dialog. They will not appear in Diff panes, nor will their parent folders (or packages) display badges if they are modified.

Current Mercurial file status is indicated by adding the following badges to project, package and directory icons:

- Locally Modified Badge. A blue badge on a folder or package node marks folders or packages that contain locally modified or new files. In the case of packages, this badge applies only to the package itself and not its subpackages. For folders, the badge indicates local modifications in that folder or any of its subfolders.
- Conflict Badge. A red badge on a folder or package node marks folders or packages that contain files for which the repository copy of the file contains changes that conflict with the local version. In case of packages, this badge applies only to the package itself and not its subpackages. For folders, the badge indicates local modifications in that folder or any of its subfolders.

The IDE enables you to clone an external Mercurial Repository and to make it available within the IDE. You effectively create a copy or clone of the entire repository to work with in the IDE. To do so, you need to be able to access a Mercurial repository that you have read privileges for.

Note that the parent folders (or packages) of files excluded from commits and displayed in the strike-through style will not display badges if they are modified.

### 3.4.3 How to Set Up Mercurial

Before you can take advantage of the IDE's Mercurial support, you need to have Mercurial client software installed on your system. The IDE supports Mercurial client versions 1.04 and higher. The IDE's Mercurial support works by using the same commands as the Mercurial command line interface.

You can download Mercurial as either sources or as a binary package from the following link:

<http://mercurial.selenic.com/wiki/Download?action=show&redirect=BinaryPackages>

**To set the path to the Mercurial executable file in the IDE:**

1. Select Tools > Options (NetBeans > Preferences on OS X) from the main menu. The **Options** dialog opens.
2. Select the Team icon along the top of the dialog, then click the **Versioning** tab. In the left pane under Versioning Systems, select Mercurial. User-defined options for Mercurial display in the main window of the dialog.
3. In the **Mercurial Executable Path** text field, either type in the path to the executable file or click **Browse** to navigate to it on your system. Note that you need not include the Mercurial executable file in the path.
4. Click **OK**.

After the Mercurial client is set up, you can run Mercurial commands from the IDE's **Team > Mercurial** menu. To clone an external Mercurial repository, select **Team > Mercurial > Clone Other**. In the process of cloning, the IDE automatically imports all of the history and status information for the cloned files.

If you have already cloned a Mercurial repository, the IDE automatically recognizes the files as versioned files if those files are part of an open IDE project or if they are added to the Favorites window. You can call Mercurial commands on such files from the Mercurial menu or by right-clicking a file or folder and choosing from the Mercurial submenu.

### 3.4.4 How to Clone an External Mercurial Repository

The IDE enables you to clone an external Mercurial Repository and to make it available within the IDE. You effectively create a copy or clone of the entire repository to work with in the IDE. To do so, you need to be able to access a Mercurial repository that you have read privileges for.

**To clone a Mercurial Repository:**

1. Select **Team > Mercurial > Clone Other** from the IDE's main menu. The Mercurial Clone Other wizard opens.
2. In the Mercurial Repository panel of the wizard, enter a URL that contains the connection protocol and the location of the repository.
3. Click **Next**.
4. In step 2 of the wizard, Clone External Repository, specify the default pull and push paths to the repository.
5. In step 3 of the wizard, Destination Directory, specify the Parent directory into which you want to place the Clone of the repository. Alternatively, you can click the **Browse** button to navigate to the desired directory on your system.
6. Specify the Clone Name.
7. Click **Finish**. The IDE initiates the clone action and the IDE's status bar indicates the progress of the files downloading from the repository to your local working directory. You can also view files being cloned from the Output window (**Ctrl+4**).

---

**Note:** Select the Scan for NetBeans projects after clone option if you intend to immediately work with the cloned sources.

If the sources contain projects, a dialog will appear prompting you to open them in the IDE.

If the sources do not contain a project, the dialog will appear prompting you to create a new project from the sources and then open them in the IDE. When you create a new project for such sources, select the appropriate project category in the Create Project wizard and then use the Create Project with Existing Sources template in that category.

---

The IDE supports the following Mercurial URLs:

Protocol	Access Method	Example
file	Direct repository access (on local disk)	file:///repository_path
http	Access using WebDAV protocol to a Mercurial-aware server	http://hostname/repository_path
https	Access using HTTP protocol with SSL encryption	https://hostname/repository_path
static-http	Access using HTTP also, albeit slower, allows access to a Mercurial repository where you simply use a web server to publish the .hg directory as static content	static-http://hostname/repository_path
ssh	Access using SSH	ssh://hostname/repository_path

SSH requires an accessible shell account on the destination machine and a copy of Mercurial (hg) in the remote path or specified with as remotecmd.

path is relative to the remote user's home directory by default. Use an extra slash at the start of a path to specify an absolute path: ssh://example.com//tmp/repository

Mercurial does not use its own compression through SSH; the right thing to do is to configure it in your ~/.ssh/config, for example:

- Host \*.mylocalnetwork.example.com  
Compression no
- Host \*  
Compression yes

### 3.4.5 How to Place Projects Under Version Control

The IDE enables you to place any project you are working on under version control. You effectively import your sources into a local Mercurial repository. The repository files are placed under a .hg directory under the project directory.

#### To place an IDE project under version control:

1. In the Projects window, select an unversioned project and select either:

**Versioning > Initialize Mercurial Repository** from the node's context menu.

**Team > Mercurial > Initialize Repository** from the IDE's main menu.

2. Putting the project under Mercurial revision control. The IDE initiates the Mercurial initialize action and the IDE's status bar indicates the progress of the repository creation under your local working directory. You can also view files being added to the repository from the Output window (**Ctrl+4**).  
All the project files are now registered in the repository as Locally New.  
The new files and their status can be viewed by clicking on **Mercurial > Show Changes** from the context menu.
3. Select **Mercurial > Commit** from the project's context menu to commit these project files to the Mercurial repository. The **Commit - [ProjectName]** dialog box opens.
4. Type your message in the **Commit Message** text area, and then click **Commit**.

---

**Note:** The committed files are placed together with the .hg directory in the Mercurial repository directory. The commit details are available in the IDE Output window (**Ctrl+4** on Windows/**Command-4** on OS X).

---

### 3.4.6 How to Use the Mercurial Diff Viewer

The IDE's graphical Diff Viewer enables you to compare revisions of a file side by side using color coding to emphasize the differences between the files being compared.

You access the Diff Viewer in one of the following manners:

- Select a version-controlled file or folder (for example, from the Projects, Files or Favorites window) and select either **Mercurial > Diff** from the context menu or **Team > Mercurial > Diff** from the main menu
- Select the Diff view from the Status Window

The Diff Viewer tool bar provides you with the following functionality:

- Navigate among differences found in the compared files. Choose to view differences as they appear from top to bottom by clicking the appropriate icon:
  - Go to previous difference.
  - Go to next difference.
- Refresh the status of the diff (for example, in the event that changes to your local working directory have been made externally), and update the status of the repository versions of files listed in the diff:
  - Update the working directory from the repository
- Commit any changes contained in your local working directory:
  - Commit the selected item to the repository.

The Diff Viewer provides the following UI components:

---

Blue	Indicates lines that have been changed since the earlier revision.
------	--

Green	Indicates lines that have been added since the earlier revision.
-------	--

Red	Indicates lines that have been removed since the earlier revision.
-----	--

---

The following icons allow you to make changes directly to your local working copy:

---

Replace	Inserts the highlighted text into your Working Tree copy.
Move All	Reverts the whole local Working Tree copy.
Remove	Removes the highlighted text from the local Working Tree copy.

---

### 3.4.7 How to Use the Mercurial Status Window

The Mercurial Status window presents a real-time view of the changes made in your local working copy for selected version-controlled directories. It opens by default in the bottom panel of the IDE, listing added, deleted or modified files.

To open the Status window, select **Window > Versioning > Mercurial** from the main menu. Alternatively, you can select a versioned file or folder (for example, from the Projects, Files, or Favorites window) and either select **Mercurial > Status** from the node's context menu, or select **Team > Status** from the main menu.

By default, the Status window displays a list of all modified files within the selected package or folder. The toolbar label includes the current revision of the working directory and the Mercurial ChangeSet ID. You can click the column headings above the listed files to sort the files by name, status or location.

The Status window toolbar includes buttons that enable you to invoke the most common Mercurial tasks on all files displayed in the list. The following table lists the Mercurial commands available in the toolbar of the Status window:

---

Icon	Name	Function
file	Changes between HEAD and Working Tree	Displays a list of files that are either already staged or only modified/created and not staged yet.
http	Refresh status	Refreshes the status of the selected files. Files displayed in the Status window can be refreshed to reflect any changes that may have been made externally.
https	Diff All	Opens the Diff Viewer providing you with a side-by-side comparison of your local copies and the versions maintained in the repository.
ssh	Update All	Updates the working directory from the repository.
sftp	Update Target	Allows you to specify the revision to update the working directory to.
git	Commit All	Enables you to commit local changes to the repository.

---

You can access other Mercurial commands in the Status window by selecting a table row that corresponds to a modified file, and choosing a command from the context menu. You can, for example, perform the following actions on a file:

- **Open.** Open the file in the Source Editor.
- **Diff.** Launch the Diff Viewer for this file, to compare the local file with that stored in the Repository.
- **Commit.** Launches the Commit Dialog to allow this file to be Committed.

- **Mark as Resolved.** Allows you to mark a file that is in a Conflict state after a Merge as resolved.
- **Show Annotations.** Displays commit message, author, date, and revision number information in the left margin of files open in the Source Editor.
- **Revert Modifications.** Enables you to revert any changes to the copy in your local working directory back to the current version maintained in the repository.
- **Revert Delete.** Enables you to revert any delete actions that you have committed to files in your local working copy. When you invoke the Revert Delete command, the specified file will be retrieved from the repository and reinstated into your local working copy.
- **Exclude from Commit.** Allows you to mark the file to be excluded when performing a commit.

---

**Note:** You can also exclude files from a commit in the Commit dialog box.

You can select multiple files in the Status window by holding down the **Ctrl** key when you select files.

---

### 3.4.8 How to Merge File Revisions

NetBeans IDE enables you to merge changes between repository revisions and your local working copy. Specifically, this combines two separate changesets in a repository into a new changeset that describes how they combine.

**To merge file revisions:**

1. In the Projects, Files, or Favorites window, right-click the files or folders on which you want to perform the merge operation and select **Mercurial > Branch/Tag > Merge Changes**. The **Merge with Revision** dialog displays.
2. In the **Choose From Revisions** drop-down list, select the revision.  
You are porting all changes made on a local working copy file from the time it was created.
3. Ensure the Description, Author, and Date data are correct.
4. Click **Merge**.

The IDE incorporates any differences found between the repository revisions and your local copy of the file. If merge conflicts occur, the file's status is updated to Merge Conflict to indicate this.

---

**Note:** After merging revisions to your local working copy, you must still commit changes using the Commit command in order for them to be added to the repository.

---

### 3.4.9 How to Switch Branches in the Repository

The Switch Branch In Repository dialog box enables you to switch the branch that your working directory is on to a branch that already exists in your repository.

**To switch a branch:**

1. In the **Choose from Branches** list, specify the branch you want to switch to.

2. (Optional) Specify your filtering criteria in the **Filter** text field to filter the displayed branches.
3. Review information specific to the branch in the following fields:
  - **Author:** Name of a person who committed the revision.
  - **Date:** Date when a commit was made.
  - **Description:** A message specified during the commit of the revision.
4. (Optional) Select the **Do a Forced Update** option to switch to a selected branch even if your local changes are lost.
5. Click **Switch**.

---

**Note:** The name of the branch you have switched to is displayed in a label that follows the project name. Select **View > Show Versioning Labels** from the main menu to display a label.

---

### 3.4.10 How to Commit Changes to the Repository

Mercurial versioned files and folders must be recognized by the IDE as such in order to call Mercurial actions on them. To do so, you must first clone sources from an external repository or place the project under Mercurial control.

It is a good idea to update any copies you have against the repository prior to performing a commit in order to ensure that conflicts do not arise.

**To perform an update on sources that you have modified:**

- Select **Team > Update** from the main menu.

Once your working copies of version-controlled files have been edited, you can then place changes into the repository using the Mercurial Commit action. If you have an task repository set up with your source code repository, the commit dialog box lets you associate your commit action with an existing task.

**To commit changes in local files to the repository:**

1. Select a version-controlled file or folder (for example, from the Projects, Files, or Favorites window) and select **Mercurial > Commit** from the context menu. The Commit Dialog opens, listing all files that contain local changes. If the files you want to commit do not already exist in the repository, the commit action will add them.
2. Enter a commit message in the Commit Message text area, indicating the purpose of the commit.
  - Click the Recent Messages icon ( ) in the upper right corner of the dialog to view recent commit messages.
3. Click **Commit**. The IDE executes the commit and sends your local changes to the repository. You can also view files being committed to the repository from the Output window (**Ctrl+4**).

---

**Note:** When working in the Commit dialog, you can exclude individual files from a commit. To do so, click on the **Commit Action** column for the specific file and select **Exclude from Commit**. The file name responds by displaying in strike-through text.

---

### 3.4.10.1 Updating Tasks

You can update a task associating your commit action with an existing task in your repository's task repository. To do so, click on the Update Task heading in the Commit dialog box to expand it, then specify the following:

Element	Description
Task Repository	Specify the task repository that your source code repository uses, by selecting a task repository from the drop-down list. The drop-down provides you with a list of all task repositories registered with the IDE. If your source code repository's task repository is not registered, click the New button to register it.
Task	Specify the task ID. You can do this by typing in the ID, or part of the description.

You can also specify the following options:

Element	Description
Resolve as FIXED:	When selected, the status of the task is marked as Resolved.
Add Commit Message from Above	When selected, the commit message is added to the task.
Add Revision Information to the Task	When selected, the task is updated to include the revision information such as the author, date, etc. You can click Change Format to modify the format of the revision information that is added to the task.
Add Task Information to Commit Message	When selected, the task ID and summary are added to the commit message. You can click Change Format to modify the format of the task information that is added to the message.
After Commit	When selected, the task is updated after you commit the changes.
After Push	When selected, the task is updated only after the changes are pushed to the source code repository.

### 3.4.10.2 Ignoring Files

If your local working directory includes files or directories that you do not want to place under version control, you can set the IDE to ignore them permanently using the Ignore command.

You cannot employ the Ignore command on files that already exist in the repository.

#### To ignore local files in your working directory:

1. Select the file or directory you wish to ignore from the Projects, Files, Favorites, or Status window.
2. Select **Ignore** (or **Mercurial > Ignore**) from the context menu of selected file or directory. The IDE ignores the file or directory whenever Mercurial commands are called on it or on the directory within which it is stored.

To change the status of ignored files so that they can be acted upon in Mercurial, select the specific file and select **Mercurial > Unignore**.

### 3.4.10.3 Pushing Local Changes to the Shared Repository

Before pushing changes that you have committed locally to the shared repository, you need to synchronize your local repository with the shared repository.

**To synchronize your local repository with the shared repository:**

- Select **Team > Remote > Fetch** from the main menu.

After you perform a successful Fetch, your local repository becomes synchronized with the shared repository.

**To push changes:**

- Select **Team > Remote > Push Current Branch**, **Team > Remote > Push All Branches**, or **Team > Remote > Push** from the main menu.

The output from a successful Push will list any changesets created.

---

**Note:** Since you maintain a copy of the entire repository on your system, the general practice is to make multiple commits to your local repository and only after the particular task is complete, perform the push to the shared repository.

---

**3.4.11 How to Set Mercurial Project Properties**

Open the Mercurial Properties Dialog in one of the following ways:

- Select **Team > Mercurial > Properties** from the IDE's main menu
- Select **Mercurial > Properties** from the context menu of a Mercurial-versioned project

The following table describes the Mercurial properties that can be set:

Element	Description
default-pull	Specify the local repository path to Pull changes from when choosing <b>Mercurial &gt; "Pull from - default"</b> from the Mercurial controlled project's context menu.
default-push	Specify the local repository path to Push changes to when choosing <b>Mercurial &gt; "Push to - default"</b> from the Mercurial controlled project's context menu.
username	Set the username to record when committing changes to the repository. This property overrides the global Mercurial User Name Option for this project.

---

**Note:** To Push to other external locations select **Team > Mercurial > Share > Push Other**.

---

To Pull from other external locations select **Team > Mercurial > Share > Pull Other**.

---

**3.4.12 How to Set Mercurial Global Options**

Open the Mercurial Options window:

- Select **Tools > Options**. The IDE's Options window opens.
- Select the **Team > Versioning** tabs, then select Mercurial under Versioning Systems.

The following list describes the Mercurial Global Options property that can be set.

Element	Description
Mercurial User Name	Set the username to record when committing changes to the repository. May be overridden for a given project using the Mercurial Project Properties username setting.
Mercurial Executable Path	Set the executable path to the installed Mercurial Hg executable. You only need to use this if Mercurial has been placed somewhere other than your standard PATH.
Default Export Filename	Set the export filename used when using <b>Team &gt; Export Diff Patch</b> from the IDE's main menu when a Mercurial-versioned project is selected. The following identifiers can be used in the filename: %b: Project Directory name %r: Current revision number being exported. %h: Changeset ID being exported.
Mercurial Status Labels	Set the Status labels to be displayed in the Project view for a Mercurial controlled project. To enable, select <b>View &gt; Show Versioning Labels</b> from the IDE's main menu. The following labels can be used: {status}: Status of the file. {revision}: Revision of the file. {folder}: Branch or Tag Name.
Mercurial Extensions	Opens a dialog to allow you to add Mercurial extensions. Currently, the NetBeans IDE adds the hgk extension to support the <b>Mercurial &gt; View</b> from the a Mercurial project's context menu. You can change its location from this dialog if necessary.

**Note:** Select the **Automatically open Output window** option to set the Output window to display when versioning operations that involve interaction with the repository take place (for example, clone, fetch, pull, push). This option is selected by default.

### 3.4.13 How to Enable Support for Mercurial Queues

Since Mercurial Queues is an extension, you must explicitly enable it before you can use it.

#### To start running Mercurial Queues on your Mercurial repository:

1. On your system, browse to the .hgrc file. The default location of the .hgrc file on OS X and Linux is \$HOME/.hgrc, on Windows - %USERPROFILE%\Mercurial.ini.
2. Add the following lines to your .hgrc file:

```
[extensions]
hgext.mq =
```

3. Save the file.

### 3.4.14 How to Create a Patch

#### To create a patch:

1. In the Projects window, select a versioned project and select **Team > Queues > Create Patch** (alternatively, right-click the project name and select **Mercurial > Queues > Create Patch** from the context menu).

- The Create Patch dialog box displays.
2. Specify the name for a patch in the Patch Name text field.
  3. Provide the description of a patch in the Patch Message field.
- Alternatively, select either of the following options and click OK afterwards:
- click **Recent Messages** to select a message from a list of most recent commit messages
  - click **Load Template** to select a message template for a commit message

---

**Note:** The provided description of a patch will be used as a commit message when turning the patch into a permanent changeset.

---

4. Select files to be included into the patch in the Files To Include In Patch table.
5. (Optional) Specify information pertinent to a task related to the created patch using the fields of the Update Task area.
6. Click Create Patch.

A new patch is created and added to the .hg/patches directory.

### 3.4.15 How to Refresh a Patch With Local Modifications

To save your progress into the patch you are creating, you need to complete the following steps:

1. Select **Team > Queues > Refresh Patch** from the main menu (alternatively, right-click the project name and select **Mercurial > Queues > Refresh Patch** from the context menu).

The Refresh Patch dialog box displays.

2. Provide the description of a patch in the **Patch Message** field.

Alternatively, select either of the following options and click **OK** afterwards:

- click the **Recent Messages** button to select a message from a list of most recent commit messages
- click the **Load Template** button to select a message template for a commit message

---

**Note:** The provided description of a patch will be used as a commit message when turning the patch into a permanent changeset.

---

3. Select files to be included into the patch in the **Files To Include In Patch** table.
4. (Optional) In the **Update Task** area, specify information pertinent to a task related to the changes being committed.
5. Click **Refresh Patch**.

The patch you are working on is updated.

### 3.4.16 How to Compare Patch Revisions

**To generate a patch for review and compare revisions of a patch side by side**

1. Select **Team > Queues > Diff** from the main menu (alternatively, right-click the project name and select **Mercurial > Queues > Diff** from the pop-up menu).  
The Diff Viewer displays differences found in the current patch and all your uncommitted/unrefreshed local changes in side-by-side panels
2. Review and revise differences found in the compared files using either Graphical or Textual Diff Viewer.

---

**Note:** The Graphical Diff Viewer highlights changes in the files using the following color encoding.

---

Element	Description
Blue	Indicates lines that have been changed since the earlier patch.
Green	Indicates lines that have been added since the earlier patch.
Red	Indicates lines that have been removed since the earlier patch.

The following icons enable you to make changes directly within the Graphical Diff Viewer.

Element	Description
Replace	Inserts the highlighted text from the previous patch into the current patch.
Replace All	Reverts current version of a patch to the state of its selected previous version.
Remove	Removes the highlighted text from the current version of a patch so that it mirrors the previous version of a patch.

### 3.4.17 How to Switch Between Patches

**To switch to a particular patch in a patch queue series:**

---

**Note:** To switch between patches there *must* be *no* local modifications in the working copy, otherwise the switch fails.

---

1. Select **Team > Queues > Go To Patch** from the main menu (alternatively, right-click the project name and select **Mercurial > Queues > Go To Patch** from the context menu).

The **Go To Patch** dialog box displays a list of all patches available in a stack.

---

**Note:** Names of applied patches display in bold.

---

---

**Note:** Select **Team > Queues > Pop All Patches** to remove the applied patches from the top of the stack and update the working directory to undo the effects of the applied patches.

---

2. Select the required patch and click **Go**.

The IDE applies the changes contained in the selected patch to the chosen project, file, or folder.

### 3.4.18 How to Finish Applied Patches

Once your work on a patch is done, it can be turned into a permanent changeset.

**To turn all applied patches in a patch queue series into regular changesets:**

1. Select **Team > Queues > Finish Patches** from the main menu (alternatively, right-click the project name and select **Mercurial > Queues > Finish Patches** from the context menu).

The **Finish Patches** dialog box displays.

2. Select the name of a patch to be finished in the patches field.

---

**Note:** All patches in the series before the selected patch will also be finished.

---

3. Click **Finish Patches**.

The IDE turns all applied patches up to the selected patch into regular changesets.

### 3.4.19 How to Shelve Changes (Mercurial)

Shelving allows to make changes to a project without committing and pushing them to Mercurial.

**To temporarily set aside some not yet committed changes in a working directory as a patch file:**

1. Select a versioned project, file, or folder.
2. Select **Team > Shelve > Shelve Changes** from the main menu.
3. Specify the name for a patch to be shelved in the **Patch Name** field.
4. Leave the **Back up any Locally Modified files to <file>.orig** option selected to get all local changes stored in a `<file>.orig` file directly in the working copy (if required).
5. Leave the **Remove Newly Added Files and Folders** option selected to get all newly added files and folders deleted (if required).
6. Click **Shelve**.

---

**Note:** After a patch with all the modifications is created and shelved, the local, not yet committed, modifications are reverted in the working copy (that is the working copy gets clear of the local changes). The previously modified files become up to date.

---

### 3.4.20 How to Rebase (Mercurial)

Sometimes it is required to move revisions from a point to another.

**To append the private changes on top of any destination commit:**

1. Select a versioned project, file or folder in the Projects window.
2. Select **Team > Branch/Tag > Rebase** from the main menu.  
Alternatively, right-click the selected item and choose **Mercurial > Branch/Tag > Rebase** from the context menu.
3. Set the required properties in the **Select Changesets to Rebase** dialog box.
4. Click **Rebase** to complete rebasing according to the defined settings.

## 3.5 Versioning Applications with CVS

CVS (Concurrent Version System) is a type of version control system that aids developer groups working with shared source files in common repositories by managing file revision history information.

The IDE's CVS built-in support enables you to manage changes to version-controlled files as you work. In the IDE, you can call CVS commands on both files and directories in the Projects, Files, Versioning, and Favorites windows. For an overview of the typical workflow when using CVS in the IDE, see [Section 3.5.1, "How to Work with CVS."](#).

### 3.5.1 How to Work with CVS

The following table outlines the basic workflow when working with CVS in the IDE.

Element	Description
Set up CVS	<a href="#">Set up CVS. See Section 3.5.5, "How to Set Up CVS."</a>
Synchronize local files with repository	<a href="#">Check out files from a repository. See Section 3.5.7, "How to Check Out Files from a Remote Repository (CVS)."</a> <a href="#">Import your project into a CVS repository.</a>
Edit sources	<a href="#">Make changes to local copies of versioned files.</a> <a href="#">Diff file revisions between repository versions and your local working copies. See Section 3.5.16, "How to Resolve Merge Conflicts."</a> <a href="#">Merge changes in repository revisions with the local copies. See Section 3.5.15, "How to Merge File Revisions from a Branch (CVS)."</a>
Update local versions	<a href="#">Update local versions of files with changes committed to the CVS repository. See Section 3.5.8, "How to Update Files in a Local Working Directory (CVS)."</a>
Resolve conflicts	<a href="#">Resolve conflicts between local versions of files with changes committed to CVS repository revisions. See Section 3.5.9, "How to Compare File Revisions in CVS."</a>
Commit sources	<a href="#">Commit local changes to files into the repository. See Section 3.5.10, "How to Commit Local Changes to a Remote Repository (CVS)."</a>

### 3.5.2 How to Use CVS with the IDE

The IDE's CVS support enables you to manage the evolution of changes in version-controlled files by displaying status information directly in the IDE's various

windows (for example, Projects, Files, Favorites, Versioning) so that you can call CVS commands as you work.

The Versioning window presents a real-time view of the changes in selected directories. It opens at the bottom of the IDE whenever you view changes made in version-controlled files. By default, the Versioning window displays a list of all modified files within the selected package or folder.

You can open the Versioning window by selecting **Team > (CVS >) Show Changes** from the main menu. Alternatively, you can right-click any versioned project, directory, or file and select **CVS > Show Changes**.

Using the filter buttons in the toolbar, you can limit the list of displayed files to either locally or remotely modified files. The Versioning window toolbar also includes buttons that enable you to invoke the most common CVS tasks on all files displayed in the list, such as committing, updating, refreshing status, and performing diffs.

The following CVS commands are available in the Versioning window's toolbar:

Element	Description
Refresh status	Refreshes the status of the selected files and folders. Files displayed in the Versioning window can be refreshed to reflect any changes that may have been made externally.
Diff all	Opens the Diff Viewer providing you with a side-by-side comparison of your local copies and the versions maintained in the repository.
Update all	Updates all selected files from the repository.
Update local versions	Update local versions of files with changes committed to the CVS repository.
Commit All	Enables you to commit local changes to the repository.

You can call CVS commands on versioned-controlled files (those existing within a registered working directory) using the CVS menu in the IDE's main window. In addition, many CVS commands are available in the context-sensitive menu for a selected file or directory.

You can access the CVS commands context-sensitive menu by right-clicking a file or directory in any of the following:

- Projects window
- Files window
- Favorites window
- Versioning window
- History Viewer
- Source Editor (that is, from the tab of an open file)

---

**Note:** The IDE's status bar, located in the bottom right of the interface, displays the current command's status.

---

### 3.5.3 How to View File Status Information (CVS)

The IDE's CVS support enables you to view and manage the evolution of changes in version-controlled files.

### 3.5.3.1 Viewing Revision Information

The IDE's CVS support enables you to view version status information in many of the IDE's windows, including the Versioning, Projects, Files, and Favorites windows. The Versioning window, however, represents the primary place within which to manage version-controlled files by displaying a list of all of the new, modified, and removed files in the currently selected project or directory.

To open the Versioning window, choose either:

- **CVS > Show Changes** from the context menu of a version-controlled file or folder from the Projects, Files, or Favorites window.
- **Team > (CVS >) Show Changes** from the main menu.
- **Window > Versioning > CVS** from the main menu.

The IDE's CVS support provides file status information in the following ways:

Element	Description
Status Labels	Textual indication of file status in the Versioning, Projects, Files, and Favorites windows. To display status labels, select <b>View &gt; Show Versioning Labels</b> from the main menu.
Status Color.	Graphical indication of file status in the Versioning, Projects, Files, and Favorites windows.
Status Badges	Graphical indication of the status of files contained within your project, folder, and package nodes. Displayed in the Projects, Files, and Favorites windows.
Revision Annotations	Displays commit message, author, date, and revision number information in the left margin of files open in the Source Editor. To display annotations, select a versioned file and select <b>Show Annotations</b> (or <b>CVS &gt; Show Annotations</b> ) from its context menu. Alternatively, select <b>Versioning &gt; Show Annotations</b> from the main menu.

The IDE displays version-controlled files using the following color coding and font styles:

Element	Description
Green	Indicates that the file is a new local file that does not yet exist in the repository.
Blue	Indicates that the file has been modified locally.
Red	Indicates that the file contains conflicts. You must employ the Resolve Conflicts command ( <b>CVS &gt; Resolve Conflicts</b> ) for such files.
Grey	Indicates that the file is ignored by CVS and will not be included when calling versioning commands. In the Versioning window, grey text signifies deleted files.
Strike-through	Indicates that the file is excluded when calling the Commit command. All other CVS commands, however, work as usual. Note that files displayed in the strike-through style only appear in the Versioning window and Commit dialog. They will not appear in Diff panes, nor will their parent folders (or packages) display badges if they are modified.

Current CVS file status is indicated by adding the following badges to project, package and directory icons:

Element	Description
Locally Modified Badge	A blue badge on a folder or package node marks folders or packages that contain locally modified or new files. In the case of packages, this badge applies only to the package itself and not its subpackages. For folders, the badge indicates local modifications in that folder or any of its subfolders.
Conflict Badge	A red badge on a folder or package node marks folders or packages that contain files for which the repository copy of the file contains changes which conflict with the local version. In case of packages, this badge applies only to the package itself and not its subpackages. For folders, the badge indicates local modifications in that folder or any of its subfolders

### 3.5.4 How to Work with Version Histories

Searching the histories of files can be helpful when you need to find specific commits, when backporting bugs for example. The IDE's Search Histories window enables you to view a summary of a file's evolution over time by revision number.

You can view file histories in the IDE's Search History window. To do so, select a versioned file (for example, from the Projects, Files, or Favorites window) and select either **CVS > Search History** from the context menu or **Team > (CVS >) Search History** from the main menu. Alternatively, in the Versioning window you can select **Search History** from a file's context menu.

Using the Search History window, you can search a file or folder's history for versions based on several criteria. The Search Histories window enables you to search for changes in version controlled files by:

Element	Description
Message	The description submitted with the commit.
Username	The author of the commit.
From Tag	The tag, revision number, or date from which the commit was made.
To Tag	The tag, revision number, or date before which the commit was made.

#### To search for a specific revision:

1. Select the file or folder for which you want to find a specific revision.
2. Select **CVS > Search Histories** from the context menu. The IDE opens the Search History window to display the selected item's revision history.
3. Enter search criteria information in the Message, Name, From, and To fields. You can also use the **Browse** buttons to designate any tag, branch, revision, or date you want to limit your search to.
4. Click **Search**. The IDE displays the file's revision history in the Search History window.

---

**Note:** If you want to view other files that were included in a specific commit, you can broaden your search using the "*your\_project*" and Open Projects links to the right of each revision.

---

### 3.5.4.1 Comparing Revisions in the Search History Window

You can compare a file with previous versions from within the Search Histories window using the Diff button.

#### To view revision differences in the Search History window:

1. Click the Diff button in the toolbar.
2. Select the revision against which you want to compare your local working copy. The IDE displays a diff of the file's revisions within the Search History window. Note that you can navigate between the differences using the Previous Difference and Next Difference buttons.

## 3.5.5 How to Set Up CVS

In the IDE, no special setup is necessary in order to use CVS. You need only to check out the files located in a remote repository to a local working directory on your system. In the process of checking out, the IDE automatically registers the working directory where your local copies of version-controlled files and their status information will be stored.

If you have previously checked out files to a local working directory on your system using another CVS executable or the command line, you can still work with the files using the IDE's CVS support. The IDE automatically recognizes the files as versioned files, enabling you to call CVS commands on them. See [Checking Out Files from a Remote Repository](#) for more information.

---

**Note:** If you want to set up a local repository, you must do so manually using a CVS command line client. For information, see the CVS documentation at <http://ximbiot.com/cvs/manual/>.

---

## 3.5.6 How to Adjust CVS Settings in the IDE

The IDE enables you to make changes to CVS settings, such as:

- applying arbitrary metadata to files or directories by adding, removing, or modifying the status label format of versioning labels for versioned files
- adjusting CVS settings in the IDE, such as excluding new files from commit actions automatically
- limiting the number of characters per line in commit messages

#### To change the status label format for versioned files:

- In the Status Label Format text box, enter the format in which you want the status labels to be displayed. You can enter your own label, otherwise specify a CVS-reserved label by clicking **Add Variable** and selecting from the available options.

---

**Note:** In order to make versioning labels visible, select **View** from the main menu, then select the **Show Versioning Labels** option. Versioning labels display in light grey text to the immediate right of versioned files in the Projects, Files, Favorites windows.

---

**To automatically exclude new files from commit actions:**

- Select the **Apply "Exclude from Commit" On New Files Automatically** option. When selected, newly created files will have to be manually included in commit actions.

**To limit the number of characters per line in commit messages:**

1. Select the **Wrap Commit Messages To (characters)** option.
2. In the adjacent text box, enter the number of characters to indicate the length of each line. Commit messages longer than the specified number of characters will be wrapped to multiple lines.

### 3.5.7 How to Check Out Files from a Remote Repository (CVS)

In order to work on shared files located in a remote repository, you need to copy the necessary files and directories to your local working directory. This is done by checking out the files from the repository.

**To check out files from a remote repository:**

1. Select **Team > (CVS >) Checkout** from the main menu.
2. On the first page of the CVS Checkout wizard, specify the location of the CVS repository by choosing it from the CVS Root drop-down menu. If you are unfamiliar with the syntax, you can click the Edit button and enter the required information using the dialog. For information on supported CVS protocols, view supported CVS root types below.
3. Enter your password in the Password field.
4. If you are using a proxy, click the Proxy Configuration button and enter the required information. Click **Next**.
5. On the Module to Checkout page, specify the modules you want to check out in the Module field or click the Browse button to choose from a list of all modules in the repository. Note that if you do not specify a module, the entire repository will be checked out.
6. If you need to specify a specific branch, revision number, or tag for the checkout, enter the information in the Branch field or click the Browse button to choose from a list of all branches in the repository.
7. Specify the local working directory into which you want to check out the selected modules or branches. Alternatively, you can click the Browse button to navigate to the desired directory on your system.
8. Click **Finish** to check out the files. The IDE initiates the check out command and the IDE's Status bar, displayed in the lower right corner of the IDE, indicates the progress of the files downloading from the repository to your local working directory.

If the checked-out sources contain a project, a dialog will appear prompting you to open them in the IDE. If the sources do not contain a project, the dialog will appear prompting you to create a new project from the sources and then open them in the IDE.

The IDE directly supports the following CVS root types:

Element	Description
pserver	For connecting to pserver repositories. :pserver:username@hostname:/repository_path
ext	For connecting to CVS repositories using SSH, etc. :ext:username@hostname:/repository_path.

The following root types require an external CVS executable:

Element	Description
fork	For protocol access to local CVS repository :fork:/repository_path
local	For direct access to local CVS repository :local:/repository_path

### 3.5.8 How to Update Files in a Local Working Directory (CVS)

Updating files or folders enables you to incorporate any changes that other developers have committed to the repository since your previous checkout or update.

#### To update a local version of a file or folder:

- Select a versioned file (for example, in the Projects, Files, or Favorites window) and select **CVS > Update**. Alternatively, select a file in the Versioning window and select **Update** from the context menu. The IDE incorporates any changes existing in the repository version of the file.

---

**Note:** It is recommended that you perform an update on all files prior to committing them to the repository. Doing so allows you to become aware of any conflicts prior to performing the commit.

---

#### 3.5.8.1 Updating Projects with Dependencies

If you are working on a project for which there are other required projects, you can update both the main project and all dependent projects using the IDE's Update with Dependencies command.

#### To update an entire project and all dependent projects:

1. Right-click the project node in the Projects window.
2. Select **CVS > Update With Dependencies**. The IDE incorporates any changes existing in the repository version of all of the projects with your local working copy.

### 3.5.9 How to Compare File Revisions in CVS

The Diff command compares different revisions of a file and displays the differences found graphically.

The IDE's graphical Diff Viewer enables you to compare different versions of a file side by side using color coding to emphasize the differences between the files being compared.

The previous and next difference buttons in the toolbar enable you to navigate among the differences in the file. You can also refresh the status and update files from within the Diff Viewer using the Refresh Diff and Update buttons. When you are finished

comparing the files, you can commit your changes using the Diff Viewer's Commit button.

**To generate a graphical diff comparing a repository revision to your working copy:**

- Right-click a versioned file node in the Projects, Files, or Versioning window and select **Diff** (or **CVS > Diff**). The IDE displays the results in the Diff Viewer in a new tab of the main window.

If you want to perform a diff on all files contained in a folder, select a folder and then select **CVS > Diff** from the context menu. All files that contain differences between your local version and the repository version will be listed in the upper pane of the Diff Viewer. You can then view individual diffs on files by clicking a file from the list.

**To make changes to a file while comparing it in the Diff Viewer:**

1. Navigate through differences between the two versions using the previous difference and next difference arrow icons.
2. Click any of the displayed icons to perform immediate changes to your local copy.

---

**Note:** Your local copy appears in the right pane of the Diff Viewer. You can also make changes to your local copy by typing directly into it in the Diff Viewer.

Any changes made in the Diff Viewer are automatically saved as they occur.

---

The following icons enable you to make changes directly within the Diff Viewer:

Element	Description
Replace	Inserts the highlighted text from the previous revision into the current revision.
Move All	Reverts the file's current version to the state of the selected previous version.
Remove	Removes the highlighted text from the current version so that it mirrors the previous version.

### 3.5.10 How to Commit Local Changes to a Remote Repository (CVS)

Once your working copies of version-controlled files have been edited, you can then place your changes into the repository using the Commit command.

**Note:** Project directories must be located within a local working directory that is registered with the IDE in order to call CVS commands on them.

**To commit changes in local files to a remote repository:**

1. In the Versioning window, confirm that you want to commit the listed files.
2. Ensure that your local copies of the files are up-to-date by clicking the **Update All** button prior to committing your changes.
3. Once the Update command has finished, commit the files by clicking the **Commit All** button. If the files you want to commit do not already exist in the repository, the Commit command will add them.

4. Enter a message describing your changes in the CVS Commit dialog that appears and click **Commit**. The IDE executes the Commit command and the selected files are committed to the repository.

Click the **Recent Messages** icon in the upper right corner of the dialog to view recent commit messages.

---

**Note:** You can also commit files or directories from the Projects window, Files window, or Favorites window by right-clicking the files or directories you wish to commit and choosing **CVS > Commit**.

---

### 3.5.10.1 Excluding Files from a Commit

The IDE enables you to specify which files in your local working directory are recognized when using CVS commands. Using the **Exclude from Commit** command, you can set the IDE to temporarily disregard files when making commits.

#### To exclude local files in your working directory:

1. Select the files or directories you wish to ignore in the Versioning window, Projects window, Files window, or Favorites window.
2. Right-click and select the **CVS > Exclude from Commit** option. If you have the Versioning window opened, note that the IDE displays the file or directory name in strike-through text and disregards the item whenever the commit command is called. Note that the files remain excluded until you include them.

To remove the exclude status from a file or directory, right-click the selected file or directory, select **CVS** and deselect the **Exclude from Commit** option.

### 3.5.10.2 Ignoring Files

If your local working directory includes files or directories that you do not want to place under version control, you can set the IDE to ignore them permanently using the **Ignore** command.

#### To ignore local files in your working directory:

1. Select the files or directories you wish to ignore in the Versioning window, Projects window, Files window, or Favorites window.
2. Right-click and select **CVS > Ignore**. Note that the **Ignore** command is unavailable for files that already exist in the repository. The IDE ignores the file or directory whenever CVS commands are called on them or the directories within which they are stored.

## 3.5.11 How to Revert Modifications (CVS)

Occasionally it may be desirable or necessary to revert local file changes to their current status maintained in the repository. The IDE enables you to do this using the **Revert Modifications** command.

#### To revert modifications:

1. Select the versioned file for which you want to revert changes (for example, from the Projects, Files, Favorites or Versioning window) and select **Revert Modifications** (or **CVS > Revert Modifications**) from the context menu. A Confirm Overwrite dialog displays notifying you that your local copy is about to be overwritten with the current repository version.

2. Click Yes to initiate the process. Your selected file will be overwritten with the repository version and changes are automatically reflected if the file is already opened in the IDE's editor.

### 3.5.12 How to Recover Deleted Files (CVS)

When working with version-controlled sources, you may sometimes find it useful to be able to recover locally deleted files from the repository. The IDE's version control support enables you to recover versioned files by performing a revert delete action on files maintained in your local working copy.

**To revert deletes made to files in your local working copy:**

1. Select the project or folder that previously contained the deleted files (for example, from the Projects, Files or Favorites window) and select **CVS > Show Changes** from the context menu. The Versioning window opens in the bottom panel of the IDE displaying file changes. The file names of recently deleted files display in grey text with status listed as Locally Deleted.
2. Right-click the file you want to recover and select **Revert Delete**.
3. In the Confirm Overwrite dialog that displays, click **Yes** to enable the IDE to recover the file from its local history repository. The file disappears from the Versioning window and is relisted in the project directory within your local working copy.

### 3.5.13 How to Work with Tags

It is often helpful to tag version controlled files for reference purposes, when you want to mark the point a particular branch was made, for example. Using the CVS Tag command you can attach an existing tag to a file or folder or create a new tag.

**To apply a tag to a file or directory:**

1. Right-click the file or folder and select **CVS > Tag**.
2. Enter the tag name in the text field. Alternatively, click **Browse** to select from a list of existing tags.  
If the tag you entered does not already exist, a new tag will be created.
3. If you want to ensure that the tag is not attached to local revisions that are not up to date, select the **Avoid Tagging Locally Modified Files** checkbox.
4. If you want to move an existing tag to the current revision in your local working directory, select the **Move Existing Tags** checkbox. The IDE attaches the tag on the specified file or directory.

### 3.5.14 How to Work with Branches in CVS

The IDE's CVS support enables you to maintain different versions of an entire code base using branches. Branches are created by attaching a branch tag to a file or folder in the repository. When you modify files in a branch it does not affect the files in the main code line or "trunk". You can, however, merge any modifications you make in the branch back to the trunk at a later stage.

### 3.5.14.1 Checking out Branches

If you need to edit files on a branch that already exists, you can check out the branch to copy the files to a local working directory. You must, however, create a new local working directory within which to checkout the branch.

**To check out a branch to a local working directory:**

1. Select **Team > (CVS >) Checkout** from the main menu.
2. On the first page of the CVS Checkout wizard, specify the location of the CVS repository by choosing a preconfigured URL from the CVS Root drop-down menu. If you are unfamiliar with the syntax, click **Edit** and enter the required information in the Edit CVS Root dialog.
3. Enter your repository password in the Password field.
4. If you are using a proxy, click the **Proxy Configuration** button and enter the required information. Click **Next**.
5. On the Module to Checkout page, specify the files and directories you want to check out in the Module field or click the Browse button to choose from a list of all modules in the repository.
6. Specify the branch, revision number, or tag to checkout in the Branch field or click the Browse button to choose from a list of all branches in the repository.
7. Specify the local working directory into which you want to check out the selected branches. Alternatively, you can click the Browse button to navigate to the desired directory on your system.
8. Click **Finish** to check out the files. The IDE initiates the checkout command for the branch. The IDE's Status bar, displaying in the lower right corner of the IDE, indicates the progress of the files downloading from the repository branch to your local working directory.

### 3.5.14.2 Switching to a Branch

If you want to switch a project, directory, or file to a branch that already exists for stabilization or experimentation purposes, you can do so using the **Switch to Branch** command.

**To switch to a branch:**

1. In the Projects, Files, or Versioning window, right-click the file or folder you want to be the root of your branch and select **CVS > Switch to Branch**.
2. In the Switch to Branch dialog, enter a branch name or click **Browse** to see a list of available branches.
3. Click **Switch**. The IDE moves the selected file to the specified branch by adding a branch tag.

### 3.5.14.3 Creating a Branch

If you want to work on a separate version of your file system for stabilization or experimentation purposes, you can do so by creating a branch. Branches are created by adding a branch tag to a project, directory, or file revision.

**To create a branch:**

1. In the Projects, Files, or Versioning window, right-click the file or folder you want to be the root of your branch and select **CVS > Branch**.

2. In the Branch dialog, enter a Branch name or click Browse to see a list of available branches.
3. If you want to tag the file or directory, select the Tag Before Branching checkbox and enter a Tag name.
4. If you want to switch to the new branch after creating it, select the Switch to This Branch Afterwards checkbox.
5. Click Branch. The IDE creates the branch by adding a branch tag to the selected file or directory.

---

**Note:** When you add a branch tag, the branch is created in the repository immediately and does not require using the Commit command.

---

### 3.5.15 How to Merge File Revisions from a Branch (CVS)

The Merge command is useful for incorporating changes made on different branches of the repository with your local working copy of a file or directory. Any changes you make on one branch can be merged to another branch, including back to the trunk. You can also merge the difference between two earlier revisions with your local working copy.

#### To merge a file from a branch with your local working copy:

1. In the Projects, Files, Favorites, or Versioning window, select the files or folders that you want to merge into your local working copy and select **CVS > Merge Changes from Branch**.
2. In the dialog that appears, select the branch you want to merge the changes from. If you want to merge changes from a branch other than the trunk, enter the branch name or click **Browse** to see a list of available branches.
3. If you want to merge changes after a specific tag, select the **Merge Only Changes Made after Tag** checkbox and designate the tag in the Tag Name field. Alternatively, you can click **Browse** to choose from a list of available tags.
4. If you want to tag the merge, select the **Tag Trunk after Merge** (or **Tag branch\_name after Merge**) checkbox and enter the desired tag name. Alternatively, click **Browse** to choose from a list of available tags.
5. Click Merge. The IDE incorporates any differences found in the branch version into your local copy of the file. If merge conflicts occur, the file's status is updated to Merge Conflict to indicate this.

After merging file changes from a branch to your local working directory, you must still commit the changes using the Commit command in order for them to be added to the repository.

### 3.5.16 How to Resolve Merge Conflicts

When merge conflicts occur, a Merge Conflict badge appears on the parent folder (or package) of file. Within the file itself, each conflict is marked with arrows followed by the lines from the two revisions that caused the conflict.

---

**Note:** Merge conflicts must be resolved prior to checking your local file into the repository.

---

**To resolve merge conflicts graphically with the Merge Conflicts Resolver:**

1. Right-click the node of the file whose status indicates that there is a conflict and select **Resolve Conflicts** from the pop-up menu. The Merge Conflicts Resolver is displayed with merge conflicts highlighted in red.
2. Use the **Next** and **Previous** difference buttons in the upper-left corner to navigate to each conflict in the file.
3. For each conflict, click **Accept** above the pane containing the text that you wish to accept. Once you have chosen the correct text, it is highlighted in green and displayed in the Merge Result pane. The text you did not choose is highlighted in blue.
4. If neither pane contains the text you want, exit the Merge Conflict Resolver and edit your source file manually. When you are done making changes, right-click the node of the file and choose **CVS > Update** from the context menu. Then repeat the procedure, beginning from Step 1.
5. After resolving each conflict, click **OK** to exit the Merge Conflict Resolver. The IDE updates your local working copy with the desired changes.

---

**Note:** You can also resolve conflicts manually in the Source Editor. To do so, open the file in the editor. For each conflict in the file, delete the arrows and text that you do not want, then save the file. The IDE updates your local working copy with the desired changes.

Once you have resolved each merge conflict, you still need to commit the file to add your changes to the repository copy.

---

### 3.5.17 How to Create and Apply a Patch

Patch files enable software developers who are not sharing a common repository to distribute and integrate changes that have been made to the code. The IDE enables you to create and apply patches that update copies of source files so that you do not have to incorporate the changes manually.

**To create a patch file:**

1. Select a versioned file (for example, in the Projects, Files, or Favorites window) for which you want to create a patch.
2. Select **Team > (CVS >) Export Diff Patch** from the main menu. The Export Diff Patch dialog opens.
3. In the dialog, enter a name for the patch file and specify the location where you want to save the patch.
4. Click **Export**. A patch file is created in the specified location containing the differences between the source file versions.

---

**Note:** The Export Diff Patch command is only available on files and folders which contain local or remote changes that have not already been merged.

---

**To apply a patch to a local file or folder:**

1. Select a versioned file or folder (for example, in the Projects, Files, or Favorites window) on which you want to apply the patch.

2. Select **Team > (CVS >) Apply Diff Patch** from the main menu. The **Apply Diff Patch** dialog displays.
3. In the dialog, type the path or navigate to the patch file you want to apply.
4. Click **Patch**. The patch is applied to the selected file and a dialog opens, confirming that the patch was applied successfully. Click **Yes** to view changes in the IDE's **Diff Viewer**.

---

**Note:** Because patches on folders use relative paths to the files within them, folder patches must be applied on the same folder node that the patch was created on to ensure that the patch is applied properly.

If you are uncertain to which file or directory the patch should be applied, you can find the context information in the patch file itself. Patch files generated by the IDE contain the context in which the patch was originally created in the first few lines of the file.

---

## 3.6 About Local History

The IDE provides built-in versioning support for local projects and files. Similar to conventional versioning systems, the IDE maintains an internal history of recent changes made to sources in your local working directory. Whenever you save a file, this is registered as a 'commit' of a new version of the file in the local history. You can inspect all revisions in the local history and diff them against the current files.

### 3.6.1 About the IDE's Local History Tools

The IDE provides a Graphical Diff Viewer. Enables you to select previously saved versions of a file and compare differences between versions graphically.

### 3.6.2 How to Browse Local File History

The IDE's local history functionality automatically maintains previous versions of your project sources. You can use local history tools to do the following:

- create versions of a file
- compare two versions side-by-side
- revert your current file to a previous version
- recover previously deleted files

Each time you make changes to, then save (**Ctrl+S**) a file, the IDE automatically creates a new version of that file.

#### To examine previous versions for a specific file:

1. Select the file from the Projects or Files window and select **History > Show History**. A graphical Diff Viewer opens in the main window.
2. Select a version from the table displayed in the upper area of the graphical diff viewer. Note that versions are chronologically ordered. Upon selecting a version, the file automatically displays on the left side in the Diff Viewer below.

---

**Note:** Select a version from the table displayed in the upper area of the graphical diff viewer. Versions are chronologically ordered. Upon selecting a version, the file automatically displays on the left side in the Diff Viewer below.

---

### 3.6.3 How to Recover Deleted Files

The IDE's local history repository maintains previous versions of your project sources, including files or folders which you have recently deleted. You can take advantage of the IDE's local history tools to recover previously deleted files and reinstate them into your project.

If you have previously deleted files and they are still maintained in the IDE's local history repository, you can invoke the Revert Deleted command to retrieve them.

**To recover previously deleted files:**

- From the Projects or Files window, right-click the project or directory that previously contained the files which you deleted and select **History > Revert Deleted**. The previously deleted files reappear beneath the project or folder you selected.

---

**Note:** In order to recover deleted files using the Revert Deleted command, files must not be created or deleted externally.

---

### 3.6.4 How to Revert a File to a Previous Version

The IDE's local history functionality enables you to revert your current file to a previous version. You do this by using the graphical Diff Viewer.

If you want to examine a previous version and revert only specific changes in your current file to areas within that version, you can perform a side-by-side comparison between the two versions in the Diff Viewer, and revert changes in your current file based on the comparison.

**To revert changes in a file using the Diff Viewer:**

1. From the Projects or Files window, right-click the file you want to revert changes to and select **History > Show History**. The graphical Diff Viewer opens.
2. Select a version from the table displayed in the upper area of the viewer. The selected version displays on the left side in the Diff Viewer below and is compared against your current version.
3. After examining differences between the two versions, use the icons displayed in the Diff Viewer to make changes:
  - Replace. Inserts the highlighted text from the previous revision into the current revision
  - Move All. Reverts the file's current version to the state of the selected previous version.
  - Remove. Removes the highlighted text from the current version so that it mirrors the previous version.

Any changes made using the Diff Viewer's icons are immediately applied to the file's current version.

You can make changes to your current version directly in the graphical Diff Viewer. To do so, place the cursor directly in the Current Version pane and edit where appropriate. The current version is instantaneously updated to reflect changes.

### 3.6.5 How to Adjust Local History Settings

The IDE's local file history support currently enables you to adjust the duration of time for which files are locally stored on your system.

**To adjust the local history storage duration:**

1. Select Tools > Options from the main menu.
2. Select the Team category along the top of the Options window.
3. Under Versioning Systems in the left column select History.
4. In the corresponding text field, enter an integer for the number of days you want the IDE to retain local history versions.



# 4

---

## Working in a Collaborative Environment

This chapter describes how to use the collaborative tools that are included in the IDE.

This chapter contains the following sections:

- [About Working in a Collaborative Environment](#)
- [Working with Tasks](#)
- [Working with the Tasks Window](#)

### 4.1 About Working in a Collaborative Environment

Developers who are collaborating on a project require a set of tools and an infrastructure that can help them stay connected to each other and work together as a team. In addition to sharing sources, team members need to be able to share information and communicate with each other, and how they share information depends on the type of information they need to share.

In a collaborative environment, team members have different roles and requirements. For example, in addition to software developers, a team might also include people in the following roles:

- Quality assurance
- Project management
- Documentation
- User experience design
- Marketing

Not all team members use the same tools, but communication between members can be simplified when the infrastructure and tools are integrated. The IDE provides integrated support for the following collaborative tools and services:

**Issue Tracking.** (Bugzilla, JIRA) An issue tracker enables developers and users to report and track tasks that are associated with a project and provides a valuable feedback mechanism for people involved in the project. The integration of an issue tracking system in the IDE enables developers to find, view and resolve project tasks from within the IDE.

**Version Control Systems.** (Subversion, Mercurial, Git) The IDE provides integrated support for version control systems to help developers manage the history of file revisions. For more information about using version control systems in the IDE, see [Section 3, "Versioning Applications with Version Control."](#)

**Team Server.** A Team Server can provide an infrastructure of services for each project hosted on the designated server. After you log in to a registered Team Server, you can open and create projects and access many of the available services from the IDE.

To register and use a Team Server in the IDE, you need to install the Team plugin from the Update Center. For more about using the Plugins manager to install the Team plugin, see [Section 2.9, "Managing Plugins in the IDE."](#)

## 4.2 Working with Tasks

You can organize issues that are recorded on a registered issue tracker as tasks in the IDE. To work with tasks in the IDE you first need to specify the issue tracker that is used as the task repository for your project. After you register a task repository with the IDE you can use the Tasks window to perform the following tasks in the IDE:

- Find, update and resolve tasks
- Create new tasks
- Organize tasks by category
- Create and save queries

### 4.2.1 About Task Repositories

A task repository is a system for tracking issues that are submitted against a project. The IDE supports two types of task repositories:

- **Local.** The IDE includes a local task repository that you can use to store personal tasks. Tasks in your local repository are only stored on your local file system and are only accessible to you from within the IDE. You store scheduling details about tasks in your local repository.
- **Remote.** A remote task repository is generally located on a remote server and is accessible to other users. You can use a remote repository to submit tasks and assign responsibility for resolving tasks to members of a team collaborating on a project.

Remote task repositories typically use an issue tracking system. The IDE provides support for the Bugzilla and JIRA issue tracking systems. For more details about the supported issue tracking systems, see the following sites

- Bugzilla: <http://www.bugzilla.org/>
- Atlassian JIRA: <http://www.atlassian.com/software/jira/overview>

---

**Note:** JIRA support requires the JIRA plugin available from the NetBeans Update Center. For more about details about adding support for JIRA, see [Section 4.2.4, "How to Add Support for JIRA."](#)

For more on the JIRA plugin, see the following FAQ:

<http://wiki.netbeans.org/FaqHowToJira>

---

### 4.2.2 How to Work with Tasks

You can use the Tasks window in the IDE to find, update and create tasks on a remote repository or your local repository.

#### 4.2.2.1 Finding and Opening Tasks

From the Tasks window you can perform a quick search for tasks by id or a string in the summary or open the Find Tasks form to create an advanced query. You can view a list of tasks that match your saved queries in the Tasks window.

After you save a query the results of the search are listed under the query name in the Tasks window. You can double-click any task in the list to open the task form in a new window. You do not need to be online to open a task that is listed under a saved query. You can update a task in the and save the changes when you are offline and then submit the changes the next time that you are online.

##### To perform a quick search of tasks:

1. Choose **Window > Tasks** from the main menu to open the Tasks window.

Alternatively, choose **Team > Find Tasks** to open the Find Tasks form and select the repository from the drop-down list. The drop-down list contains all remote task repositories that are registered with the IDE.

2. In the Repositories section of the Tasks window, click the **Search task in repository** icon (  ) for the repository that you want to search.

3. Type a task id or string in the dialog box.

When you type in the text field the dialog box displays a list of possible matches that is based on tasks that you recently viewed in the IDE.

4. Select a task from the drop-down list.

You can choose Search online Task Repository in the drop-down list to retrieve more results.

5. Click **Open**.

When you click Open the task form opens in a window in the IDE.

#### 4.2.2.2 Creating and Saving Task Queries

The IDE enables you to save and name search queries that you use repeatedly. You can create and save queries using the Find Tasks form or create a query as a URL. If you are not online you can open tasks that are listed in the Tasks window. You can also update a task when you are not online and submit the changes later when you are online again.

##### To create and save a task query:

1. Choose **Window > Tasks** from the main menu to open the Tasks window.

2. In the Repositories section, click the **Create New Query** icon (  ) for the repository to open the Find Tasks form.

Alternatively, you can choose **Team > Find Tasks** in the main menu to open the Find Tasks form

3. Specify the search criteria.

4. Click **Search** to retrieve the results of the query.

When you click **Search** the IDE searches the remote repository and displays the search results in the form.

5. Click **Save Changes** in the search form.

6. Type a name for the query. Click **Save**.

After you save the query the new named query is added under the repository node in the Tasks window. You can expand the named query node to view a list of tasks that meet the search criteria. You can double-click any task in the list to open the task form in a new window.

You can open and update tasks in the list when you are not online. If you are not online the Submit Changes button is disabled in the task form. You can click the Save Changes button to save any updates that you make to the task and submit the changes when you are online.

#### 4.2.2.3 Reporting New Tasks

To create a new task from the IDE you need to use your local repository or register a remote task repository with the IDE and then use the Report a New Task form to specify the details of the task. If you are not logged in to the task repository you will be prompted to supply the log in details when you submit the new task. If you are not online you can create and save the task and then submit the task when you are online.

1. Open the Tasks window and click the **Create New Task** icon (  ) for the repository in the Repositories section.

Alternatively, choose **Team > Report Task** from the main menu and choose a repository in the Report a New Task form.
2. Specify the details of the new task.
3. Click **Submit Task** to submit a task to a remote repository or choose **File > Save** from the main menu to save a task to your local repository.

If you are not online you can click Save Changes and submit the task when you are online. You can expand the Unsubmitted Tasks node in the Tasks window to view a list of the tasks that are not submitted. When you are online you can right-click an unsubmitted task in the list and choose Submit in the popup menu.

#### 4.2.2.4 Updating and Resolving Tasks

The IDE enables you to resolve tasks from within the IDE when you are logged in to a registered remote task repository. You can also update a task to add scheduling details. Scheduling details for the task are only stored in your local repository.

##### To update or resolve a task:

1. Open the task form in the IDE.
2. Type your comment in the New Comment field.
3. Select a new status for the task in the Status drop-down list, if applicable.

If you want to resolve the task, select **Resolved** in the drop-down list and select the appropriate entry in the Resolution drop-down list.
4. Expand the My Private Task Details section and enter a Due Date and Schedule Date, if applicable.
5. Click **Submit Changes** to submit changes to a remote repository or choose **File > Save** from the main menu to save the changes to the task to your local repository.

If you are not logged in to the remote task repository you will be prompted to log in to submit your changes. If you are not online you can click Save Changes and submit the changes when you are online.

### 4.2.3 How to Add a Task Repository

To use an issue tracker with the IDE you need to register the issue tracker as a task repository. After the task repository is registered you can use tools in the IDE to find, report and resolve tasks that are recorded in the task repository.

**To add a task repository:**

1. Choose **Window > Tasks** from the main menu to open the Tasks window.
2. Click the **Add Repository** icon ( ) in the Repositories section.
3. Specify the connection details. Click **OK**.

When you click **OK**, a node for the task repository is added below the Task Repositories node in the Services window.

**To modify the connection properties of a task repository:**

1. Choose **Window > Tasks** from the main menu to open the Tasks window.
2. Right-click the repository node ( ) in the Repositories section and choose **Properties** in the popup menu.
3. Modify the connection details. Click **OK**.

### 4.2.4 How to Add Support for JIRA

To enable integrated support for JIRA issue trackers in the IDE you need to install the JIRA plugin. You can install the JIRA plugin from the NetBeans Update Center. You need to restart the IDE to complete installation of the plugin.

If the JIRA plugin is not installed and you try to view or work with tasks that are tracked using JIRA, the IDE prompts you to use the Plugins manager to install the JIRA plugin.

**To install the JIRA plugin from the Create Issue Tracker dialog box:**

1. Choose **Window > Tasks** from the main menu to open the Tasks window.
2. Click the **Add Repository** icon ( ) in the Repositories section to open the Create Task Repository dialog box.
3. Select **JIRA** in the Connector drop-down list.
4. Click **Download JIRA plugin**.
5. Proceed through the installation process to install the plugin and restart the IDE.

**To install the JIRA plugin using the Plugins Manager:**

1. Open the Plugins manager by choosing **Tools > Plugins** from the main menu.
2. Select the **Available Plugins** tab.
3. Locate the JIRA plugin from the list of available plugins.

You can use the Search field in the Plugins manager to help you locate the plugin.

4. Select the **Install** checkbox for the JIRA plugin.
5. Click **Install**.

6. Proceed through the installation process to install the plugin and restart the IDE.

For more on how to install the JIRA plugin, see the following FAQ.

<http://wiki.netbeans.org/FaqHowToJira>

For more information on features of the JIRA issue tracking system, see the JIRA website:

<http://www.atlassian.com/software/jira/>

## 4.3 Working with the Tasks Window

The Tasks window provides an organized overview of tasks that are recorded in a task repository. To use the Tasks window you can use the local task repository or register a remote task repository with the IDE. You can add a remote task repository in the Services window.

The Categories section of the Tasks window displays lists of tasks that are organized by category. The Repositories section of the Tasks window displays a list of all tasks that are the results of a saved query. You can create new categories and queries from the Tasks window.

### 4.3.1 How to View Tasks

The Tasks window enables you to see lists of organized tasks. You can organize tasks by assigning a task to a category. You can also save queries and view the results of the query in the Tasks window. You can double-click any task entry in a list to open the task in a new window in the IDE.

You can enter a string in the filter text field in the Tasks window to limit the tasks that are displayed to the tasks that contain the string in the task summary.

#### To view tasks that are organized by category:

1. Choose **Window > Tasks** from the main menu to open the Tasks window.
2. Expand a category node in the Categories section to see a list of tasks that you assigned to that category.
3. Double-click a task in the list to open the task in a window in the IDE.

You can move your cursor over a task entry to view a summary of the task.

#### To view tasks that are the result of a saved query:

1. Choose **Window > Tasks** from the main menu to open the Tasks window.
2. Expand the task repository node in the Repositories section of the Tasks window to see a list of saved queries.
3. Expand a saved query to view a list of all tasks that are the result of that query

The entries for tasks that are new or were modified since the last time you opened the Tasks window are displayed in green text.

4. Double-click a task in the list to open the task in a window in the IDE.

You can move your cursor over a task entry to view a summary of the task.

#### To search for a task:

1. Choose **Window > Tasks** from the main menu to open the Tasks window.
2. Click the **Search Task in Repository** icon () for the repository that you want to search.

3. In the Search Task in Repository dialog box, type the task id or part of the task summary text, or select from recent tasks in the drop-down list.
4. (Optional) Select a category from the drop-down list if you want to simultaneously assign the task to your category. Click **Open**.

When you click Open the task opens in a window in the IDE.

### 4.3.2 How to Organize Tasks

You can use custom categories to group tasks in the Tasks window. After you create a custom category you can assign any task to that category and the task remains in that category until you explicitly remove it or you add it to a different category. A task can only be in one custom category. By default, the Categories section displays all tasks that are assigned to a custom category regardless of the status of the task.

The Categories section contains three default Schedule categories that group the tasks that have scheduling details (Today, This Week, All) and one default category for tasks that were opened recently. To hide a Schedule category, click the filter icon ( ) at the top of the Tasks window and disable the category in the list. To hide the tasks that are resolved, click the Filter icon and disable **Show finished tasks in categories**.

#### To organize tasks by custom category:

1. Choose **Window > Tasks** from the main menu to open the Tasks window.
2. Click the **Create Category** icon ( ) in the Tasks window to open the New Category dialog box.
3. Enter a name for the category in the dialog box. Click **OK**.

When you click OK a node for the new category is added under the Categories section in the Tasks window.

4. Open a task in the IDE.
5. Click **Add to Category** at the top of the task window.
6. Select a category from the drop-down list. Click **OK**.

When you click **OK**, the task is added to the list of tasks under the category node.

You can also right-click a task entry in the Repositories section and choose **Set Category** to assign the task to a category.

#### To remove a task from a custom category:

1. Choose **Window > Tasks** from the main menu to open the Tasks window.
2. Expand a category node in the Categories section to see the list of tasks that are grouped in that category.
3. Right-click the task entry that you want to remove and choose **Remove from Category** in the popup menu.

You can remove a task from a category by assigning the task to a different category.

To remove all completed tasks from a category, click the **Remove all finished tasks from categories** icon ( ).

#### To add schedule details to a task:

1. Open the task form in the IDE

You can add scheduling information to tasks in your local or the remote repository, however the scheduling information is only stored in your local repository.

2. Expand the My Private Task Details section.
3. Enter a Due Date and choose a Schedule Date option in the drop-down list.
4. Choose **File > Save** from the main menu.

Tasks that have scheduling details are automatically added to one or more of the default Schedule categories.

### 4.3.3 How to Configure Tasks Window Settings

You can use the Options window in the IDE to configure how often tasks are updated in the Tasks window.

**To configure settings for the Tasks window:**

1. Select the **Tasks** tab in the Team category of the Options window.
2. Specify how often the IDE checks the task repository to update the list of tasks in the Tasks window.
3. Specify the number of tasks that are displayed in each section of the Tasks window. Click **Apply**.

---

# Working with NetBeans Modules

This chapter describes the basics of working with NetBeans Modules.

This chapter contains the following sections:

- [About NetBeans Modules](#)
- [About the NetBeans Platform](#)
- [Working with NetBeans Modules](#)
- [Working with Rich-Client Applications](#)
- [Module and Rich-Client Application Tasks: Quick Reference](#)
- [Setting Up Modules](#)
- [Using the NetBeans APIs](#)
- [Bundling Supporting Items](#)
- [Registering Modules](#)
- [Communicating Between Modules](#)
- [Building Modules](#)
- [Trying Out a Module](#)
- [About Distributing Modules](#)
- [Branding a Rich-Client Application](#)
- [Distributing Rich-Client Applications](#)

## 5.1 About NetBeans Modules

A NetBeans module is a Java archive file which contains Java classes written to interact with the NetBeans APIs. For more information, see [Section 5.7, "Using the NetBeans APIs"](#).

A module identifies itself as a module by an entry in its MANIFEST.MF file. NetBeans modules are packaged as NBM files (.nbm extension) for non-installer distribution, usually via the Plugins manager under the **Tools** menu.

NetBeans modules are written with one of two aims in mind:

- **Extending the IDE** - You can very easily generate skeleton code for extending the IDE's functionality with new features. For example, you can use the skeleton code to write modules that make your favorite cutting-edge technologies available to the NetBeans IDE. Or, if you miss some functionality in the IDE, you can add it

yourself, by using the skeleton code to write a module that provides the desired functionality.

- **Building a rich-client application** - You can use the core of the IDE as a platform on top of which you develop standalone desktop applications. The core of the IDE is a separate product called the NetBeans Platform. For more information, see [Section 5.2, "About the NetBeans Platform"](#). By basing your application on the NetBeans Platform, you can save a lot of development time, because you can reuse the platform's existing features such as menus, toolbars, and windowing systems.

Even though it is a separate product, there is little need to download the NetBeans Platform separately—you can develop the rich-client application in the IDE and then exclude the modules that are specific to the IDE but that are superfluous to your application. Only when you want to use a different version of the platform than is included in the IDE, does it make sense to download the NetBeans Platform and install the modules that define the application into it.

## 5.2 About the NetBeans Platform

The NetBeans Platform provides an application's common requirements—such as menus, document management, and settings—right out of the box. Building an application "on top of NetBeans" means that, instead of writing applications from scratch, you only provide the parts of your application that the NetBeans Platform doesn't already have. At the end of the development cycle, you bundle your application with the NetBeans Platform, saving you time and energy and resulting in a solid, reliable application.

## 5.3 Working with NetBeans Modules

A NetBeans module is a set of Java classes written to interact with the NetBeans APIs, for extending the IDE or for creating your own application on the NetBeans Platform. For more information, see [Section 5.2, "About the NetBeans Platform"](#) and [Section 5.7, "Using the NetBeans APIs"](#).

The following table outlines the development cycle of NetBeans modules, from creation to distribution.

Step	Description
1. Set up the module	<ol style="list-style-type: none"><li>1. Begin creating your module by using the module project template. For more information, see <a href="#">Section 5.6.2, "How to Create a Module Project"</a>.</li><li>2. Optionally, if the module will consist of a collection of module projects, use the module suite project template. For more information, see <a href="#">Section 5.6.4, "How to Create a Module Suite Project"</a>. For example, the collection may make use of external JAR files. If this is the case, put the external JAR files on the module's classpath by using one or more library wrapper module project templates. For more information, see <a href="#">Section 5.6.3, "How to Create a Library Wrapper Module Project"</a>.</li></ol>

Step	Description
2. Develop the module	<ol style="list-style-type: none"> <li data-bbox="833 223 1462 283">1. Right-click a module project in the Projects window and choose <b>New &gt; Other</b>.</li> <li data-bbox="833 297 1462 403">2. In the New File wizard, choose the best NetBeans API template for your programming needs. For more information, see <a href="#">Section 5.7.1, "Generating Skeleton API Implementations"</a>.</li> <li data-bbox="833 418 1462 498">3. After using a wizard, double-click the file that you would like to edit. Use the Source Editor to edit the file.</li> <li data-bbox="833 513 1258 530">4. Refer to the NetBeans API Javadoc.</li> </ol>
3. Build the module	Choose <b>Build &gt; Build Main Project</b> or right-click any project and choose <b>Build Project</b> .
4. Try out the module	Right-click a module project node and choose Run (later Reload in Target Platform) or Install/Reload in Development IDE to try out the module. For more information, see <a href="#">Section 5.12, "Trying Out a Module"</a> .
5. Distribute the module	<ol style="list-style-type: none"> <li data-bbox="833 741 1462 825">1. Right-click the module project and choose <b>Create NBM</b>. For more information, see <a href="#">Section 5.11.2, "How to Build an NBM File"</a>.</li> <li data-bbox="878 840 1139 857">An NBM file is created.</li> <li data-bbox="833 872 1462 931">2. Distribute the NBM file for installation via the Plugins manager.</li> </ol>

## 5.4 Working with Rich-Client Applications

A rich-client application is a complete, functioning, standalone Swing application, built on top of the NetBeans Platform. For more information, see [Section 5.2, "About the NetBeans Platform"](#).

The following table outlines the development cycle of rich-client applications, from creation to distribution.

To perform this task:	Follow these steps:
1. Set up the application.	<ol style="list-style-type: none"> <li data-bbox="763 1313 1339 1431">1. Begin creating your application by using the NetBeans Platform application project template. For more information, see <a href="#">Section 5.6.5, "How to Create a NetBeans Platform Application Project"</a>.</li> <li data-bbox="763 1446 1339 1516">2. Optionally, before going further, brand the application by providing a splash screen and other external customizations.</li> </ol>

To perform this task:	Follow these steps:
2. Develop the application.	<ol style="list-style-type: none"><li>1. Begin creating each distinct part of your application by using the module project template. For more information, see <a href="#">Section 5.6.2, "How to Create a Module Project"</a>.</li><li>2. Right-click a module project in the Projects window and choose <b>New &gt; Other</b>.</li><li>3. In the New File wizard, choose the best NetBeans API template for your programming needs. For more information, see <a href="#">Section 5.7.1, "Generating Skeleton API Implementations"</a>.</li><li>4. After using a wizard, double-click the file that you would like to edit. Use the Source Editor to edit the file.</li><li>5. Refer to the NetBeans API Javadoc.</li></ol>
3. Build the application.	Choose <b>Build &gt; Build Main Project</b> or right-click any project and choose <b>Build Project</b> .
4. Try out the application.	Right-click the application project node and choose <b>Run Project</b> .
5. Brand the application.	<ul style="list-style-type: none"><li>▪ Right-click the application node in the Projects window and choose <b>Properties</b>. Use the Application panel to brand the name of the application launcher.</li><li>▪ Right-click the application node again, choose <b>Branding</b>, and then use the tabs Basic, Splash Screen, Window System, and Resource Bundles to brand these details of the application.</li></ul>

To perform this task:	Follow these steps:
6. Distribute the application.	<p>You can distribute the application in any of the following ways:</p> <ul style="list-style-type: none"><li>▪ Create a ZIP Distribution<ul style="list-style-type: none"><li>▪ Right-click the application project and choose <b>Package as   ZIP Distribution</b>. For more information, see <a href="#">Section 5.15.1, "How to Build a ZIP Distribution"</a>. The application's executable and its clusters are packaged in a ZIP distribution.</li><li>▪ Distribute the ZIP file.</li></ul></li><li>▪ Create a JNLP Application<ul style="list-style-type: none"><li>▪ Right-click the application project and choose <b>JNLP   Build</b>. For more information, see <a href="#">Section 5.15.2, "How to Build a JNLP Application"</a>. The application's JNLP files are created.</li><li>▪ Put the JNLP application on a server.</li></ul></li><li>▪ Create an installer<ul style="list-style-type: none"><li>▪ Right-click the application project and choose <b>Project Properties</b>. In the Installer panel, specify the installers you would like to create.</li><li>▪ Close the Project Properties dialog, right-click the application project and choose <b>Package as   Installers</b>. The installers are created in the application 'dist' folder, visible in the Files window (Ctrl-2).</li></ul></li></ul>

To perform this task:	Follow these steps:
7. Distribute updates to the application.	<ol style="list-style-type: none"> <li>1. Right-click the application project and choose <b>Create NBMs</b>. For more information, see <a href="#">Section 5.11.2, "How to Build an NBM File"</a>. An NBM file is created for each module project in the application project. In addition, an autoupdate descriptor is created. For more information, see <a href="#">Section 5.13.2, "How to Generate an Autoupdate Descriptor"</a>.</li> <li>2. Distribute the NBM file for installation via the Plugins manager.</li> </ol>

## 5.5 Module and Rich-Client Application Tasks: Quick Reference

This section common tasks you can perform with module projects.

To perform this task:	Follow these steps:
Create a project.	<ol style="list-style-type: none"> <li>1. Choose <b>File &gt; New Project</b> (Ctrl-Shift-N).</li> <li>2. Select the right template for your project. For more information, see <a href="#">Section 5.6.1, "About Module Project Templates"</a>.</li> </ol>
Add a JAR file to a project's classpath.	<ol style="list-style-type: none"> <li>1. Choose <b>File &gt; New Project</b> (Ctrl-Shift-N).</li> <li>2. From the NetBeans Modules category, select <b>Library Wrapper Module</b>. For more information, see <a href="#">Section 5.6.3, "How to Create a Library Wrapper Module Project"</a>.</li> <li>3. Follow the steps in the rest of the wizard.</li> </ol>
Set up deployment dependencies between projects.	<p>After making a Module Suite project, you can create module projects in it, or add existing module projects to it. Then you can set up dependencies among projects in the suite by using the Libraries panel of each project's Project Properties dialog box. You create a module suite project as follows:</p> <ol style="list-style-type: none"> <li>1. Choose <b>File &gt; New Project</b> (Ctrl-Shift-N).</li> <li>2. From the NetBeans Modules category, select <b>Module Suite</b>. For more information, see <a href="#">Section 5.6.4, "How to Create a Module Suite Project"</a>.</li> <li>3. Follow the steps in the rest of the wizard.</li> </ol>
Build a project.	<p>Choose <b>Build &gt; Build Main Project</b> (F11) or right-click any project node and choose <b>Build Project</b>. In addition, you can use the IDE to build the following:</p> <ul style="list-style-type: none"> <li>▪ NetBeans Module (.nbm) file. For more information, see <a href="#">Section 5.11.2, "How to Build an NBM File"</a>.</li> <li>▪ ZIP Distribution. For more information, see <a href="#">Section 5.15.1, "How to Build a ZIP Distribution"</a>.</li> <li>▪ Java Web Start (.jnlp) file. For more information, see <a href="#">Section 5.15.2, "How to Build a JNLP Application"</a>.</li> </ul>
Clean a project.	Right-click the project node and choose <b>Clean Project</b> .
Run a project	Choose <b>Run &gt; Run Main Project</b> (F6) or right-click any project node and choose <b>Run Project</b> .

To perform this task:	Follow these steps:
Attach source code to libraries for debugging.	<ol style="list-style-type: none"> <li>Choose <b>Tools &gt; Library Manager</b> in the main menu.</li> <li>If the JAR file is not already registered in the Ant Library Manager, create a new library using the <b>Add Library</b> button.</li> <li>Select the library in the left panel of the Ant Library Manager.</li> <li>In the Classpath tab, click <b>Add JAR/Folder</b> and specify the location of the JAR file containing the compiled class files. The JAR must be the copied version in the project's <code>release/modules/ext/</code> directory, not its original location that you picked it up from when creating the library wrapper project. A library can contain multiple JAR files.</li> <li>In the Sources tab, add the folder or archive file containing the source code.</li> </ol>
Add Javadoc to a project.	<ol style="list-style-type: none"> <li>Choose <b>Tools &gt; Ant Libraries</b> in the main menu.</li> <li>If the JAR file is not already registered in the Ant Library Manager, register the JAR file as described above.</li> <li>In the Javadoc tab, click <b>Add ZIP/Folder</b> and specify the location of the Javadoc files.</li> </ol>
Set the main project.	In the main menu, choose <b>Run &gt; Set Main Project</b> and choose the project name.

## 5.6 Setting Up Modules

The IDE contains a set of standard project templates and file templates for setting up modules. The standard distribution of the IDE contains the following module project templates:

- **Module.** Use a module project as the place where you code your module.
- **Library Wrapper Module.** Use library wrapper module projects to put one or more library JAR files on a module's classpath.
- **Module Suite.** Use a module suite project to group and deploy a set of interdependent module projects and library wrapper module projects.
- **NetBeans Platform Application.** Use a NetBeans Platform Application project as the skeleton framework as the starting point of your own applications.

For more information, see [Section 5.6.1, "About Module Project Templates"](#).

The standard distribution of the IDE contains the following module file templates:

- **Action.** Creates an action that can be invoked from a menu item, pop-up menu, toolbar button, or keyboard shortcut.
- **Code Generator.** Creates a new menu item in the Insert Code popup in the editor of your choice.
- **File Type.** Lets the IDE recognize a new type of file.
- **Java SE Library Descriptor.** Adds a new class library to the Ant Library Manager of the user's IDE.
- **JavaHelp Help Set.** Creates all the files needed for building a JavaHelp help set.

- **Installer/Activator.** Creates a ModuleInstall class for a NetBeans module or a BundleActivator for OSGi bundles.
- **Layout of Windows.** Lets you design the layout of the windows in your application.
- **Options Panel.** Adds a new panel to the Options window.
- **Project Template.** Adds a new template to the New Project wizard.
- **Quick Search Provider.** Creates a new entry in the Quick Search field.
- **Update Center.** Registers an update center in the Plugins manager. As a result, the user does not need to manually register the Update Center via the Plugins manager.
- **Window.** Creates a new window with an Open action invoked from a menu item.
- **Wizard.** Creates a new wizard for creating, for example, new files in the IDE.
- **XML Layer.** Creates a layer file for registering folders and files into the central registry, also known as the System FileSystem.

Some of the file templates are used to kickstart your work with the NetBeans APIs. Other file templates are used for bundling supporting items, such as project samples and JavaHelp help sets, with your modules. For more information, see [Section 5.7.1, "Generating Skeleton API Implementations"](#) and [Section 5.8, "Bundling Supporting Items"](#).

### 5.6.1 About Module Project Templates

The IDE contains a set of standard project templates for module development. The standard distribution of the IDE contains the following module project templates:

- **Module.** Use a module project as the place where you code your module.
- **Library Wrapper Module.** Use library wrapper module projects to put one or more library JAR files on a module's classpath.
- **Module Suite.** Use a module suite project to group and deploy a set of interdependent module projects and library wrapper module projects.
- **NetBeans Platform Application.** Use a NetBeans Platform Application project as the skeleton framework as the starting point of your own applications.

#### Important Files

Module projects, library wrapper module projects, and module suite projects have an Important Files node where the IDE stores the Ant script, layer.xml file, and other project data.

The following table lists the highlights of the Important Files node and whether each is found in a module project template, a library wrapper module project template, a module suite project, or all three:

Item	Description	Availability
XML Layer	The System Filesystem registration file ( <code>layer.xml</code> ). You use this file to register new items in the System Filesystem. When you use file-level templates, the IDE automatically registers items for you. For example, when you use the Action wizard, you specify that a Java class should be invoked as a menu item, toolbar button, or keyboard shortcut, and the IDE registers your specification accordingly in the <code>layer.xml</code> file.	Module Project
Build Script	The build script called by the IDE. This build script only contains an import statement that imports targets from <code>nbproject/build-impl.xml</code> . Use the <code>build.xml</code> to override targets from <code>build-impl.xml</code> or to create new targets.	All
Module Manifest	The JAR manifest ( <code>MANIFEST.MF</code> ) with sections defining attributes for the module. By default, the <code>MANIFEST.MF</code> file created for a module project specifies its name, <code>layer.xml</code> , localizing bundle, and specification version.	Module Project
Project Metadata	The <code>project.xml</code> file. IDE-generated metadata file for specifying module dependencies and classpaths. Although you can edit <code>project.xml</code> manually, you generally do not need to. When you use a file-level template to create items, such as actions, the IDE automatically specifies module dependencies required by the item in question. Should you need to specify module dependencies yourself, use the Libraries panel in the project's Project Properties dialog box. For more information, see <a href="#">Section 5.7.1, "Generating Skeleton API Implementations"</a> .	All

## 5.6.2 How to Create a Module Project

Use a module project template to create a module project. The module project is the place where you code your module. Module projects implement the module's features and functionality that extend the NetBeans APIs, which you kickstart by using file templates. In addition, module projects include the business logic that integrate the library wrapper module projects into the module. Code housed in module projects also provides, for example, the user interface for receiving and processing user input.

To create a module project:

1. Choose **File > New Project** (Ctrl-Shift-N).
2. From the NetBeans Modules category, select **Module**. Click **Next**.
3. In the Name and Location panel, specify the following:
  - **Project Name**. Specifies the folder in which the project will be housed, prepended by the project location, which is specified in the next field.
  - **Project Location**. Specifies the location where you want to store the project metadata and source code.
  - **Project Folder**. Specifies the folder where you want to store the project metadata and source code. The folder is a concatenation of the project location and the project name.

- **Standalone Module.** Specifies that the module does not belong to a module suite. In addition, you must specify which platform the module will be compiled and deployed against. To add additional platforms to the list, click Manage and use the NetBeans Platform Manager.
  - **Add to Module Suite.** Specifies that the module belongs to a module suite. If the module suite is not currently open in the IDE, click Browse to locate the module suite in your filesystem.
4. Click **Next**.
5. In the Basic Module Configuration panel, specify the following:
- **Code Name Base.** Specifies a unique name for the module. A main package will be created with the same name as the code name base. If your code name base is `org.modules.foo`, your default package structure will be `org/modules/foo`.
  - **Module Display Name.** Specifies the name that will be displayed in the Projects window.
  - **Localizing Bundle.** Specifies location of the `Bundle.properties` file. Normally, the default suggestion should be appropriate.
  - **OSGi Bundle.** If the "Generate OSGi Bundle" checkbox is selected, an OSGi bundle will be created, instead of a NetBeans module. The difference will be visible in the Manifest file.
6. Click **Finish**.

### 5.6.3 How to Create a Library Wrapper Module Project

Use a library wrapper module project template to create a library wrapper module project. Library wrapper module projects put one or more library JAR files on a module's classpath. You have to export some or all of the packages as public, and you need to have a regular module depend on it, or the library wrapper module becomes useless. The wizard that you use to create the library wrapper module project automatically exports as public any packages it finds in the JARs.

To create a library wrapper module project:

1. Choose **File > New Project** (Ctrl-Shift-N).
2. From the NetBeans Modules category, select **Library Wrapper Module**.
3. In the Select Library panel, specify the following:
  - **Library.** Specifies one or more JAR files that make up the library. Use Shift-Click and Ctrl-Click to select more than one JAR file. You are encouraged to only include more than one JAR file if their versions are likely to increment at the same time.
  - **License.** Specifies the License of the JAR files.
4. In the Name and Location panel, specify the following:
  - **Project Name.** Specifies the folder in which the project will be housed, prepended by the project location, which is specified in the next field.
  - **Project Location.** Specifies the location where you want to store the project metadata and source code.

- **Project Folder.** Specifies the folder where you want to store the project metadata and source code. The folder is a concatenation of the project location and the project name.
  - **Add to Module Suite.** Specifies that the module belongs to a module suite. If the module suite is not currently open in the IDE, click Browse to locate the module suite in your filesystem.
5. Click **Next**.
  6. In the Basic Module Configuration panel, specify the following:
    - **Code Name Base.** Specifies a unique name for the module. A main package will be created with the same name as the code name base. If your code name base is `org.modules.foo`, your default package structure will be `org/modules/foo`.
    - **Module Display Name.** Specifies the name that will be displayed in the Projects window.
    - **Localizing Bundle.** Specifies location of the `Bundle.properties` file. Normally, the default suggestion should be appropriate.
  7. Click **Finish**.

#### 5.6.4 How to Create a Module Suite Project

Use a module suite project template to create a module suite. A module suite groups and deploys a set of interdependent module projects and library wrapper module projects. For rich-client applications, the module suite project is a skeleton application to which you can attach a splash screen, a progress bar, a name for the application's executable, and a title for the application's titlebar.

To create a module suite project:

1. Choose **File > New Project** (Ctrl-Shift-N).
2. From the NetBeans Modules category, select **Module Suite**.
3. In the Name and Location panel, specify the following:
  - **Project Name.** Specifies the folder in which the project will be housed, prepended by the project location, which is specified in the next field.
  - **Project Location.** Specifies the location where you want to store the project metadata.
  - **Project Folder.** Specifies the folder where you want to store the project metadata and source code. The folder is a concatenation of the project location and the project name.
  - **NetBeans Platform.** Specifies the platform against which the module will be compiled and deployed. To add additional platforms to the list, click **Manage** and use the NetBeans Platform Manager.
4. Click **Finish**.

#### 5.6.5 How to Create a NetBeans Platform Application Project

Use a NetBeans Platform Application project template to create the starting point of your application on top of the NetBeans Platform. This template project provides a skeleton application to which you can attach a splash screen, a progress bar, a name for the application's executable, and a title for the application's titlebar.

To create a NetBeans Platform Application project:

1. Choose **File > New Project** (Ctrl-Shift-N).
2. From the NetBeans Modules category, select **NetBeans Platform Application**.
3. In the Name and Location panel, specify the following:
  - **Project Name.** Specifies the folder in which the project will be housed, prepended by the project location, which is specified in the next field.
  - **Project Location.** Specifies the location where you want to store the project metadata.
  - **Project Folder.** Specifies the folder where you want to store the project metadata and source code. The folder is a concatenation of the project location and the project name.
  - **NetBeans Platform.** Specifies the platform against which the module will be compiled and deployed. To add additional platforms to the list, click **Manage** and use the NetBeans Platform Manager.
4. Click **Finish**.

## 5.7 Using the NetBeans APIs

The NetBeans APIs are the public interfaces and classes which are available to module writers. They are divided into specific APIs for dealing with different types of functionality. The contents and behavior of the Java source packages and its subpackages, as specified in the NetBeans API List, are the NetBeans APIs.

After you register the NetBeans API sources and Javadoc, you can refer to them in the Source Editor, while developing NetBeans modules. For more information, see [Section 5.7.3, "How to Register the NetBeans Sources and Javadoc"](#) and [Section 5.7.4, "How to Use the NetBeans Sources and Javadoc"](#).

For the NetBeans API List, see <http://bits.netbeans.org/dev/javadoc/>.

### 5.7.1 Generating Skeleton API Implementations

To simplify the process of working with the NetBeans APIs, the NetBeans IDE provides several wizards that guide you through the initial phase of working with a NetBeans API. For more information, see [Section 5.7, "Using the NetBeans APIs"](#). For example, the New Action wizard provides the basis of an implementation of the NetBeans Actions API, the New File Type wizard provides the basis of an implementation of the NetBeans Datasystems API, and so on.

The NetBeans API wizards are as follows:

- **Module.** Use a module project as the place where you code your module.
- **Library Wrapper Module.** Use library wrapper module projects to put one or more library JAR files on a module's classpath.
- **Module Suite.** Use a module suite project to group and deploy a set of interdependent module projects and library wrapper module projects.
- **NetBeans Platform Application.** Use a NetBeans Platform Application project as the skeleton framework as the starting point of your own applications.

### Project Templates

Project templates come in two types:

- Templates that users build on when creating their own project. For example, in the IDE a user chooses the 'Web Application' project template in the New Project wizard, then the IDE creates a project consisting of a JSP file, a web.xml file, a server-specific deployment file, and project metadata within a specific structure that is useful for web application projects.
- Samples that illustrate some aspect of project functionality. For example, in the Samples directory within the New Project wizard, an Anagram Game is included to demonstrate Java SE functionality. Samples are a kind of project template; they have the same behavior as project templates, but they are used for a different purpose.

A project template is made available to the IDE's New Project wizard once it has been registered in the `layer.xml` file. For more information, see [Section 5.9.2, "About XML Layer Files"](#).

You use the New Project Template wizard to create the basic files and to register the template in the `layer.xml` file.

Before you can use the New Project Template wizard, you must have project in the IDE that is structured in exactly the way that you would like it to be available in the New Project wizard. For example, if you are going to create a new project sample, you must first lay it out in the IDE. Then use the New Project Template wizard to add the template to the New Project wizard.

When you make the module that contains the new project template available as an NBM file, the user can install it via the Plugins manager and then, once it has been successfully installed, select it from the New Project wizard.

The standard distribution of the IDE contains the following module file templates:

- **Action.** Creates an action that can be invoked from a menu item, pop-up menu, toolbar button, or keyboard shortcut.
- **Code Generator.** Creates a new menu item in the Insert Code popup in the editor of your choice.
- **File Type.** Lets the IDE recognize a new file type.
- **Java SE Library Descriptor.** Adds a new class library to the Ant Library Manager of the user's IDE.
- **JavaHelp Help Set.** Creates all the files needed for building a JavaHelp help set.
- **Installer/Activator.** Creates a ModuleInstall class for a NetBeans module or a BundleActivator for OSGi bundles.
- **Layout of Windows.** Lets you design the layout of the windows in your application.
- **Options Panel.** Adds a new panel to the Options window.
- **Project Template.** Adds a new template to the New Project wizard.
- **Quick Search Provider.** Creates a new entry in the Quick Search field.
- **Update Center.** Registers an update center in the Plugins manager. As a result, the user does not need to manually register the Update Center in the Plugins manager.
- **Window.** Creates a new window with an Open action invoked from a menu item.
- **Wizard.** Creates a new wizard for creating, for example, new files in the IDE.
- **XML Layer.** Creates a layer file for registering folders and files into the central registry, also known as the System FileSystem.

The NetBeans API wizards create the starting point for your development activities. Once you have worked through a wizard, you build your module's functionality on top of the files that the wizard creates for you.

#### Related NetBeans API Javadoc

The NetBeans API wizards create the starting point for your development activities. Once you have worked through a wizard, you build your module's functionality on top of the files that the wizard creates for you.

The following table lists the Javadoc that you will need to refer to when building on top of the skeleton API implementations:

File Template	Related NetBeans Javadoc
Action	Utilities API
File Type	Datasystems API
Module Installer	Module System API
Options Panel	Options Dialog and SPI
Project Template	Project UI API Project API
Window	Window System API
Wizard	Dialogs API

##### 5.7.1.1 About Actions

Actions are defined by the Actions API.

The Actions API is a standard representation of the actions a user can invoke. It provides an interface to such IDE elements as toolbars, menus, and keyboard shortcuts, allowing third parties to create actions that are sensitive to context and invocable in more than one way. The Actions API offers the ability to write the action once, and have it automatically apply as appropriate. For example, a user action might both show up in a toolbar and be selected by a keyboard shortcut, all from the same implementation.

Actions are typically presented in pop-up menus, or attached to a component such as a window, node, or data object.

##### 5.7.1.2 How to Create an Action

The New Action wizard creates a new action. For more information, see [Section 5.7.1.1, "About Actions".](#)

To create a new action:

1. Right-click a module project and choose **New > Action**.
2. In the Action Type page, you set the type:
  - **Always Enabled.** Specifies that the action will be invoked from a menu item in the menu bar, from a toolbar button in a toolbar, or from a keyboard shortcut anywhere in the IDE.
  - **Conditionally Enabled.** Specifies the classes for which the action will be created. This is normally one of your own business objects. However, optionally, you can also include one or more of the following, which creates the action for all classes that subclass them:

- DataObject
- EditCookie
- EditorCookie
- OpenCookie
- Project

**3. User Selects One Node.**

**4. User May Select Multiple Nodes.** Specifies the conditions under which the action will be enabled. By default, will be enabled if one, and only one, node holding the business object of interest is selected. If checked, it will also be enabled if multiple nodes all holding the desired business object(s) are selected.

**5. Click Next.**

**6.** In the GUI Registration page, you have to specify how the user will be able to call the new action.

- **Category.** Specifies where the action will be displayed in the Keymap section of the Options window.
- **Global Menu Item.** Specifies the menu where the action will be displayed as an item. You can also specify the position within the menu, and whether you want a separator to appear before it, after it, or both.
- **Global Toolbar Button.** Specifies the toolbar where the action will be displayed as a button. You can also specify the position within the toolbar.
- **Global Keyboard Shortcut.** Specifies a shortcut that will invoke the action.

Only if Conditionally Enabled is selected in the Action Type page can you set the following items:

- **File Type Context Menu Item.** Specifies the file type where the action will be displayed in the pop-up menu.
- **Editor Context Menu Item.** Specifies the IDE editor where the action will be displayed in the pop-up menu.

**Click Next.**

**7.** In the Name, Icon, and Location page, you set the following:

- **Class Name.** Specifies the name of the new Action class.
- **Display Name.** Specifies the action's label. (Optional)
- **Icon.** Specifies the icon that will accompany the action. For example, if the action will be invoked by a toolbar button, the icon specified here is displayed on the toolbar button.
- **Package.** Specifies the name of the package where the class will be housed.

**Click Finish.**

**8.** After completing the wizard, you can do the following:

- Tweak the `layer.xml` file to change the icon or to rearrange the position of the action within the actions provided by the available modules. For more information, see [Section 5.9.2, "About XML Layer Files"](#).

### 5.7.1.3 About Code Generators

A code generator is an action that is added to the Insert Code popup, which the user invokes when pressing Alt-Insert. When invoked, the action is intended to insert code into the editor. Traditionally, code generators are only found in Java documents but, from NetBeans 6.1 onwards, they can also be found in all other types of documents.

Via the Code Generator SPI and the accompanying Code Generator wizard, you can quickly and easily extend the code generator popup for a specific MIME type with new entries.

### 5.7.1.4 How to Create a Code Generator

A code generator is an item added to the popup that appears in an editor when Alt-Insert is pressed. For more information, see [Section 5.7.1.3, "About Code Generators"](#).

By default, for example, when you press Alt-Insert in a Java class, a constructor can be generated.

To create a new code generator:

1. Right-click a module project and choose **New > Other**. In the New File wizard, choose Code Generator under the Module Development category.
2. In the Code Generator page, set the following:
  - **Class Name**. Specifies the name of the new code generator.
  - **MimeType**. Specifies a unique Multipurpose Internet Mail Extension (MIME) type that enables the code generator to be registered in the layer.xml file.
  - **Generate Code Generator Context Provider**. Specifies that a class should be created, and registered in the layer.xml file, for adding new objects to the code generator's lookup.
3. Click **Finish**.

### 5.7.1.5 About File Types

Many file types are recognized by default by the IDE. For example, JSP files, Java source files, and HTML files are recognized as such and the IDE provides functionality specific to the file type. For example—for JSP and HTML files, the IDE provides special syntax highlighting that is different from the syntax highlighting provided for Java source files. In addition, the menu items provided for JSP files are different from those provided for HTML files. For example, you can compile a JSP file but not an HTML file.

Recognition of a file type is generally made possible via its extension. All JSP files have a .jsp extension, while all Java source files have a .java extension. On the basis of this distinction, the IDE provides distinct functionality for these file types. You can use the New File Type wizard to let the IDE recognize additional file types, i.e., file types that are not recognized by the IDE by default. For example, if you have a file type with the file extension .xyz, you can let the IDE recognize all files with this extension and then provide functionality specifically for this file type.

But you can also let the IDE distinguish between XML files. Whether an XML file has an .xml extension, you can let the IDE provide different functionality for abc.xml than for def.xml, based on the namespace defined for the XML file in question. If the namespace of each distinguishable XML file is distinct, the namespace is used to distinguish the file types in this case.

When you use the New File Type wizard, the IDE creates the following files for you:

File	Purpose
xxxDataObject.java	A class that extends Class MultiDataObject.
xxxResolver.xml	Declarative resolution of MIME-type.
xxxTemplate.xxx	Dummy template, registered in the layer.xml file as a file template.

### 5.7.1.6 How to Create a File Type

The New File Type wizard creates a new file type. For more information, see [Section 5.7.1.5, "About File Types"](#).

To create a new file type:

1. Right-click a module project and choose **New > File Type**.

2. In the File Recognition page, set the following:

- **MIME Type.** Specifies a unique Multipurpose Internet Mail Extension (MIME) type that enables the file type to be registered in the layer.xml file. Together, the MIME type and the filename extension or XML root element enable the IDE to distinguish one file type from another.

- **by**

- **Filename Extension.** Specifies one or more file extensions that the IDE will recognize as belonging to the specified MIME type. The file extension can optionally be preceded by a dot. Separators are commas, spaces, or both. Therefore, all of the following are valid:

.abc,.def

.abc .def

abc def

abc,.def ghi , .wow

- **XML Root Element.** Specifies a unique namespace that distinguishes the XML file type from all other XML file types.

The value that you specify is the namespace associated with the root element of your XML document. Note that this is not the same thing as the name of the root element. For example, in the following example XML document, the namespace is "sample":

```
<mydata xmlns="sample"></mydata>
```

3. In the Name, Icon, and Location page, set the following:

- **Class Name.** Specifies the name of the new files that will be generated.
- **Icon.** Specifies the icon that will accompany the new file type. For example, when you see the new file type in the Projects window, Files window, or Favorites window, it will be identified by the icon specified here. (Optional)
- **Use MultiView.** Specifies that additional files should be generated as the starting point of an extensible multiview editor.
- **Package.** Specifies the name of the package where the class will be housed.

4. After completing the wizard, you can do one or more of the following:

- Tweak the `layer.xml` file to change the icon or to rearrange the position of the action within the actions provided by the available modules. For more information, see [Section 5.9.2, "About XML Layer Files"](#).
- Add properties to the default property sheet used by the new file type.
- Use the New Action wizard to add actions to the new file type's pop-up menu. When you do so, select **Conditionally Enabled** in the first page of the New Action wizard.
- Use the New Wizard wizard to create a New File wizard that the user will use to create new files of the new type defined in the New File Type wizard.
- Use the New Project Template wizard to add a sample project containing example files of the new file type to the New Project wizard.
- Add advanced support for the file type, such as syntax highlighting.

#### 5.7.1.7 About Module Installers

A module installer is a Java class that provides hooks for running code on startup or when a module is loaded. It can also run cleanup code when a module is uninstalled or disabled.

**Note:** In general, using a module installer is not recommended, because it slows down startup time. Before using a module installer, make sure that there is no declarative way of doing what you are trying to do. The main declarative way of installing items is to use the `org.openide.util.lookup.ServiceProvider` annotation or create an XML layer file that declares information about the items your module is installing. For more information, see [Section 5.9.2, "About XML Layer Files"](#).

Then, when they are needed to do actual work, your items will be instantiated.

In addition to providing a module installer class, you need to add an entry to the MANIFEST file. The Module Installer Wizard creates a skeleton implementation of a module installer, adds the entry to the MANIFEST file, as well as entries to the `project.xml` file, which provides the module's metadata. For more information, see [Section 5.7.1.9, "How to Create a Module Installer/Activator"](#).

#### 5.7.1.8 Installing Modules

Applications can install modules dynamically. Any application can include the Update Center module, to allow users of that application to download digitally-signed upgrades and new features via the web, directly into the running application.

Installing an upgrade or a new release does not force users to download the entire application again. And in an application with multiple modules, upgrades of specific functionality can be incremental, further improving customer response time and time-to-market.

#### 5.7.1.9 How to Create a Module Installer/Activator

A module installer (for a NetBeans module) or activator (for an OSGi bundle) is a Java class that provides hooks for running code on startup or when a module is loaded. It can also run cleanup code when a module is uninstalled or disabled.

To create a new installer/activator:

1. Right-click a module project and choose **New > Other**. In the New File wizard, choose **Installer/Activator** under the Module Development category.

2. In the New Installer/Activator page, you have to set Package. This specifies the name of the package where the module installer will be housed. The module installer will always be named `Installer`.

#### **5.7.1.10 About Options Panels**

An Options panel is a category of the Options window. The Options window is where the user defines settings, such as the location of the web browser used by the IDE. The category can be displayed as a primary panel (such as the "General" panel in the IDE) or as an addition to one of the other panels, for example, the Miscellaneous panel (such as the "Ant" or "GUI Builder" panel in the IDE's Miscellaneous panel).

You use the Options Panel wizard to create options panels.

#### **5.7.1.11 How to Create an Options Panel**

An Options panel adds a category to the IDE's Options window. For more information, see [Section 5.7.1.10, "About Options Panels"](#).

The category can be displayed as a primary panel (such as the "General" panel in the IDE) or as an addition to an existing panel, for example, the Miscellaneous panel (such as the "Ant" or "GUI Builder" panel in the IDE's Miscellaneous panel).

To create a new Options panel:

1. Right-click a module project and choose **New > Other**. In the New File wizard, choose **Options Panel** under the Module Development category.
2. In the Choose Panel Type page, choose one of the following:
  - **Create Secondary Panel**. Specifies that the category will be displayed as an addition to an existing panel, for example, the Miscellaneous panel (such as the "Ant" or "GUI Builder" panel in the IDE's Miscellaneous panel).
  - **Create Primary Panel**. Specifies that the category will be displayed as a primary panel (such as the "General" panel in the IDE).
3. In the Location page, you have to set the following:
  - **Class Prefix**. Specifies the prefix of the source files generated by the wizard.
  - **Package**. Specifies the name of the package where the source files will be housed.
4. After completing the wizard, you can do one or more of the following:
  - Add a titled border to make the Options panel resemble other NetBeans Options categories.
  - Use the GUI Builder to add items such as checkboxes and textfields to the generated panel.

#### **5.7.1.12 About Quick Search Providers**

A quick search provider is a Java class that lets you plug new items into the Quick Search feature in the IDE or any other application on top of the NetBeans Platform.

Quick search providers are registered in the `layer.xml` file. You can let the IDE create and register them by means of the Quick Search Provider wizard.

#### **5.7.1.13 How to Create Quick Search Providers**

A quick search provider is a Java class that lets you plug new items into the Quick Search feature in the IDE or any other application on top of the NetBeans Platform.

To create a new quick search provider:

1. Right-click a module project and choose **New > Other**. In the New File wizard, choose **Quick Search Provider** under the Module Development category.
2. In the File Recognition page, you have to set the following:
  - **Provider Class Name**. Specifies the class name of the stub that the wizard will generate.
  - **Package**. Specifies the package where the stub class will be generated.
  - **Category Display Name**. Specifies the display name of the category that the stub will create.
  - **Command Prefix**. Specifies prefix for narrowing the search to the category that the stub will create.
  - **Position in Popup**. Specifies the position of the new item in the within the Quick Search feature.

#### 5.7.1.14 About Windows

A window component creates a window (also known as 'view') for a module. For example, the IDE's Projects window is a 'window', just as the Navigator, Output window, Palette, and Debugger. The main class in each of the modules that define these windows subclasses Class TopComponent.

Understanding and working with the TopComponent class is the key to creating useful and reliable windows in the IDE. By using the Window wizard, you can be sure of the success of the initial development phase. This is because when you use the New Window wizard, the IDE creates a Matisse GUI Builder form that extends the TopComponent class.

#### 5.7.1.15 How to Create a Window

A window component creates a window (also known as 'view') for a module. For more information, see [Section 5.7.1.14, "About Windows"](#).

The IDE provides the Window wizard to simplify the initial creation process.

To create a new window component:

1. Right-click a module project and choose **New > Window**.
2. In the Basic Settings page, you set the following:
  - **Window Position**. Specifies the location of the window component when open in the IDE. Depending on the modules installed in the platform that forms the basis of your application, you're able to choose from several window positions, including the following:
    - bottomSlidingSide. The window component will be available as a button on the bottom bar of the IDE.
    - commonpalette. The default position of the IDE's Palette for HTML/JSP code snippets.
    - debugger. The default position of the IDE's Debugger.
    - editor. The default position of the IDE's Source Editor.
    - explorer. The default position of the IDE's Projects window.
    - leftSlidingSide. The window component will be available as a button on the left sidebar of the IDE.

- navigator. The default position of the IDE's Navigator.
  - output. The default position of the IDE's Output window.
  - properties. The default position of the IDE's Properties window.
  - rightSlidingSide. The window component will be available as a button on the right side of the Source Editor.
  - **Open on Application Start.** Specifies whether the window component is open by default or not.
  - **Keep preferred size when slided-in.** Specifies that the size of the window when minimized will be determined by the preferred size property of the window. When selected, the following line is added to the TopComponent's constructor: `putClientProperty("netbeans.winsys.tc.keep_preferred_size_when_slided_in", Boolean.TRUE);`
  - **Sliding not allowed.**  
**Closing not allowed.**  
**Undocking not allowed.**  
**Dragging not allowed.**  
**Maximization not allowed.** Specify that the window should not have one or more features that it has by default.
3. In the Name, Icon, and Location page, you have to set the following:
- **Class Name.** Specifies the name of the new Action and TopComponent classes.
  - **Icon.** Specifies the icon that will accompany the window component. For example, you will see the icon in the label of the window component. (Optional)
  - **Package.** Specifies the name of the package where the class will be housed.
4. After completing the wizard, you can do one or more of the following:
- Use the Form editor to design the window.
  - Place the window component in a different position. In the `layer.xml` file, tweak the Modes section.
  - Add your own code to `componentOpened()` and `componentClosed()` to specify what will happen when the window opens and closes.

### 5.7.1.16 About Wizards

Wizards are defined by the Wizard Descriptor API.

The Wizard API lets you create wizard panels that have steps, graphics, left side-bar text, and a user panel on the right. The Wizard wizard in the IDE helps you by creating several of the basic Java source files, containing sample code, for you.

Depending on the selections you make in the Wizard Type panel in the New Wizard wizard, the IDE creates the following Java source files for you:

- *Registration Type: Custom*  
*Wizard Step Sequence: Static*  
**Created files:**
  - `SampleAction.java`

- VisualPanel.java (for each wizard step)
- WizardPanel.java (for each wizard step)

These files are ideal for uncomplicated wizards that progress sequentially from panel to panel without divergences or reversals. A menu item or toolbar button invokes the wizard and subsequent steps are generally linear and forward-directed.

- *Registration Type: Custom*

*Wizard Step Sequence: Dynamic*

Created files:

- WizardIterator.java
- VisualPanel.java (for each wizard step)
- WizardPanel.java (for each wizard step)

These files are for wizards that provide more flexibility to the user. A WizardDescriptor.Iterator class guides progress from one panel to the next. The developer has a lot more freedom in coding the wizard, but has a more complex task since there are many more possibilities to consider. Even though the Custom/Simple wizard type can also be extended to provide support for panel skipping and reversals, the Custom/Dynamic type was made for this purpose. For example, the Add Server Instance wizard offers different panels depending on the type of server that the user wants to register.

- *Registration Type: New File*

Created files:

- WizardIterator.java
- VisualPanel.java (for each wizard step)
- WizardPanel.java (for each wizard step)
- .HTML file (for the description area in the New File wizard)

These files are for wizards that are used to create new files. This wizard is registered in the New File wizard via the layer.xml file. All the necessary entries in the layer.xml file are created for you by the Wizard wizard. In addition, when you make this choice, the Wizard wizard creates a WizardDescriptor.Iterator.

The iterator lets you provide the direction and sequence of the wizard. The New File wizard can be as simple or as complex as your needs dictate. An HTML file is created by the Wizard wizard so that you can provide a description for your new wizard in the New File wizard.

#### 5.7.1.17 How to Create A Wizard

The Wizard wizard helps you by creating several of a wizard's basic Java source files, containing sample code, for you. For more information, see [Section 5.7.1.16, "About Wizards"](#).

When you want to create a wizard, you do not always need to use the Wizard wizard. The table below helps you to chose.

---

When you want to create a...	Use a...
New Project wizard	New Project Template wizard

---

When you want to create a...	Use a...
New Sample wizard	New Project Template wizard
New File wizard	'New File' registration type in the New Wizard wizard
Any other type of wizard	'Custom' registration type in the New Wizard wizard

Depending on the type of wizard you want to create, do the following:

To create a New File wizard or a custom wizard:

1. Right-click a module project and choose **New > Project Template**.
2. In the Wizard Type page, you have to set the following:

- **Registration Type.** Specifies where the user will be able to find the wizard in the IDE. For more information, see [Section 5.7.1.16, "About Wizards"](#).
- **Wizard Step Sequence.** Specifies whether the Wizard wizard will create an implementation of this NetBeans API class: WizardDescriptor.Iterator.

If you choose Static, an implementation of this NetBeans API class will not be created. The default progression from panel to panel will then be supported by your wizard, sequential progression without divergences or reversals.

If you choose Dynamic, the Wizard wizard will create an implementation of the WizardDescriptor.Iterator class. This class guides progress from one panel to the next. The developer has a lot more freedom in coding the wizard, but has a more complex task since there are many more possibilities to consider. When you choose "Static", you can also extend the wizard to provide support for panel skipping and reversals, but the "Dynamic" type was made for this purpose. For example, the Add Server Instance wizard offers different panels depending on the type of server that the user wants to register.

- **Number of Wizard Panels.** For each panel, the IDE creates two files—a visual panel (called `xxxVisualPanel.java`) and a wizard panel (called `xxxWizardPanel`) for retrieving the current values from the visual panel. Only if you enter an integer greater than 0, will you be able to progress to the next panel in the Wizard wizard.

In the Name, Icon, and Location page, you have to set the following:

- **Class Name Prefix.** Specifies the name of the new panels and action class.
  - **Package.** Specifies the name of the package where the class will be housed.
3. Only if the wizard will be registered as a New File wizard can you set the following items:
    - **Display Name.** Specifies the display name for the new wizard in the New File wizard.
    - **Category.** Specifies the New File wizard category where the new file wizard will be located.
    - **Icon.** Specifies the icon that will be displayed in the New File wizard.
  4. Click **Finish**.

After completing the wizard, you can do one or more of the following:

- Tweak the `layer.xml` file to change the icon or rearrange the position of the template in the New Project wizard.

- Change the description that will appear in the New Project wizard's Description box.

To create a new project template or a sample wizard:

1. Right-click a module project and choose **New > Project Template**.
  2. In the Select Project page, you have to specify the project that you want to make available as a project template or sample. You can use the Project drop-down list to select an open project or you can click Browse to browse to a project in your filesystem.
- Click **Next**.
3. In the Name, Icon, and Location page, you have to set the following:
    - **Template Name**. Specifies the name of the new project template.
    - **Display Name**. Specifies the template's label. For example, you will see this label in the Plugins manager. (Optional)
    - **Category**. Specifies the template's category. For example, this will enable the user to find the template more easily in the New Project wizard. (Optional)
    - **Package**. Specifies the name of the package where the classes will be housed.
  4. Click **Finish**.

After completing the wizard, you can do one or more of the following:

- Tweak the `layer.xml` file to change the icon or rearrange the position of the template in the New Project wizard.
- Change the description that will appear in the New Project wizard's Description box.

## 5.7.2 Extending Skeleton API Implementations

Once you have used the NetBeans API wizards, you extend the generated code by using the NetBeans APIs. For more information, see [Section 5.7.1, "Generating Skeleton API Implementations"](#). Several tools are provided specifically to help you at this stage of your development cycle. Among them are the following:

- Access to NetBeans sources and Javadoc (for more information, see [Section 5.7.4, "How to Use the NetBeans Sources and Javadoc"](#)). After you register the NetBeans sources and Javadoc, you can access them inside the IDE (see [Section 5.7.3, "How to Register the NetBeans Sources and Javadoc"](#) for details). This gives you a quick and easy reference to the API classes and methods that you are implementing.
- Search facility for NetBeans APIs (for more information, see [Section 5.7.5, "How to Search for NetBeans APIs"](#)). If you know the class that you need to use, but not the module (API or non-API) to which it belongs, a search facility is provided to help you. Once you have identified the module, the IDE registers it in the module project's `project.xml` file.
- Code Templates for NetBeans APIs (for more information, see [Section 5.7.6, "Code Templates for NetBeans APIs"](#)). For several common tasks, you can type an abbreviation in the Java editor, press the registered expansion key (which is the Tab key, by default), and then the abbreviation will expand to a full piece of code.

### 5.7.3 How to Register the NetBeans Sources and Javadoc

The NetBeans Platform Manager is a tool for registering different NetBeans platforms with the IDE. A NetBeans platform can be an installation of the NetBeans Platform (see [Section 5.2, "About the NetBeans Platform"](#) for details) or an installation of the NetBeans IDE. The default platform is always configured, and is the running IDE. Even in the case that a target platform is the complete IDE, you can easily create an application based only on the platform subset, so that there is little need to download the NetBeans Platform separately. The NetBeans Platform Manager lists all your registered NetBeans platforms in the left pane and lists the platform that the IDE currently deploys to as the default.

You can open the NetBeans Platform Manager by choosing **Tools > NetBeans Platform Manager**.

Using the NetBeans Platform Manager, you can:

- Register a new platform.
- View a registered platform's available modules.
- Register source code for a platform in the Sources tab. If you register the sources, you do not need to register Javadoc separately, because Javadoc is included in the sources. Either the ZIP file containing the sources or the unpacked root folder can be registered in the Sources tab. You can download the NetBeans sources from <https://netbeans.org/community/sources/>.
- Register Javadoc documentation for a platform in the Javadoc tab. Javadoc is available from the Plugins manager as "NetBeans API Documentation". Choose **Tools > Plugins** to access the Plugins manager.

Once you have registered a platform, you can configure a module project to use that platform for deployment. Go to the project's Project Properties dialog box, select the Libraries node, and choose the appropriate platform in the NetBeans Platform drop-down list.

### 5.7.4 How to Use the NetBeans Sources and Javadoc

Once you have registered the NetBeans Sources and Javadoc (see [Section 5.7.3, "How to Register the NetBeans Sources and Javadoc"](#) for details), you can use them as follows:

- In the Source Editor, when you use code completion (Ctrl-Space), Javadoc for the related NetBeans API class is shown in the code completion window.  
You do not need to register Javadoc for this feature to work. Just registering the NetBeans sources suffices.
- In the Source Editor, right-click on a class identifier that belongs to the NetBeans APIs and choose Show Javadoc (Alt-F1). Javadoc for the related NetBeans API class is shown in a separate pane in the Source Editor.
- In the Source Editor, hold down the Ctrl key and move the mouse over a class identifier belonging to the NetBeans APIs. A hyperlink appears. Click on the hyperlink and the cursor jumps to the related NetBeans API class.

### 5.7.5 How to Search for NetBeans APIs

When using the NetBeans APIs, you may know the class that you need to use, but not the API (or non-API module) to which it belongs. Until you make the API available to your module, you are unable to make use of its classes.

To search for a NetBeans API:

1. Right-click the Libraries node and choose **Add Module Dependency**.  
The Add Module Dependency dialog box appears
2. In the Add Module Dependency dialog box, in the Filter textbox, start typing the name of the class that you need to use. The Module list narrows, showing only the modules that satisfy the filter.  
If the Show Non-API Modules checkbox is unchecked (default), the IDE excludes any module for which the current module cannot access any packages without an implementation dependency. These are known as "non-API modules". In other words, only modules with public packages are included, or friend packages where this module is a friend.
3. When you find the module that you need to use, click **OK**. The IDE adds an entry to the `project.xml` file.

## 5.7.6 Code Templates for NetBeans APIs

In the Java editor in the IDE, type an abbreviation listed below, press the expansion key (which is Tab, by default), and then the expanded text shown below will be generated.

- Abbreviation: **2do**

Description: Convert FileObject to DataObject

Expands to:

```
try {  
    ${dobType type="org.openide.loaders.DataObject" editable="false"  
              default="DataObject"}  
    ${dob newVarName default="dob"} = ${dobType}.find(${fo  
              instanceof="org.openide.filesystems.FileObject" default="fo"});  
    ${cursor}  
    {catch (${etype type="org.openide.loaders.DataObjectNotFoundException"  
              default="DataObjectNotFoundException" editable="false"}  
          ${exName newVarName default="ex" editable="false"}) {  
        ${exctype type="org.openide.util.Exceptions" editable="false"  
                  default=""}.printStackTrace(${exName});  
    }  
}
```

Example:

```
try {    DataObject dataObject = DataObject.find(myFo);  
} catch (DataObjectNotFoundException dataObjectNotFoundException) {  
    Exceptions.printStackTrace(dataObjectNotFoundException);  
}
```

- Abbreviation: **2f**

Description: Convert FileObject to java.io.File

Expands to:

```
 ${fileType type="java.io.File" default="File" editable="false"}  
 ${file newVarName default="f"} =  
 ${FileUtilType type="org.openide.filesystems.FileUtil"  
             editable="false"}).toFile(${fo  
             instanceof="org.openide.filesystems.FileObject"  
             default="fo"});  
 ${cursor}
```

Example:

```
File file = FileUtil.toFile(myFo);
```

- Abbreviation: **2fo**

Description: Convert java.io.File to FileObject

Expands to:

```
 ${fileType type="org.openide.filesystems.FileObject"
 default="FileObject" editable="false"}
 ${file newVarName default="f"}
 = ${FileUtilType type="org.openide.filesystems.FileUtil"
 editable="false").toFileObject(${FileUtilType}.normalizeFile
 (${f instanceof="java.io.File" default="f"})); ${cursor}
```

Example:

```
FileObject fileObject = FileUtil.toFileObject(FileUtil.normalizeFile(myFile));
```

- Abbreviation: **Lka**

Description: Find all implementations of a certain type registered in META-INF/services.

Expands to:

```
 ${coltype type="java.util.Collection" default="Collection" editable="false"}
 ${obj newVarName default="obj"} =
 ${lkptype editable="false" default="Lookup"
 type="org.openide.util.Lookup".getDefaultValue().lookupAll(${Type}.class);
 ${cursor}
```

Example:

```
Collection<? extends Type> collection =
 Lookup.getDefault().lookupAll(Type.class);
```

- Variation: **lka** (i.e., the first character is lowercase)

Description: Find all implementations of a certain type from a local lookup, e.g., TopComponent, Node, or DataObject.

Example:

```
Collection<? extends Type> collection = myNode.lookupAll(Type.class);
```

- Abbreviation: **Lkp**

Description: Find a single typed implementation registered in META-INF/services.

Expands to:

```
 ${Type} ${obj newVarName default="obj"} = ${lkptype editable="false"
 default="Lookup"
 type="org.openide.util.Lookup".getDefaultValue().lookup(${Type}.class);
 ${cursor}
```

Example:

```
Type type = Lookup.getDefault().lookup(Type.class);
```

Variation: **lkp** (i.e., the first character is lowercase)

Description: Find a single implementation of a certain type from a local lookup, e.g., TopComponent, Node, or DataObject.

Example:

```
Type type = myNode.lookup(Type.class);
```

- Abbreviation: **Lkr**

Description: Assign a single typed instance from META-INF/services to a Result object, to which you can listen for changes.

Expands to:

```
 ${coltype type="org.openide.util.Lookup.Result" default="Lookup.Result"
 editable="false"} ${obj newVarName default="res"} = ${lkptype editable="false"
 default="Lookup"
 type="org.openide.util.Lookup".getDefaultValue().lookupResult(${Type}.class);
 ${cursor}
```

Example:

```
Lookup.Result<? extends Type> res =
Lookup.getDefault().lookupResult(Type.class);
```

Variation: **lkr** (i.e., the first character is lowercase)

Description: Assign a single typed instance from a local lookup to a Result object, to which you can listen for changes.

Example:

```
Result<? extends Type> all = myNode.lookupResult(Type.class);
```

- Abbreviation: **Iko**

Description: Create a lookup for a local object, e.g., TopComponent, Node, or DataObject.

Expands to:

```
 ${Type} ${obj newVarName default="obj"} = ${prov
 instanceof="org.openide.util.Lookup.Provider".getLookup().lookup(${Type}.class
 ); ${cursor}}
```

Example:

```
Type type = myNode.getLookup().lookup(Type.class);
```

- Abbreviation: **stat**

Description: Create code for writing text obtained from a Bundle.properties file into the status bar.

Expands to:

```
 ${coltype type="org.openide.awt.StatusDisplayer" default="StatusDisplayer"
 editable="false".getDefaultValue().setStatusText(${bundleType
 type="org.openide.util.NbBundle" default="NbBundle"
 editable="false".getMessage(getClass(), "${KEY}")); ${cursor}}
```

Example

```
StatusDisplayer.getDefaultValue().setStatusText(NbBundle.getMessage(getClass(),
 "KEY"));
```

- Abbreviation: **nb**

Description: Get a text from a Bundle.properties file.

Expands to:

```
 ${coltype type="org.openide.util.NbBundle" default="NbBundle"
 editable="false"}.getMessage(${classVar editable="false" currClassName
 default="getClass()".class, "${KEY}"})
```

Example:

```
NbBundle.getMessage(DemoAction.class, "KEY")
```

Variation: **nbb** (i.e., add an additional 'b' character)

Description: Pass in parameters for formatting the text.

Example:

```
NbBundle.getMessage(DemoAction.class, "KEY", params)
```

## 5.8 Bundling Supporting Items

The IDE provides wizards for bundling various supporting items with your module. Using these wizards means you can simply point-and-click at the items you want to include, and then the IDE does all the work for you.

You can use wizards to quickly and easily bundle the following items with your module:

- Libraries. External JARs can be included in your NetBeans Platform modules in various ways:
  - If you are extending the IDE, you can add a new library to the Ant Library manager, which enables the user to add the library to the classpath of their application. For this you need a Java SE library descriptor, which the IDE can create for you.
  - If you are creating extensions to the IDE or are creating your own applications on the NetBeans Platform, you can include external JARs, NetBeans modules, and OSGi bundles. You can also include groups and combinations of these items. The IDE provides support for all of these scenarios.

For details, see [Section 5.8.1, "How to Bundle a Library"](#).

- Project Templates. Project templates are used when users begin to create their own project. For example, in the IDE a user chooses the 'Web Application' project template in the New Project wizard, then the IDE creates a project consisting of a JSP file, a web.xml file, a server-specific deployment file, and project metadata within a specific structure that is useful for web application projects.

For details, see [Section 5.8.2, "How to Bundle a Project Template or Sample"](#).

- Project Samples. Project samples that illustrate some aspect of project functionality are often provided by modules that inject a new technology into the IDE. For example, in the Samples directory within the New Project wizard, an Anagram Game is included to demonstrate Java SE functionality. A sample is a kind of project template; it has the same behavior as a project template, but it is used for a different purpose.

For details, see [Section 5.8.2, "How to Bundle a Project Template or Sample"](#).

- Update Center URLs. When you bundle the URL to a update center, you help your users, because they will not need to register your update center manually. For more information, see [Section 5.13.1, "About Update Centers"](#).

Instead, when they install your module, your update center will automatically be accessible via the Plugins manager. As a result, when your users go to **Tools > Plugins** manager, they will immediately see your update center in the 'Select Update Center(s) to connect' list. An update center declaration consists of the URL to the autoupdate descriptor and a display name. For more information, see [Section 5.8.3, "How to Bundle an Update Center's URL"](#).

- JavaHelp Help Sets. The IDE provides a number of JavaHelp help sets. You can add your own, or, if you are creating an application on top of the NetBeans Platform, you can provide a help set specifically for your application. (You can also hide help sets provided by other modules.) The files required for a JavaHelp help set can be generated for you by the IDE. Registration of the help set in the IDE can also be automated. As a result, you do not need to think about these infrastructural matters, so that you can focus your time and energy on the *content* of your help set.

For more information, see [Section 5.8.4, "How to Bundle a JavaHelp Help Set"](#).

- License. With the creation of any software, there is often a need for that software to have a software license. The IDE's support for creating modules allows you to add a license to the module that is under creation.

For more information, see [Section 5.8.5, "How to Bundle a License"](#).

## 5.8.1 How to Bundle a Library

You bundle libraries with your module by using the New Java SE Library Descriptor wizard. This wizard creates a new Java SE library descriptor, which registers the library in the Ant Library Manager. For more information, see [Section 5.8, "Bundling Supporting Items"](#).

### Java SE Library Descriptor

A Java SE library descriptor is an XML file that, when registered in the `layer.xml` file, adds a new class library to the IDE's Ant Library Manager. Whenever the module containing the library descriptor is enabled, the library is present in the Ant Library Manager. For example, Struts support provides the Struts libraries in the Ant Library Manager.

To create a new Java SE Library Descriptor:

1. Right-click a module project and choose **New > Java SE Library Descriptor**.
2. In the Select Library page, you have to specify the library that you want to make available in the Ant Library Manager via the module. You can select a library that is available in the Ant Library Manager or you can click Manage Libraries to use the Ant Library Manager to add the library that you want to make available via the module.
3. Click **Next**.
4. In the Name, Icon, and Location page, set the following:
  - **Library Name**. Specifies the name of the new library descriptor.
  - **Display Name**. Specifies the descriptor's label.
5. Click **Finish**.

## 5.8.2 How to Bundle a Project Template or Sample

The New Project Template wizard creates a new project template or project sample. For more information, see [Section 5.8, "Bundling Supporting Items"](#).

You can create an NBM file that contains your template or sample. For more information, see [Section 5.11.1, "About NBM Files"](#).

When the user installs the NBM file, the template or sample will be available in the New Project wizard.

To create a new project template or sample:

1. Right-click a module project and choose **New > Project Template**.
2. In the Select Project page, you have to specify the project that you want to make available as a project template or sample. You can use the Project drop-down list to select an open project or you can click Browse to browse to a project in your filesystem.
3. Click **Next**.
4. In the Name, Icon, and Location page, you have to set the following:
  - **Template Name**. Specifies the name of the new project template.
  - **Display Name**. Specifies the template's label. For example, you will see this label in the Plugins manager. (Optional)
  - **Category**. Specifies the template's category. For example, this will enable the user to find the template more easily in the New Project wizard. (Optional)
  - **Package**. Specifies the name of the package where the classes will be housed.
5. Click **Finish**.
6. After completing the wizard, you can do one or more of the following:
  - Tweak the layer.xml file to change the icon or rearrange the position of the template in the New Project wizard.
  - Change the description that will appear in the New Project wizard's Description box.

## 5.8.3 How to Bundle an Update Center's URL

The New Update Center wizard lets you create a module that registers a update center in the user's Plugins manager. For more information, see [Section 5.13.1, "About Update Centers"](#).

In effect, the module bundles the URL to the update center so that the user does not need to register the URL manually. For more information, see [Section 5.13.4, "Manually Registering an Update Center URL"](#).

To bundle an update center's URL:

1. Right-click a module project and choose **New > Other**. In the New File wizard, choose **Update Center** from the Module Development category. Click **Next**.
  2. In the Update Center Declaration page, you have to set the following:
    - **URL to Update Descriptor**. Specifies the URL to the autoupdate descriptor.
- You can let the IDE generate the autoupdate descriptor for you. For more information, see [Section 5.13.2, "How to Generate an Autoupdate Descriptor"](#).

- **Display Name.** Specifies the name that you would like the user to see in their Plugins manager.

3. Click **Finish**.

#### 5.8.4 How to Bundle a JavaHelp Help Set

A JavaHelp help set provides help files that explain the features and functionality of your module. For more information, see [Section 5.8, "Bundling Supporting Items"](#).

To create a new JavaHelp help set:

1. Right-click a module project and choose **New > Other**. In the New File wizard, choose **JavaHelp Help Set** under the Module Development category.
2. In the Location page, note the files that will be created and also note that they will be housed in a new folder called `docs`:
  - **-about.html**. A sample HTML file that is registered in the `idx.xml` file, `map.xml` file, and `toc.xml` file.
  - **-hs.xml**. Helpset file.
  - **-idx.xml**. Index file. Using the map ID created in the `map.xml` file, you add items to the `idx.xml` file, with the name of the topic that you want displayed in the index.
  - **-map.xml**. Map file. Each HTML file must be registered in the map file. The map ID that you create for the HTML file is used in the `toc.xml` file and `idx.xml` file.
  - **-toc.html**. Table of contents file. Using the map ID created in the `map.xml` file, you add items to the `toc.xml` file, with the name of the topic that you want displayed in the table of contents.
  - **-helpset.xml**. The reference file that is registered in the `layer.xml` file.

**Note:** Each of the names above is prefixed by the name of the project. For example, if the project name is `myproject`, the files above would be `myproject-about.html`, `myproject-hs.xml`, etc.

Optionally, if you do not want to include the IDE's default JavaHelp help sets with your module, you can hide them. Particularly when you are creating a rich-client application on top of the NetBeans Platform, it is unlikely that you will want the IDE's JavaHelp help sets to be included with your application.

To hide a JavaHelp help set:

1. In the Projects window, expand the **Important Files** node.
2. In the **Important Files** node, expand the **XML Layer** node, and then wait a moment while the subnodes are loaded.
3. Expand the `<this layer in context>` node.
4. Within the **Services/JavaHelp** node, select the nodes of the JavaHelp sets that you want to delete. Choose **Delete**.

In the `layer.xml` file, notice that tags have been added, each with a `_hidden` flag. When your module is installed, the `_hidden` flag tells the IDE, or the application built on the NetBeans Platform, to exclude the specified items.

### 5.8.5 How to Bundle a License

Especially when you want to distribute a module, bundling a license is standard procedure. For more information, see [Section 5.8, "Bundling Supporting Items"](#).

A license is typically a plain text file.

To bundle a license:

1. Right-click the module project node and choose **Properties**.
2. In the Project Properties dialog box, click **Packaging**.
3. Specify the packaging information, which includes the license.

## 5.9 Registering Modules

### 5.9.1 About the System Filesystem

The system filesystem is the central repository for configuration data in NetBeans. It is composed at runtime of a stack of XML layer files supplied by modules in the system. For more information, see [Section 5.9.2, "About XML Layer Files"](#).

It is a virtual filesystem that contains configuration information. NetBeans stores a wide variety of configuration information in the system filesystem. For example, the system filesystem contains a folder called **Menu**, which contains subfolders with names such as **File** and **Edit**. These subfolders contain files that represent Java classes which implement the actions that appear in the File and Edit menus.

When you create a module, you are free to create your own folders in the system filesystem to store data that relate to your module. You can also add objects to existing folders. One of the reasons to use the system filesystem is that it enables an application to be constructed piecemeal from pluggable components (modules) without requiring the use of a monolithic "master controller" that knows about everything.

One important aspect of a NetBeans virtual filesystem is that it can fire events to notify the rest of the system when something in it changes. NetBeans listens for changes in the system filesystem, and if, for example, something creates a new object in one of the menu folders, that new item will appear in the menu.

### 5.9.2 About XML Layer Files

Layer files (`layer.xml`) are small XML files provided by modules, which define a virtual filesystem. The layer file defines folders and files that will be merged into the system filesystem that makes up the runtime configuration information that the NetBeans Platform and its derivatives (such as the IDE) use. For more information, see [Section 5.9.1, "About the System Filesystem"](#).

Layer files help to make it possible for modules to be dynamically installed. The components of the IDE whose content is composed from folders in the system filesystem listen for changes in folders and files in a filesystem. If a module is added at runtime, the system filesystem fires changes; the user interface notices that the contents of the folder has changed and updates the user interface to reflect the changes.

New modules created using a module project template (for details, see [Section 5.6.2, "How to Create a Module Project"](#)) do not have an XML layer, but you can add one with the New XML Layer wizard in the New File dialog. You can then expand the

node for it under Important Files in your module project to see and modify its contents. The way it is declared is simple:

- In your jar, provide the layer file, for example, com/foo/mymodule/resources/layer.xml
- In your module's manifest, include the following line somewhere in the top section:

```
OpenIDE-Module-Layer: com/foo/mymodule/resources/layer.xml
```

Just as the New XML Layer wizard creates an empty layer.xml file, so some module file templates may add entries, creating a new layer if necessary (for details, see [Section 5.7.1, "Generating Skeleton API Implementations"](#)). Other templates just add Java annotations and so do not need an XML layer.

You can use the System Filesystem Browser to tweak the layer.xml file, or you can do so manually using code completion in the Source Editor.

### 5.9.3 How to Edit an XML Layer File

When editing an XML layer file, you can use code completion (Ctrl-Space) in the Source Editor. Alternatively, you can use nodes in a tree view—each node represents an item that a module makes available to the system.

To edit an XML layer file:

1. In the Projects window, expand the **Important Files** node.
2. Expand the XML Layer node. After a moment, two subnodes appear:
  - <this layer>. The folders and files provided by the current module project's layer file.
  - <this layer in context>. All the folders and files provided by all the layer files in all modules available in the system.
3. Expand a subnode to explore the folders and files within it. When you right-click a subnode, you can, for example, rename it. You can also use actions such as "Cut", "Copy", and "Paste". In addition, you can drag a subnode and drop it elsewhere, to reorder a folder or a file. All of these actions will result in XML entries being added to your module project's layer file.

### 5.9.4 How to View the System Filesystem

The system filesystem is the general registry for publicly accessible data and objects. For more information, see [Section 5.9.1, "About the System Filesystem"](#). It is a virtual filesystem that contains configuration information. Layer files (layer.xml) define folders and files that will be merged into the system filesystem. For more information, see [Section 5.9.2, "About XML Layer Files"](#).

To view the system filesystem:

1. Create a module project. For more information, see [Section 5.6.2, "How to Create a Module Project"](#).
2. In the Projects window, expand the **Important Files** node.
3. Expand the XML Layer node. After a moment, two subnodes appear:
  - <this layer>. The folders and files provided by the current module project's layer file.

- <this layer in context>. All the folders and files provided by all the layer files in all modules available in the system.
- 4. Expand a subnode to explore the folders and files within it.

When you right-click a subnode, you can, for example, rename it. You can also use actions such as "Cut", "Copy", and "Paste". In addition, you can drag a subnode and drop it elsewhere, to reorder a folder or a file. All of these actions will result in XML entries being added to your module project's layer file.

## 5.10 Communicating Between Modules

For registration and discovery of any kind of interface or class, in any module within the system, the NetBeans Platform supports two uniformly suitable solutions:

- `Lookup`. One of the most fundamental classes in the NetBeans APIs is `org.openide.util.Lookup`.
- `ServiceLoader`. This is the new `java.util.ServiceLoader` class, introduced in JDK 1.6. For more information, see [Section 5.10.1, "About Service Providers"](#).

Factors that might determine which approach to take:

- `Lookup` is available in versions for older JDKs and thus you can use it as a replacement of `ServiceLoader` when running on JDKs older than 1.6.
- `Lookup` is ready to work inside of the NetBeans runtime container. It knows how to discover all the modules in the system, how to effectively read its defined services, etc.
- `Lookup` supports listeners. Client code can attach a listener and observe changes in lookup content. This is a necessary improvement to adapt to the dynamic environment created by the NetBeans runtime container, where modules can be enabled or disabled at runtime, which in turn can affect the set of registered service providers.
- `Lookup` is extensible and replaceable. While the `ServiceLoader` class in JDK 1.6 is a final class with hard-coded behavior, the NetBeans `Lookup` is an extensible class that allows various implementations. This can be useful while writing unit tests. Or you can write an enhanced version of `lookup` that not only reads `META-INF/services` but, for example, finds the requested service providers around the Internet, etc.
- `Lookup` is a general purpose abstraction. While the JDK's `ServiceLoader` can de-facto have just one instance per classloader, there can be thousands of independent `Lookup` instances, each representing a single place to query services and interfaces. In fact this is exactly the way `Lookup` is used in NetBeans—it represents the *context* of each dialog, window element, node in a tree, etc.

For a four-part tutorial series that covers `Lookup`, see NetBeans Selection Management Tutorial I—Using a TopComponent's `Lookup`.

### 5.10.1 About Service Providers

In version 1.3, the JDK started to use a concept called *service providers*. This concept introduces a completely declarative style of registration, which is based just on the current classpath of a Java virtual machine and nothing else. This has an important advantage greatly contributing to the ease of use of this registration style: in order to change the set of registered providers, just pick up a JAR file that offers such a

provider and include it in application classpath. Immediately its provider will be accessible to all code that searches for it.

The basic idea is that each JAR file (in NetBeans terminology, each module) that wishes to provide an implementation of some interface, for example `javax.xml.parsers.DocumentBuilderFactory`, can create its own implementation of the interface, say `org.sakson.MyFactory`, and expose it to the system as a service by creating a `META-INF/services/javax.xml.parsers.DocumentBuilderFactory` file inside of its own JAR file. The file then contains a name of the implementation class per line. In this example it would contain one line registering the sakson factory: `org.sakson.MyFactory`.

The `DocumentBuilderFactory.newInstance` method then searches for all `META-INF/services/javax.xml.parsers.DocumentBuilderFactory` files by using `ClassLoader.getResources("META-INF/services/javax.xml.parsers.DocumentBuilderFactory")`, reads their content, and instantiates the class(es) found there by calling their default constructors. The first implementation of the `DocumentBuilderFactory` is then returned from the `newInstance` method.

While you can manually create the registration of a service in your module, usually you will use the `org.openide.util.lookup.ServiceProvider` annotation, which creates such a registration for you automatically.

As already mentioned, this style has been in place since JDK 1.3 and is a standard way to deal with service providers. Not only has NetBeans adopted this style, it is also gaining in popularity among other Java developers. As a result, JDK 1.6 has introduced the new utility class `java.util.ServiceLoader`.

## 5.11 Building Modules

The IDE uses an Ant build script to build your modules. The IDE generates the build script based on the options you enter in the project's Project Properties dialog box. You can set the module's dependencies, versioning, and packaging information in the Project Properties dialog box. You can further customize program execution by editing the Ant script and Ant properties for the project.

You can customize the build process by doing any of the following:

- Enter basic options, like module dependencies, packaging settings and versioning information in the Project Properties dialog box.
- Customize the IDE-generated Ant targets or create new targets in `build.xml`.

To build a module:

1. In the Projects window, right-click the node of the module project you want to build.
2. Choose **Build Project**.

To build all the modules belonging to a module suite project, right-click the module suite project and choose **Build All**.

If the IDE does not support your desired build process, see the harness/README file. This README file contains detailed descriptions of various file layouts and how to edit them.

### 5.11.1 About NBM Files

An NBM file is a NetBeans module packaged for delivery via the web. For more information, see [Section 5.1, "About NetBeans Modules"](#). The principal differences between NBM files and module JAR files are:

- An NBM file can contain more than one JAR file—modules can package any libraries they use into their NBM file.
- An NBM file contains metadata that NetBeans will use to display information about it in the Plugins manager, such as the manifest contents, the license, etc.
- An NBM file may be signed for security purposes.
- NBM files are just ZIP files with a special extension. They use the JDK's mechanism for signing JAR files. Unless you are doing something unusual, you need not worry about the contents of an NBM file—just let the standard Ant build script for NBM creation take care of it for you.

### 5.11.2 How to Build an NBM File

An NBM file is a NetBeans module in binary format, packaged for delivery via the web. For more information, see [Section 5.11.1, "About NBM Files"](#).

To build an NBM file:

1. In the Projects window, right-click the node of the module project you want to build.
2. Choose **Create NBM**.

To build all NBM files in a module suite:

1. In the Projects window, right-click the node of the module suite project you want to build.
2. Choose **Create NBMs**.

When you choose Create NBMs in a module suite's contextual menu, an autoupdate descriptor is generated by the IDE. The autoupdate descriptor describes all the NBM files. When you put the autoupdate descriptor on a server, your users can access it and retrieve your NBM files via the autoupdate descriptor. For more information, see [Section 5.13.2, "How to Generate an Autoupdate Descriptor"](#).

## 5.12 Trying Out a Module

When you test your module, it is advisable to deploy it to a different instance of the IDE. This way, if there are problems with your module that cause the IDE to crash, you will not lose unsaved work or need to restart the IDE.

However, if you are confident of your module or if you are demonstrating module development, you may want to see the effect of your module more quickly. In these cases, it is worth taking the risk that you will lose unsaved files and have the inconvenience of being required to restart the IDE, if bugs in your module cause the IDE to crash.

To install a module project in a new instance of the IDE:

1. Choose a target platform. For more information, see [Section 5.7.3, "How to Register the NetBeans Sources and Javadoc"](#).
2. Right-click the module project and choose **Run**.

The target platform starts up and installs the module project. By default, the target platform is the IDE in which you are developing your module project.

3. If you wish to test changes to the module without restarting the target platform, right-click the module project and choose **Reload in Target Platform**.

To install a module project in the current instance of the IDE:

1. Save your work.
2. Right-click the module project and choose **Install/Reload in Development IDE**.

### 5.12.1 Deploying Modules

When an application based on a NetBeans Platform is run, the NetBeans Platform's Main class is run. It then finds the available modules and builds an in-memory registry of them, and performs any tasks they specify for startup. Generally, a module's code is loaded into memory as it is needed.

## 5.13 About Distributing Modules

You can distribute a module in one or more of the following ways:

- **Contribute to the Plugin Portal.** This is the central location where you can upload your NetBeans modules so that they may be easily found, all in the same place. For more information, see <http://plugins.netbeans.org/PluginPortal/>.
- **Create your own update center.** With your own update center, you can distribute modules that are specific to your application, as well as modules that are not appropriate for distribution via the Plugin Portal. For more information, see [Section 5.13.1, "About Update Centers"](#).  
For quick updates to your modules or for making modules available to a smaller subset of end users, creating your own update center might be a good idea.
- **Other.** You can simply zip up a module's sources and transfer them via e-mail or upload them onto a server. Alternatively, you can use the IDE to create a binary NBM file, which you can then transfer via e-mail or upload to a server, instead of transferring the sources.

### 5.13.1 About Update Centers

An update center is nothing more than an XML file that is known as the “autoupdate descriptor”, together with the modules that it describes. The autoupdate descriptor lists all the modules that you would like to make available. For each module, the autoupdate descriptor provides information such as a name, a description, and a list of modules that it depends on.

Most importantly, the autoupdate descriptor specifies a URL for each module that it describes. Each module, in binary NBM file format, must live on a server. For more information, see [Section 5.11.1, "About NBM Files"](#).

The autoupdate descriptor itself must also live on a server. The server where the autoupdate descriptor lives need not be the same server as where the modules live. The modules can live together on the same server or be spread across different servers.

After the autoupdate descriptor and its associated modules are on a server, you must let your end users know that they are there. The URL to the autoupdate descriptor must be registered in the end users' IDEs. This can be done in one of two ways:

- **Automatically.** You can provide a module that registers the URL to the update center for them. This module can be generated in the IDE, without any coding on your part.
- **Manually.** You must tell your end user what the URL to your autoupdate descriptor is. Then, the end users need to register the URL to your autoupdate descriptor. They must then do so manually in the Plugins manager.

Once the URL to the update center is registered, your end users can access your modules via the Plugins manager, which they can find under the Tools menu. Not only new modules, but new versions of existing modules can be distributed in this way.

### 5.13.2 How to Generate an Autoupdate Descriptor

An autoupdate descriptor is an XML file that describes the NBM files that you want to make available to your users. For more information, see [Section 5.11.1, "About NBM Files"](#).

For example, an autoupdate descriptor specifies a name, a description, and a URL for each module that it describes.

When you put an autoupdate descriptor on a server, and make its URL available to your users, your users are able to register the URL in your IDE. After registering the URL, they can access your NBM files via the Plugins manager, under the Tools menu.

To generate an autoupdate descriptor:

1. If you have not already done so, create a module suite project. For more information, see [Section 5.6.4, "How to Create a Module Suite Project"](#).
2. In the Projects window, right-click the node of the module suite project you want to build.
3. Choose **Create NBMs**.

The IDE builds the NBM files in the module suite project. The IDE also creates a file called `updates.xml`, which is the autoupdate descriptor. To see it, look in the Files window (Ctrl-2).

4. Tweak the autoupdate descriptor, if needed. For example, customize the `distribution` element for each module, so that the URL to the NBM file is correct.

### 5.13.3 How to Bundle an Update Center's URL

The New Update Center wizard lets you create a module that registers a update center in the user's Plugins manager. For more information, see [Section 5.13.1, "About Update Centers"](#). In effect, the module bundles the URL to the update center so that the user does not need to register the URL manually. For more information, see [Section 5.13.4, "Manually Registering an Update Center URL"](#).

#### To bundle an update center's URL:

1. Right-click a module project and choose **New > Other**. In the New File wizard, choose **Update Center** from the **Module Development** category. Click **Next**.
2. In the **Update Center Declaration** page, you have to set the following:
  - **URL to Update Descriptor.** Specifies the URL to the autoupdate descriptor.

You can let the IDE generate the autoupdate descriptor for you; for details, see [Section 5.13.2, "How to Generate an Autoupdate Descriptor"](#).

- **Display Name.** Specifies the name that you would like the user to see in their Plugins manager.
3. Click **Finish**.

#### 5.13.4 Manually Registering an Update Center URL

If you have not been provided with a module that bundles an update center's URL, you need to register the update center manually. For more information, see [Section 5.8.3, "How to Bundle an Update Center's URL"](#).

Once you have registered the update center, you can access its modules via the Plugins manager, which you can access under the **Tools** menu.

To manually register an update center's URL:

1. Choose **Tools > Plugins**.  
The Plugins manager opens.
2. Click the **Settings** tab at the top of the Plugins manager.
3. Click **Add**.
4. Type the name and URL of the autoupdate center. The name can be anything you want it to be. Click **Finish**.
5. Click **Close**.

You have now manually registered the update center's URL.

### 5.14 Branding a Rich-Client Application

Once the functionality of an application built on top of the NetBeans Platform is complete, the appearance of the application still closely resembles the default NetBeans Platform. You can use the IDE to personalize your application. In particular, you can do the following:

- Replace the application's splash screen.
- Replace the strings in the NetBeans Platform's bundle files.
- Define a progress bar.
- Define the application's title bar.
- Disable one or more of the window system's default features.

To brand a rich client application:

1. Right-click the application project in the Projects window and choose **Branding**.
2. In the Branding editor, click the tab related to your branding needs:
  - **Basic.** Brand the title bar and the application icons.
  - **Splash Screen.** Brand the splash screen and progress bar.
  - **Window System.** Remove features from the window system.
  - **Resource Bundles.** Customize the strings provided by the NetBeans Platform.
3. Click **OK** to confirm your changes and exit the editor.

### 5.14.1 How to Brand the Window System

You can brand an application's window system by disabling the following of its default features:

- **Window Drag and Drop.** The ability to reorganize the window layout by dragging windows to new positions.
- **Floating Windows.** The ability for windows to be undocked into a standalone frames.
- **Sliding Windows.** The ability to minimize (slide out) a window.
- **Maximized Windows.** The ability to maximize a window by clicking its header.
- **Closing Windows.** The ability to close a document/non-document window.
- **Window Resizing.** The ability to adjust the width and height of windows by dragging splitter bars.
- **Respect Minimum Size When Resizing Windows.** The ability to resize internal windows to zero width/height using splitter bars.

To brand the window system:

1. Right-click the application project in the Projects window and choose **Branding**.
2. In the Branding editor, click **Window System**.
3. Deselect the items listed above, depending on your needs.
4. Click **OK** to confirm your choices and exit the Branding editor.

### 5.14.2 How to Add a Splash Screen

1. Right-click the application project in the Projects window and choose **Properties**.
2. In the Project Properties dialog box, click **Splash Screen**. Do the following:
  - Define the progress bar.
  - Choose a splash screen.
3. Click **OK** to confirm your choices and exit the Project Properties dialog box.

### 5.14.3 How to Remove Unwanted Modules

You can exclude unwanted modules from a standalone application, leaving only those you really need. By default, all the modules that the IDE uses are included.

To remove unwanted modules:

1. Right-click a module-suite project and choose **Properties**.
2. In the Project Properties dialog box, click **Libraries**.
3. Uncheck all the clusters that you want to exclude from the standalone application. The only cluster that all standalone applications need is `platformNNN`.
4. Expand the nodes of the clusters that you want to keep, and uncheck any module that you want to exclude from the cluster.
5. Click **OK**.

When you brand a standalone application (for details, see [Section 5.14, "Branding a Rich-Client Application"](#)), the IDE asks you whether it should exclude IDE-related

modules for you. If you do not let the IDE exclude IDE-related modules, you can do so manually using the steps above.

## 5.15 Distributing Rich-Client Applications

Rich-client applications are complete, functioning, standalone Swing applications. The Swing libraries provide a rich collection of user interface elements. However, the Swing libraries do not provide a mechanism for joining the user interface elements together into an application. For this purpose, NetBeans provides the NetBeans Platform, which is the application framework on top of which you build your application. For more information, see [Section 5.2, "About the NetBeans Platform"](#).

Each distinct part of a rich-client application is provided by a separate module, several of which serve to provide the user interface elements from the Swing libraries. For example, if your rich-client application is an editor, you might have one module that provides syntax highlighting, while another provides file templates.

### Branding

Before you distribute a rich-client application, you need to consider whether you want to leave it resembling NetBeans. For example, your rich-client application uses the NetBeans splash screen by default. Branding, the final stage before creating distribution packages, involves making decisions such as what the splash screen should look like and whether the application will include a progress bar during startup. In the module suite project's Project Properties dialog box, you define such settings. For more information, see [Section 5.14, "Branding a Rich-Client Application"](#).

While branding, also consider whether your rich-client application needs all the modules that the IDE uses. For example, if your rich-client application is not an editor, you will not need the modules that relate to editor functionality. Similarly, it is unlikely that all of the IDE's menu items and toolbar buttons are needed by your application.

### Releasing

Once a rich-client application is branded, you can distribute it over the web as a web-startable JNLP application. Alternatively, you can distribute the ZIP file. See [Section 5.15.2, "How to Build a JNLP Application"](#) and [Section 5.15.1, "How to Build a ZIP Distribution"](#) for details. Updates to the modules that make up a rich-client application can be distributed via the Update Center.

### 5.15.1 How to Build a ZIP Distribution

Once your rich-client application is complete, one way to distribute it is via a ZIP file. The IDE can create the ZIP file for you, containing the application.

To build a ZIP distribution:

1. In the Projects window, right-click the node of the application.
2. Choose **Package as | ZIP Distribution**.

### 5.15.2 How to Build a JNLP Application

Java Web Start is a helper application that becomes associated with a Web browser. When a user clicks on a link that points to a special launch file (JNLP file), it causes the browser to launch Java Web Start, which then automatically downloads, caches, and runs the given Java Technology-based application. The entire process is typically completed without requiring any user interaction, except for the initial single click.

To build a JNLP application:

1. In the Projects window, right-click the node of the module suite project you want to build.
2. Choose **JNLP | Build**.
3. Depending on whether you have set up the module suite project as a standalone application, do the following:
  - If you have already set up the module suite project as a standalone project, the IDE builds a `jnlp` folder in your `build` folder and adds a `master.jnlp` file in your main project folder. Open the Files window to see the JNLP file and the folder within the `build` folder.
  - If you have not set up the module suite project as a standalone application, the IDE will not be able to build the JNLP application. The IDE will tell you this by means of a dialog box.
    - Click **Configure Application** in the dialog box. The Project Properties dialog box opens. Click **Build**.
    - Select the **Create Standalone Application** radio button. Click **OK**.
    - In the Projects window, right-click the node of the module suite project you want to build.
    - Choose **Build JNLP Application**.

The IDE builds a `jnlp` folder in your `build` folder and adds a `master.jnlp` file in your main project folder. Open the Files window to see the JNLP file and the folder within the `build` folder.

### 5.15.3 How to Run a Rich-Client Application

Unless your NetBeans modules are designed to be deployed as standalone modules, you should always deploy them by deploying the application (the module suite project) that contains them. If you run the Run Project command on an individual module that is part of an application built on top of the NetBeans Platform, the IDE only deploys that module itself.

To deploy a standalone application:

- In the Projects window, right-click the module suite project and choose one of the following:
  - **Run Project**. Starts the target platform, undeploys the standalone application if it is already deployed, deploys the standalone application to the target platform, starts the platform and loads the application as one unit.
  - **Run JNLP Applications**. Builds and runs the application as a JNLP application. For more information, see [Section 5.15.2, "How to Build a JNLP Application"](#).

### 5.15.4 How to Use the NetBeans Shared JNLP Repository

By default, the JNLP application generated for a module suite project always contains all the module suite project's modules as well as all the modules that the module suite project depends on. This may be useful for intranet usage, but it is a bit less practical for wide internet use. When on the internet, it is much better if all the applications built on the NetBeans Platform refer to one repository of NetBeans modules, which means that such modules are shared and do not need to be downloaded more than once.

There is such a repository for NetBeans modules. It does not contain all the modules that NetBeans IDE has, such as, for example, the ant module, which is not JNLP-ready, but it contains enough to make many JNLP applications possible.

To use the shared JNLP repository:

1. In the Projects window, expand the **Important Files** node.
2. Double-click **NetBeans Platform Config**.
3. Add the jnlp.platform.codebase property with, as its value, the URL of the 5.0 JNLP Repository:

```
jnlp.platform.codebase=http://www.netbeans.org/download/5_0/jnlp
```

# 6

---

## Creating Java Projects

This chapter describes creating and working with standard and free-form Java projects, including using projects using templates.

This chapter contains the following sections:

- [About Creating Java Projects](#)
- [Using Java Project Templates](#)
- [Importing an Eclipse or JBuilder Project](#)
- [Setting the Main Project](#)
- [Adding Multiple Sources Roots to a Project](#)
- [Sharing a Library with Other Users](#)
- [Adding a Javadoc to a Project](#)
- [Setting the Target JDK](#)
- [Moving, Copying, and Renaming a Project](#)
- [Deleting a Project](#)

### 6.1 About Creating Java Projects

A project is a group of source files and the settings with which you build, run, and debug those source files. In the IDE, all Java development has to take place within a project. For applications that involve large code bases, it is often advantageous to split your application source code into several projects.

The IDE includes several project templates designed to support different types of development including web applications, general Java applications, and so forth. The IDE's set of standard project templates automatically generate an Ant script and properties. The IDE also contains free-form project templates that you can use to base a project on an existing Ant script. In addition to Ant, the IDE also supports Maven, an open source build management tool. Maven uses a project object model (POM) that describes a set of standards that all projects using Maven follow allowing for consistency between projects. You can easily create a Maven project by choosing a Maven project template and providing a few project details.

---

**Note:** You do not need an in-depth understanding of Ant or Maven to work with the IDE as the provided set of standard project templates automatically generate an Ant script or Maven POM files based on the options you enter in the IDE.

---

For more information about Ant, see <http://ant.apache.org/>.

This chapter focuses primarily on the use of Ant scripts to build a project. For information on using Maven to build and manage a project in JDeveloper, see "Working with Maven in the IDE" and "Working with Maven Repositories." For more information about Maven, see <http://maven.apache.org/>.

When you finish creating a project, it opens in the IDE with its logical structure displayed in the **Projects** window and its file structure displayed in the **Files** window:

- The **Projects** window is the main entry point to your project sources. It shows a logical view of important project contents such as Java packages and Web pages. You can right-click any project node to access a context menu of commands for building, running, and debugging the project, as well as opening the **Project Properties** dialog box. The **Projects** window can be opened by choosing **Window > Projects**.
- The **Files** window shows a directory-based view of your projects, including files and folders that are not displayed in the **Projects** window. From the **Files** window, you can open and edit your project configuration files, such as the project's build script and properties file. You can also view build output like compiled classes, JAR files, WAR files, and generated Javadoc documentation. The **Files** window can be opened by choosing **Window > Files**.

---

**Note:** If you need to access files and directories that are outside of your project directories, you can use the **Favorites** window. You open the **Favorites** window by choosing **Window > Favorites**. You add a folder or file to the **Favorites** window by right-clicking in the **Favorites** window and choosing **Add to Favorites**.

---

## 6.2 Using Java Project Templates

You can facilitate creating a Java application by using one of the available templates provided by the IDE. For each type of Java application, the IDE provides two types of project templates:

- **Standard templates.** Templates in which the IDE controls all source and classpath settings, compilation, running, and debugging.
- **Free-form templates.** Templates in which your own Ant script controls all classpath settings, compilation, running, and debugging.

### 6.2.1 Standard Project Templates

With standard project templates, the IDE controls all aspects of how your application is built, run, and debugged. You set a project's source folder, classpath, and other project settings when creating the project and in the Project Properties dialog box. The IDE generates an Ant build script in which all of your settings are stored.

The IDE comes with the following standard templates:

-  **Standard Java Applications:**
  - Java Application - An empty Java SE project with a main class.
  - Java Class Library - An empty Java class library with no main class.
  - Java Project with Existing Sources - A Java SE project with existing sources.
-  **JavaFX Applications:**

- JavaFX Application (without FXML) - An empty JavaFX application with no main class.
- JavaFX Application With FXML - An empty JavaFX FXML-enabled application.
- JavaFX Preloader - A JavaFX application with a preloader to facilitate loading the application particularly in applet or webstart mode.
- JavaFX Swing Application - A JavaFX application enabled with Swing and a main class containing sample JavaFX-n-Swing code.
-  **Web Applications:**
  - Web Application - An empty web application.
  - Web Application with Existing Sources - A web application with existing sources.
-  **EJB Modules:**
  - EJB Module - An empty EJB module.
  - EJB Module with Existing Sources - An EJB module with existing sources.
-  **Enterprise Applications:**
  - Enterprise Application - An empty enterprise application.
  - Enterprise Application with Existing Sources - An enterprise application with existing sources that conform to the Sun Java BluePrints Guidelines.
-  **Enterprise Application Clients:**
  - Enterprise Application Client - An empty enterprise application client.
  - Enterprise Application Client with Existing Sources - An enterprise application client with existing sources.
-  **NetBeans Modules:**
  - Module Project - An empty module with a `layer.xml` file and a `Bundle.properties` file. You use a module to implement the logic that integrates the library wrappers into the platform and provides a user interface for receiving user input.
  - Module Suite Project - A library wrapped in a module project and a `Bundle.properties` file. You use a library wrapper to put a library JAR file on a module's classpath and export some or all of the JAR file's packages from the module as public packages.
  - Library Wrapper Module Project - An empty module suite. You use a module suite to group and deploy a set of interdependent modules and library wrappers.

Depending on what modules you have installed, your IDE may contain additional templates. Consult the help for your additional modules for more information.

For information on creating an Web Application Project, see [Chapter 12, "Developing Web Applications."](#)

For information on creating an Enterprise Application Project, see [Chapter 14, "Developing Enterprise Applications."](#)

For information on creating an EJB Module Project, see [Chapter 16, "Developing with Enterprise Beans."](#)

### 6.2.1.1 Source Folders

In standard projects, you can have multiple source folders and multiple JUnit test folders for each Java SE, web, and EJB project.

Right-click the Source Packages node of a project and choose **Properties** to add or remove source folders. A single source folder cannot, however, be added to more than one project. If you need a source folder to be part of several projects, you should create a separate project for the source folder and add it to the classpath of each of your projects.

For information on add multiple sources to a project, see [Section 6.5, "Adding Multiple Sources Roots to a Project."](#)

### 6.2.1.2 Project Settings

When you create a project from a standard project template, the IDE generates an Ant script that controls all aspects of program compilation, execution, and debugging.

Right-click a project node in the Projects window and choose **Properties** to set basic project settings in the Project Properties dialog box. All changes are immediately registered in the project's Ant script. You can also set more complex options by editing the project's Ant script and properties file directly.

For information on editing Ant build scripts, see [Section 6.2.3.3, "Editing IDE-Generated Ant Scripts."](#)

### 6.2.1.3 Switching a Java SE Project to a JavaFX Deployment Model

If you have an existing standard Java SE project and you want to take advantage of the JavaFX deployment model, you can quickly switch your current project by choosing **Properties** from the project's context menu, selecting the Deployment category, and clicking **Switch Project to JavaFX Deployment Model**. This feature saves you the time consuming effort of creating a new JavaFX application and manually copying over the project artifacts to the new project and resetting options, removing the main class, and so on.

Instead of a `main` class, JavaFX projects use an extension to the `Application` class. When you switch the project, the IDE creates a `project_nameFX.java` class and automatically adds it to the Source Packages folder of your SE project. You then have the option of using JavaFX customizations, such as specifying a JavaFX runtime version or adding custom manifest entries through the Project Properties dialog box.

---

**Note:** Be aware that once a Java SE project has been switched to a JavaFX deployment model, it cannot be automatically reverted.

---

### 6.2.1.4 Adding a JavaFX Class to a Java SE Project

You can add a JavaFX class to a Java SE project without having to manually add a JavaFX runtime dependency. This feature is particularly advantageous for making a Java class library project usable for JavaFX development. When you select the **Keep JavaFX RT artifacts on Compile Classpath if not present by default** option from the Deployment category of the Project Properties dialog box, the IDE automatically updates the `jfxrt.jar` file dependency. If you later change the project platform, the dependency is corrected for you.

### 6.2.1.5 Project Folders

Each standard project has a project folder where the IDE stores the Ant script, project metadata, and output folders. In projects with existing sources, you can place the project source directories in the same location as the project folder or in a separate location. In empty projects, the source root is always in the same location as the project directory.

**Table 6–1 Standard Project Folder Contents**

Item	Description
build.xml	The build script called by the IDE. This build script only contains an import statement that imports targets from nbproject/build-impl.xml. Use the build.xml to override targets from build-impl.xml or to create new targets.
nbproject	The directory that contains the project Ant script and other metadata. This directory contains: <ul style="list-style-type: none"> <li>▪ build-impl.xml. The IDE-generated Ant script. You should never edit build-impl.xml directly. Always override its targets in build.xml.</li> <li>▪ project.properties. Ant properties used by the IDE to configure the Ant script. Although you can edit this file manually, you generally do not have to, as it is automatically updated by the IDE when you configure the project's properties.</li> <li>▪ project.xml and genfiles.properties. IDE-generated metadata files. Although you can edit project.xml manually, for standard projects it is generally not necessary. Never edit genfiles.properties.</li> </ul>
nbproject/private	The directory that holds properties that are defined for you only. If you are sharing the project over VCS, any properties you define in private.properties are not checked in with other project metadata and are applied only to your installation of the IDE.
build	The output directory for compiled classes.
dist	The output directory of packaged build outputs (JAR files and WAR files). Standard Java projects produce one build output per project. The dist directory also contains generated Javadoc documentation.

For information on setting the classpath for standard projects, see [Section 6.2.3.1, "Managing the Classpath."](#)

### 6.2.2 Free-Form Templates

With free-form project templates, the IDE relies on your existing Ant script for instructions on how to compile, run, and debug your applications. The settings you configure in the New Project wizard when creating a project as well as in the Project Properties dialog box are used to tell the IDE how your Ant script manages your source code and must be consistent with the settings in your Ant script.

For example, all classpath elements are handled by your Ant script. When you declare the classpath for a free-form project, you are only telling the IDE which classes to make available for code completion and refactoring. These settings do not affect the actual classpath used when compiling or running your source code.

Free-form projects can contain as many source folders as your Ant script is configured to handle. If your Ant script does not contain targets for all IDE actions, like debugging and running your project, you can easily write Ant targets for these actions.

The standard distribution of the IDE contains the following  free-form Java project templates:

- **Java Free-Form Project.** A free-form project containing one or more Java source roots.
- **Web Free-Form Project.** A free-form project containing a web application and optionally other Java source roots.

### 6.2.2.1 Source Folders

Free-form Java projects can contain multiple source folders. The classpath relationships and handling instructions for your source directories must be handled in your Ant script.

### 6.2.2.2 Project Settings

In the project's properties, you declare the source folders, classpath, and output files for your project. Each source root can have a unique classpath and output file. These settings do not affect the actual contents or classpath of your application. The settings only tell the IDE how to handle the code. For example, the classpath settings tell the IDE which classes to make available for code completion.

### 6.2.2.3 IDE Commands and Ant Targets

In free-form projects, the IDE relies on your Ant script to provide targets for all IDE actions, such as running, debugging, and generating Javadoc. If your Ant script does not contain targets for these commands, the commands are disabled.

You can write debug targets in your Ant script or in a supplementary Ant script. You can also add a shortcut to any of your Ant script's targets to the contextual menu of your project's node in the Projects window.

**6.2.2.3.1 Creating a Target to Compile a Single File** If you want to be able to select files in the IDE and compile them individually, you need an Ant target for the Compile File command. The IDE offers to generate a target the first time you choose the command. The generated target looks something like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<project basedir=".." name="MyProjectName">
    <!-- TODO: edit the following target according to your needs -->
    <!-- (more info: https://netbeans.org/kb/archive/index.html) -->
    <target name="compile-selected-files-in-src">
        <fail unless="files">Must set property 'files'</fail>
        <!-- TODO decide on and define some value for ${build.classes.dir} -->
        <mkdir dir="${build.classes.dir}" />
        <javac destdir="${build.classes.dir}" includes="${files}" source="1.5"
srcdir="src" />
    </target>
</project>
```

In the generated target, you need to specify the directory where to put the compiled class or classes. You can do so by specifying a value for the `build.classes.dir` property in the generated target. For example, you might add the following line to the line above the `<target name="compile-selected-files-in-src">` entry:

```
<property name="build.classes.dir" value="build" />
```

Alternatively, you can replace the value of the provided build.classes.dir or rewrite the target entirely.

The value of the includes parameter is the value of the generated files property. The IDE uses this property to store the name of the currently selected file (or files).

---

**Note:** You can configure multiple compile.single actions to overload the F9 shortcut and menu command with different functionality depending on what file is selected. For example, you could set up a separate compile-selected-files target for JUnit test classes, then map compile.single to that target for all sources in JUnit test directories. Or you could change the pattern to \.xml\$ and map F9 to a Validate XML target for all XML files.

---

**6.2.2.3.2 Writing a Target for the Apply Code Changes Command** The Apply Code Changes command allows you to make changes to your code during a debugging session and continue debugging with the changed code without restarting your program. The IDE contains a nbjpdareload task that you can use to write a target for the Apply Code Changes command.

A typical target for the fix command looks something like this:

```
<target name="debug-fix">
  <javac srcdir="${src.dir}" destdir="${classes.dir}" debug="true" >
    <classpath refid="javac.classpath"/>
    <include name="${fix.file}.java"/>
  </javac>
  <nbjpdareload>
    <fileset dir="${classes.dir}">
      <include name="${fix.file}.class"/>
    </fileset>
  </nbjpdareload>
</target>
```

- The target compiles the currently selected file using the \${fix.file} property. (In the next section you will set up the IDE to store the name of the currently selected file in this property.)
- The nbjpdareload task reloads the corrected file in the application.

To hook this target up to the Apply Code Changes command (the same as the Fix command in previous versions of the IDE), define the following action in <ide-actions> in project.xml:

```
<action name="debug.fix">
  <target>debug-fix</target>
  <context>
    <property>fix.file</property>
    <folder>${src.dir}</folder>
    <pattern>\.java$</pattern>
    <format>relative-path-noext</format>
    <arity>
      <one-file-only/>
    </arity>
  </context>
</action>
```

- <property> now stores the context in the fix.file property.
- Since you can only run the Fix command on one file at a time, you set <arity> to <one-file-only>.

- You have to pass the full path to the .java file to the javac task and the full path to the .class file to the nbjpdareload task. You therefore set the <format> to rel-path-noext, then append .class or .java in the debug-fix target as necessary.

---

**Note:** The IDE does not define the \${src.dir} property for you. You have to define the property or import the .properties file that the Ant is using in project.xml. See [Section 6.2.4.9.1, "Using Properties in the project.xml File"](#) for more information.

---

For information on creating free-form projects in the IDE, see [Section 6.2.4, "Creating Free-Form Projects."](#)

### 6.2.3 Creating Standard Projects

You can facilitate creating a Java application by using one of the available templates provided by the IDE.

**To create a Java project:**

1. Choose **File > New Project** (Ctrl+Shift+N).
2. Select the appropriate template for your project.
3. Follow the steps in the remainder of the application wizard.

Once you create your project, you can configure the classpath and add a JAR file, library, or an IDE project to the classpath as needed. Set your project as the main project to build it and then run it. If necessary, set the main class and any arguments for your project prior to running it. Afterwards, you can debug your project by setting breakpoints or watches and running the IDE's debugging tool.

---

**Note:** Maven uses repositories that contain build artifacts and project dependencies. To create a Maven project, you need to configure Maven settings at both the IDE level and at the project level. For information on creating a Maven project, see [Section 8.12, "Working with Maven in the IDE."](#)

---

#### 6.2.3.1 Managing the Classpath

Adding a group of class files to a project's classpath tells the IDE which classes the project should have access to during compilation and execution. The IDE also uses classpath settings to enable code completion, automatic highlighting of compilation errors, and refactoring.

Source roots must only exist in a single project and cannot be shared with other projects, regardless of whether they are opened or not. If you have to use a library in several projects, create a special project within which to store it.

##### 6.2.3.1.1 Classpath and Standard Projects

For standard projects, the IDE maintains separate classpaths for compiling and running your project, as well as compiling and running unit tests. The IDE automatically adds everything on your project's compilation classpath to the project's runtime classpath. Also the project's compiled test files and everything on the tests' compilation classpath are added to the test runtime classpath.

Whenever you build a standard project for which a main class is specified, the IDE automatically copies any JAR files on the project's classpath to the dist/lib folder. The IDE also adds each of the JAR files to the Class-Path element in the application JAR's manifest .mf file. This simplifies running the application outside the IDE. For more information, see [Section 8.9, "Preparing a JAR File for Deployment Outside the IDE."](#)

#### To edit a standard project's classpath:

1. Expand the project node, right-click the Libraries node, and choose **Properties**.
2. In the Project Properties dialog box, add the necessary elements to the project's compilation classpath by clicking the appropriate button. You can add any of the following:
  - **Project.** The JAR file or WAR file produced by another project, as well as the associated source files and Javadoc documentation.
  - **Library.** A collection of JAR files or folders with compiled classes, which can optionally have associated source files and Javadoc documentation.

---

**Note:** If you have attached Javadoc and source files to a JAR file in the Library Manager, the IDE automatically adds the Javadoc and source files to the project when you register the JAR file on a project's classpath. You can step into classes and look up Javadoc pages for the classes without configuring anything else.

---

- **JAR/Folder.** A JAR file or folder that contains compiled classes.
3. (Optional) In web applications, click the **Deploy** checkbox if you do not want to package an element in the web application. By default, all classpath elements are included in the web application.
  4. (Optional) Click the **Build Projects on Classpath** checkbox if you do not want to rebuild all projects on the classpath whenever you build the current project. By default, all projects on the classpath and, in web applications, projects listed in the Packaging page, are rebuilt when you build the current project.
  5. (Optional) Click the **Move Up** and **Move Down** buttons to alter the classpath priority.
  6. (Optional) Click the **Run**, **Compile Tests**, or **Run Tests** tabs to make any changes to the these classpaths.

#### 6.2.3.1.2 Classpath and Free-form Projects

In free-form projects, your Ant script handles the classpath for all of your source folders. The classpath settings for free-form projects only tell the IDE what classes to make available for code completion and refactoring. To change a free-form project's actual compilation or runtime classpath you must edit your build.xml file directly. For information on setting runtime arguments, see [Section 10.6, "Setting the Main Class and Runtime Arguments."](#)

Free-form projects do not have Library nodes nor do free-form project's Project Properties dialog boxes include a Libraries panel. For more information, see [Section 6.2.4.5, "Declaring the Classpath in a Free-Form Project."](#)

---

**Note:** The project's classpath declaration must exactly match the classpath used by your Ant script.

---

Adding a free-form project to the classpath of a standard project does nothing unless you also declare the free-form project's build outputs in the Output page of its Project Properties dialog box.

For information on specifying the target JDK to use for your project, see [Section 6.8, "Setting the Target JDK."](#)

#### 6.2.3.1.3 Adding Annotation Processors to the Classpath

In addition to other libraries, you can add annotation processors to the classpath or the processor path of your project. The annotation processors are specified as either a library, a JAR file, or another NetBeans IDE project. You can add annotation processors on the following tabs: **Compile** and **Processor**. Additionally, you can specify the FQN (fully qualified name) of the annotation processor on the **Compile** tab.

##### To add an annotation processor to the classpath:

1. Right-click the project node and choose **Properties**.
2. Select the **Libraries** category in the Project Properties window.
3. Select one of the following tabs in the Libraries panel and add the project, library or JAR that contains the annotation processor to the project's classpath.
  - **Compile tab.** When an annotation processor is packaged together with its annotations as a single JAR file, specify this JAR file on the **Compile** tab. In this case, there is no need to add it to the **Processor** tab. The resources added on the **Compile** tab correspond to the `-classpath` option of the Java compiler.
  - **Processor tab.** When an annotation processor and its annotations are packaged into separate JAR files, add the JAR file with the processor on the **Processor** tab and the JAR file with the annotations on the **Compile** tab. The resources added on the **Processor** tab correspond to the `-processor` path option of the Java compiler.
4. Select the **Compiling** node under the Build category and select **Enable Annotation Processing** and **Enable Annotation Processing in Editor**.
5. Click Add and type the fully-qualified name of the annotation processor in the Add Annotation Processor dialog box.

#### 6.2.3.2 Creating Dependencies Between Projects

When creating Java applications you typically set up a single main project, which contains both the project main class and any required projects. Whenever you create separate standard project for each of a project's source roots, you have to set up the classpath dependencies between the main project and the required projects. A required project is a project that has been added to another project's classpath.

For standard projects that have a main class specified, the IDE automatically copies any JAR files on the project's classpath to the `dist/lib` folder when you build the project. The IDE also adds each of the JAR files to the `Class-Path` element in the application JAR's `manifest.mf` file. This simplifies running the application outside the IDE. For more information, see [Section 8.9, "Preparing a JAR File for Deployment Outside the IDE."](#)

Whenever you clean and build a project, the IDE also cleans and builds its required projects. The required project's attached Javadoc and source code are also made available to the project that you are building. For information on adding Javadoc, see [Section 6.7.1, "How to Add a Javadoc to a Project."](#)

If you want to add a free-form project to the classpath of a standard project, you have to declare all of the JAR files that are produced when the free-form project is built. Right-click the free-form project's node in the Projects window and choose **Properties**. Then specify those JAR files in the Output page of the dialog box.

#### To configure project dependencies:

1. In the Projects window, expand the node of the project to which you want to add a dependent project.
2. Right-click the Libraries node and choose **Add Project**.
3. Select the directory containing the project whose JAR files you want to add to the classpath. When you select the project directory, the project name and project JAR files are displayed in the dialog's right pane.

The file chooser depicts IDE project directories using  project folder icons.

4. Click **Add Project JAR Files**.

The IDE adds the dependent project to the primary project's classpath and display a node for it within the Projects window's Library node.

In free-form projects, the classpath is defined in your Ant script. For more information, see [Section 6.2.4.5, "Declaring the Classpath in a Free-Form Project."](#)

For information on modifying the runtime classpath, see [Section 10.5, "Setting the Runtime Classpath."](#)

#### 6.2.3.3 Editing IDE-Generated Ant Scripts

If you are working with a standard project and want to set more project options than are available in the Project Properties dialog box, you can edit the project's Ant script and Ant properties. The Ant targets and properties are defined in the following files in your project folder:

- nbproject/build-impl.xml. The Ant script that contains all of the instructions for building, running, and debugging the project. Never edit this file. However, you can open it to examine the Ant targets that are available to be overridden.
- build.xml. The main Ant script for the project. The IDE calls targets in this Ant script whenever you run IDE commands. build.xml contains a single import statement that imports targets from build-impl.xml. In build.xml, you can override any of the targets from build-impl.xml or write new targets.
- nbproject/project.properties. The Ant properties file that contains important information about your project, such as the location of your source and output folders. You can override the properties in this file.
- nbproject/private/private.properties. The Ant properties file that contains properties specific to your installation of the IDE. For example, if you are sharing the project over VCS, any properties you set in private.properties are not checked into the repository. You can copy a property from project.properties into private.properties and give the property different definitions in each file. The definitions in private.properties take precedence over those in project.properties.

---

**Note:** In a free-form project, there is no build-impl.xml. The IDE directly calls targets in the project's Ant script.

For NetBeans Plugin Module projects, build-impl.xml imports targets from the suite.xml in your target platform's harness folder.

---

For information on editing and running an Ant script, see [Section 8.2.2, "How to Edit an Ant Script"](#) and [Section 8.2.3, "How to Run an Ant Script,"](#) respectively.

#### 6.2.3.4 Customizing the IDE-Generated Ant Script

You can edit the Ant scripts generated for standard projects to fine tune the way your project is built and run.

##### To override a target from an IDE-generated build script:

- Copy the target from build-impl.xml or suite.xml to build.xml and make any changes to the target.

---

**Note:** You might need to disable the **Compile on Save** option in project properties, because when enabled it causes some commands to skip calling the ant targets. To disable the **Compile on Save** option, right-click your project in the Projects window, select **Properties**, and clear the **Compile on Same** checkbox in the Compiling category.

---

##### To add instructions to be processed before or after an Ant target is run:

- Override the -pre or -post target for the target.

For example, to insert an obfuscator after compilation, type the following in build.xml:

```
<target name="-post-compile">
    <obfuscate>
        <fileset dir="${build.classes.dir}" />
    </obfuscate>
</target>
```

##### To add a new target to the build script:

1. Create the new target in build.xml.
2. Optionally, add the new target to the dependencies of any of the IDE's existing targets. Override the existing target in build.xml then add the new target to the existing target's depends property.

For example, the following adds the new-target target to the run target's dependencies:

```
<target name="run" depends="new-target,init,compile"/>
```

Copy the dependencies exactly as they exist in build-impl.xml or suite.xml. Notice that you do not have to copy the body of the run target into build.xml.

---

**Note:** The output folder is deleted every time you clean your project. Therefore, never set the output folder to the same location as your source folder without first configuring the clean target to not delete the output folder.

---

For information on editing and running an script, see [Section 8.2.2, "How to Edit an Ant Script"](#) and [Section 8.2.3, "How to Run an Ant Script,"](#) respectively.

For information on setting JVM arguments and system properties, see [Section 10.7, "Setting JVM Arguments."](#)

### 6.2.3.5 Working with a JavaFX Project

To work with JavaFX in NetBeans IDE, you must have the following:

- Installed JavaFX 2 SDK and Runtime
- A JavaFX-enabled Java Platform within the IDE

#### 6.2.3.5.1 JavaFX Application and JavaFX FXML Applications

In a JavaFX Application without FXML, graphic layout and actions are both defined in Java classes. By default, both are defined in the main Application class.

In a JavaFX FXML application, graphic layout is defined in an FXML file and actions are defined in a controller Java class. The controller class implements the interface `javafx.fxml.Initializable`.

#### 6.2.3.5.2 JavaFX Preloaders

A preloader is a small application that is started before the main application to customize the startup experience. You can create JavaFX preloaders either by running the New JavaFX Preloader wizard or by selecting **Create Custom Preloader** in the New JavaFX Application wizard.

#### 6.2.3.5.3 Editing JavaFX Applications

Edit JavaFX Java source code as you edit any other Java source code in the IDE. For more information, see [Section 7.2, "Editing Java Code."](#)

Edit FXML files in one of two ways, depending on whether you have SceneBuilder installed:

- If SceneBuilder is installed, go to the Projects window and either double-click the FXML file's node or right-click the FXML file's node and select Open. SceneBuilder opens automatically, focused on that FXML file. For information about installing and using SceneBuilder, see the JavaFX SceneBuilder documentation at <http://docs.oracle.com/javafx/index.html>.
- If SceneBuilder is not installed, edit the FXML file as you edit any XML document in the IDE. If SceneBuilder is installed, you can choose to use the IDE's XML editor instead by right-clicking the FXML file's node and selecting Edit. For more information, see [Section 18.2, "Creating and Editing XML Documents."](#)

#### 6.2.3.5.4 Building and Running JavaFX Applications

Build and run a JavaFX application as you build and run a standard Java application, with the difference that you can configure a JavaFX project to run in the following ways:

- **Standalone application.** Project builds and runs as a standard Java desktop application.
- **Java Web Start.** Application uses JNLP. The IDE generates the JNLP page when it builds the application.
- In Browser. The IDE embeds the application in a browser, using an HTML page that calls a JNLP page. The IDE can generate the HTML page or you can use your own web page. If you run the application from the IDE, the IDE opens a browser page that points to the HTML page on your local file system.

To set how a JavaFX application is run, open the Project Properties and go to the Run category. To open Project Properties, right-click the project's node in the Projects window and select Properties from the context menu. For more information on packaging a JavaFX application, see the chapter Packaging Basics in the JavaFX Documentation at:

<http://docs.oracle.com/javafx/2/deployment/packaging.htm>

#### 6.2.3.5.5 Debugging JavaFX Applications

Debug JavaFX applications as you would debug Java applications. In addition, you can use the Java GUI visual debugger with JavaFX applications. For more information on JavaFX, see the JavaFX 2 documentation at <http://docs.oracle.com/javafx/>.

### 6.2.4 Creating Free-Form Projects

Free-form projects do not produce an Ant script or hold project metadata in another form. Instead, these projects rely on your existing Ant script to provide instructions for handling your project source.

#### To create a free-form project:

1. Choose **File > New Project** (Ctrl+Shift+N).
2. Select the appropriate template for your project.
3. Follow the steps in the remainder of the application wizard.

#### 6.2.4.1 Source Folders

Free-form Java projects can contain multiple source folders. The classpath relationships and handling instructions for your source directories must be handled in your Ant script.

#### 6.2.4.2 Project Settings

In the project's properties, you declare the source folders, classpath, and output files for your project. Each source root can have a unique classpath and output file. These settings do not affect the actual contents or classpath of your application. The settings only tell the IDE how to handle the code. For example, the classpath settings tell the IDE which classes to make available for code completion.

For information on how to declare a classpath, see [Section 6.2.4.5, "Declaring the Classpath in a Free-Form Project."](#)

#### 6.2.4.3 IDE Commands and Ant Targets

In free-form projects, the IDE relies on your Ant script to provide targets for all IDE actions, such as running, debugging, and generating Javadoc. If your Ant script does not contain targets for these commands, the commands are disabled.

You can write debug targets in your Ant script or in a supplementary Ant script. You can also add a shortcut to any of your Ant script's targets to the contextual menu of your project's node in the Projects window.

For information on mapping an Ant target to a command in the IDE, see [Section 6.2.4.6, "Mapping an Ant Target to an IDE Command."](#)

For information on setting the IDE to read targets from a separate Ant script, see [Section 6.2.4.8, "Storing IDE Targets in a Separate Ant Script."](#)

#### **6.2.4.4 Adding a Source Directory to a Free-Form Project**

Free-form projects can contain multiple source directories, as long as the project's Ant script contains instructions on handling those source directories. To make project sources available to Ant, you need to specify the classpath for the project sources. If you have any custom tasks, you also need to add these tasks to Ant's classpath.

##### **To add a source directory to a free-form project:**

1. Build the project so that all of the project's build outputs (JAR files and WAR files) exist on your computer.
2. In the **Projects** window, right-click the project node and choose **Properties**.
3. Click **Sources** in the right panel of the **Project Properties** dialog box and add the source folder.
4. Click **Classpath** in the right panel of the **Project Properties** dialog box and set the classpath for the source directory.

You do this because by default the IDE ignores your environment's CLASSPATH variable whenever it runs Ant.

---

**Note:** The classpath variable you set in the **Project Properties** dialog box does not affect the actual classpath of the project, which is specified in the Ant script. Declaring the classpath in the **Project Properties** dialog box does not change the actual compilation or runtime classpath of the source folders. However, the project classpath variable must match the classpath used by your Ant script in order to provide the correct information for code completion, error highlighting, and refactoring commands. You have to set an explicit classpath in your build scripts because the IDE ignores your environment's CLASSPATH variable whenever it runs Ant. If you change the classpath of one, you must change the class path of the other.

---

5. Click **Output** in the right panel of the **Project Properties** dialog box and specify the source folder's build output.
6. Click **OK**.

#### **Specifying the Classpath for Custom Tasks**

In free-form projects, you can call up and run custom Ant tasks in your build script. For your Ant script to use customs tasks, you must include the tasks in the Ant script's classpath. For example, you may add a task to your build script to format your code with Jalopy. In order to do this, however, you have to add the Jalopy JAR file to Ant's classpath.

You can add custom tasks to Ant's classpath within the IDE by doing either of the following:

- Providing an explicit classpath to the tasks in your build script. This is the recommended method for specifying the location of JAR files that contain custom tasks used by your Ant script, as it ensures that your build scripts will be fully portable. You can write your tasks and include instructions to compile them and produce a JAR file in the build file. To use these tasks, include the long form of taskdef, which includes a classpath. Here is a simple example of such a task:

```
<project name="test" default="all" basedir=".">
    <target name="init">
        <javac srcdir="tasksource" destdir="build/taskclasses"/>
        <jar jarfile="mytasks.jar">
            <fileset dir="build/taskclasses"/>
        </jar>
        <taskdef name="customtask" classname="com.mycom.MyCustomTask">
            <classpath>
                <pathelement location="mytasks.jar"/>
            </classpath>
        </taskdef>
    </target>
</project>
```

The advantage of this method is that no special preparation is needed to begin using the script. The script is entirely self-contained and portable. This method also makes it easier to develop your tasks within the IDE, as the script compiles them for you automatically.

To make your build scripts even more robust, use a property instead of a hard-coded location to specify the classpath to your tasks. You can store the property in the build script itself or in a separate ant.properties file. You can then change the classpath setting throughout your script by simply changing the value of the specified property.

- Configuring the Ant Classpath property in the Options window. If you cannot declare a classpath in your build script, or you are using third-party build scripts which you cannot alter, you can add tasks to Ant's classpath in the IDE in the Options window.

---

**Note:** If you modify the Ant classpath in the **Options** window, the task is on Ant's classpath for all projects when you run Ant in the IDE.

---

#### 6.2.4.5 Declaring the Classpath in a Free-Form Project

In a free-form project, all classpath relationships between your source folders are handled by your Ant script. In order for the IDE to know which classes to include in code completion and refactoring, you have to declare the classpath in the project settings.

You first declare the classpath in the New Project wizard when creating the project. You can edit all classpath declarations for an existing project in the Project Properties dialog box.

Declaring the classpath in the Project Properties dialog box does not change the actual compilation or runtime classpath of the source folders. The project's classpath declaration must exactly match the classpath used by your Ant script.

### To declare the classpath for a project:

1. Build the project so that all of the project's build outputs (JAR files and WAR files) exist on your computer.
2. In the Projects window, right-click the project node and choose **Properties**.
3. Click Java Sources Classpath in the Categories panel of the Project Properties dialog box.
4. Declare the classpath for the project.
  - To set a common classpath for all source folders, unselect the **Separate Classpath for Each Source Package Folder** checkbox. Then add or remove classpath elements with the buttons on the right of the dialog box.
  - To set a separate classpath for each source folder, leave the **Separate Classpath for Each Source Package Folder** checkbox selected. Select a source folder in the drop-down list and add or remove classpath elements for the source folder.
  - To add the build output of source folder A to source folder B, select the source folder B in the drop-down list and click **Add JAR/Folder**. Then navigate to the output folder of source folder A and select its build output. You must have a separate classpath for each source package folder to set up relational classpath dependencies between source package folders.

If you have attached Javadoc and source files to a JAR file in the Ant Library Manager, the IDE automatically adds the Javadoc and source files to the project when you register the JAR file on a project's classpath. You can step into classes and look up Javadoc pages for the classes without configuring anything else.

For information on how to add multiple source directories to a free-form project, see [Section 6.2.4.4, "Adding a Source Directory to a Free-Form Project."](#)

For information on adding Javadoc documentation associated with a JAR file, see [Section 6.7.1, "How to Add a Javadoc to a Project."](#)

For information on adding source code to a JAR file or a compiled classes folder, see [Section 10.9.3, "Attaching Source Code to a JAR File."](#)

#### 6.2.4.6 Mapping an Ant Target to an IDE Command

In a free-form project, IDE commands must be mapped to targets in your Ant script. These mappings are recorded in the `project.xml` file in your project folder.

There are three ways to map an IDE command to a target in an Ant script:

- By adjusting the settings in the **Build and Run** page of a project's **Project Properties** dialog box
 

You can use the **Project Properties** dialog box to add a shortcut to any Ant target to the project node's context menu. For example, if you have several runnable classes that you have to run often, you can write targets to run these classes and then run them using the links. Right-click the project node, choose **Properties**, and register the shortcuts in the Build and Run page of the dialog box.
- By having the IDE generate a target for you and then customizing this target to your needs. This works for the Debug Project and Compile File commands. The IDE offers to generate these targets the first time you run those commands in the project.
- By manually editing the project's `project.xml` file.

Mappings are recorded in the `project.xml` file automatically in the following cases:

- When you specify a target for a command in the Build and Run Actions page of the New Project wizard.
- When you specify a target for a command in the Build and Run panel of the Project Properties dialog box for a project.
- When you create a mapping for a command after having chosen the command from the **Build** menu or **Run** menu and having been prompted to create the mapping.
- When you have had the IDE generate a target. (The IDE offers to generate a target for the **Debug Project** and **Compile File** commands the first time you choose these commands in a free-form project.)

#### To map commands for a project:

1. In the **Projects** window, right-click the project node and choose **Properties**.
2. Click **Build and Run** in the right panel of the Project Properties dialog box.

---

**Note:** The **Project Properties** dialog box is the main tool for configuring free-form projects in the IDE.

---

3. For each command (Build Project, Clean Project, Generate Javadoc, Run Project (free-form Java projects), Deploy Project (free-form Web projects), and Test Project), choose an Ant target from the combo box. The combo box contains each of the targets in your Ant script.

---

**Note:** If your Ant script uses an `import` statement to import targets from another Ant script, the targets only appear if the `<import>` target specifies the full path to the secondary Ant script. If the `<import>` target uses a property to reference the secondary Ant script, the targets do not show up in the drop-down lists. In this case, you have to type the name of the projects in the drop-down list for the command.

---

4. Use the **Custom Menu Items** list to add shortcuts to Ant targets to the project's contextual menu.

Each IDE project has a `project.xml` file that contains important metadata about your project's contents, the location of the project's Ant script, which targets to run for IDE commands, and other information. If you want to map commands that work on the presently selected files in the IDE, or if you want to map a command to a target in a separate Ant script, you have to edit the `project.xml` file by hand.

#### To map IDE commands in `project.xml`:

1. In the **Files** window, expand the node for your project folder and expand the `nbproject` folder.
2. Double-click `project.xml` to open it in the Source Editor.
3. Enter the following in `<ide-actions>`:

```
<action name="action_name">
  <target>target_name</target>
</action>
```

---

**Note:** The `<ide-actions>` element holds the mappings for IDE commands. You enter an action element with the name for any of the standard IDE actions and define the script and target to which you want to map the command.

---

If you want to map an action to a target in a separate Ant script, add the following before the `<target>` declaration:

```
<script>path_to_Ant_script</script>
```

You can map any of the following IDE actions:

- build. Build project.
- rebuild. Clean and build project.
- compile.single. Compile selected file.
- clean. Clean project
- run. Run project.
- run.single. Run the currently selected file.
- redeploy. For Web application projects, build project, undeploy project from server, and deploy project to server.
- test. Run tests for project.
- test.single. Run the test file for the currently selected file.
- debug.test.single. Run the test file for the currently selected file in the debugger.
- debug. Debug project.
- debug.single. Compile the currently selected file.
- debug.fix. Run the **Apply Code Changes** command on the currently selected file.
- debug.stepinto. Execute one line of the project main class in the debugger and pause.
- profile.test.single. Profile the JUnit test for the selected file.
- profile. Run project in the profiler.
- profile.single. Profile the selected file.
- javadoc. Generate Javadoc for project.

Note that actions that run on the currently selected files in the IDE require additional configuration in `project.xml`.

4. If you also want the command to appear in the project's contextual menu, enter the following in `<context-menu>`:

```
<ide-action name="action_name" />
```

For example, the following maps the Debug Project to the `debug-nb` target of the project's Ant script:

```
<action name="debug">
```

```
<target>debug-nb</target>
</action>
```

The Ant targets for NetBeans IDE commands do not have to live in the same Ant script that you use to build and run the project. This is useful for users who cannot alter their Ant script. The following maps the Debug Project to the debug-nb target in a separate Ant script:

```
<action name="debug">
  <script>path/to/my/nbtargets.xml</script>
  <target>debug-nb</target>
</action>
```

---

**Note:** `<script>` must precede `<target>`.

---

You can also configure a command to run multiple targets. The targets are run in the order they appear in the action. For example, the mapping for the Clean and Build Project command looks like this:

```
<action name="rebuild">
  <target>clean</target>
  <target>compile</target>
</action>
```

#### Adding Shortcuts to Project Node Contextual Menu

`project.xml` also has a `<context-menu>` element that controls the contents of a project node's contextual menu. If you manually add an action that is run on the project, make sure you register the action name in `<context-menu>` as well. If you use the Project Properties dialog box to configure a standard project command, the IDE automatically adds the command to the project's contextual menu.

For information on creating a debug target for a free-form project's build script, see [Section 10.8.1.1, "How to Create a Debug Target for a Free-form Java Project."](#)

#### 6.2.4.7 Debugging Free-Form Projects

Similar to commands for compiling and running, debugging commands rely on various information, such as the location of your sources, the location of the compiled classes and other items on the classpath, and name of the project's main class.

In free-form projects, the IDE is not aware of any of these things. When you run a command in the IDE (such as **Build**), the IDE simply calls a target in your build script and lets the script handle the command. Therefore, for debugging to work, you also have to have a build script target for debugging. The IDE provides some custom Ant tasks to work with the debugger and also can generate a basic debug target, which attempts to fill in important details based on other targets in your script.

For more information on debugging free-form projects, see [Section 10.8.1, "Debugging Free-form Projects."](#)

#### 6.2.4.8 Storing IDE Targets in a Separate Ant Script

When using a free-form project, you sometimes have to write Ant targets for IDE actions. For example, you have to write an Ant target to run a program in the debugger or to compile the currently selected file in the IDE.

If you are not able to write these IDE targets in your project's Ant script, you can set the IDE to read these targets from a separate Ant script.

### To map an IDE command to an Ant target in a separate Ant script:

1. Write the target in a separate Ant script.
2. In the Files window, expand your project folder node and the nbproject node.
3. Double-click project.xml to open it in the Source Editor.
4. Enter the following in <ide-actions>:

```
<action name="action_name">
    <script>path_to_Ant_script</script>
    <target>target_name</target>
</action>
```

Note that <script> must precede <target>.

5. If the command also appears in the project's contextual menu, enter the following in <context-menu>:

```
<ide-action name="action_name" />
```

For information on mapping Ant targets to IDE commands, see [Section 6.2.4.6, "Mapping an Ant Target to an IDE Command."](#)

#### 6.2.4.9 Editing the project.xml File

Each IDE project has a project.xml file that includes important information about the project, such as:

- Information about what the project's type (free-form or standard, Web application or standard Java SE application)
- Mappings between project commands and targets in an Ant script
- Information about the project's contents, classpath, and target Java platform. This information is used to visualize the project and enable code completion and refactoring.

For standard projects, there is usually no need to edit the project.xml file. In free-form projects, you often have to edit the project.xml file to hook up Ant targets to IDE commands and make other customizations.

##### 6.2.4.9.1 Using Properties in the project.xml File

Like all XML files, you can define properties inside the XML file itself or store them in a separate .properties file. One way of keeping your project.xml page synchronized with the information in your Ant script is to import properties into project.xml from the same .properties file that is used by your Ant script.

---

**Note:** All file paths in project.xml are by default relative to the project folder. If your Ant script is not located in the project folder, a classdir property that points to build/classes/ does not point to the same directory for the Ant script and for the project.xml file. (The project folder is the folder that contains your nbproject folder, not the nbproject folder itself. By default, the new free-form project wizard makes your Ant script's parent folder the project folder.)

You can solve this problem by defining properties for important paths (like project.dir) and using these properties to be more exact (for example, classdir=\${project.dir}/build/classes).

---

**To create and import properties in project.xml**

1. In the Files window, double-click project.xml.
2. Enter the following between the <name> element and the <folders> element:

```
<properties>
    <property name="name">value</property>
    <property-file>my-properties-file.properties</property-file>
    <property-file>another-properties-file.properties</property-file>
</properties>
```

The syntax is different than the syntax used in Ant scripts.

---

**Note:** While you can add properties in any order, properties can only refer to other properties that have been defined previously in the file. The properties file path itself can also use property substitutions.

---

**6.2.4.9.2 Validating the project.xml File**

The IDE comes bundled with the XML schemas for free-form project.xml files and automatically validates a free-form project.xml file every time you edit and save it. You can view the XML schemas for the free-form project.xml file at the following locations:

- <http://netbeans.org/ns/freeform-project/1.xsd>. Controls the main section of the free-form project.xml (<general-data>).
- <http://netbeans.org/ns/freeform-project-java/1.xsd>. Controls the <java-data> section.
- <http://netbeans.org/ns/freeform-project-web/1.xsd>. Controls the <web-data> section (if you have one).

**6.2.5 Setting Up a Java Project Based on Existing Sources**

For Java projects developed outside of NetBeans, you use an Existing Sources template in the **New Project** wizard to make a NetBeans project. In the wizard, you identify the location of the sources and specify a location for the NetBeans project metadata. You then use the **Project Properties** dialog box to configure the project.

**To set up a NetBeans project for an existing Java application:**

1. Choose **File > New Project**.
2. Choose **Java > Java Project with Existing Sources**. Click **Next**.
3. In the **Name and Location** page of the wizard, follow these steps:
  - Type a project name.
  - (Optional) Change the location of the project folder.
  - (Optional) Change the name of the build script used by the IDE. This might be desirable if there is already a build script called build.xml that is used to build the sources.
  - (Optional) Select the **Use Dedicated Folder for Storing Libraries** checkbox and specify the location for the libraries folder.

- (Optional) Select the **Set as Main Project** checkbox. When you select this option, keyboard shortcuts for commands such as **Clean and Build Main Project** apply to this project.
4. Click **Next** to advance to the **Existing Sources** page of the wizard.
  5. In the **Source Packages Folder** pane, click **Add Folder**. Then navigate to your sources and select the source roots, click **Open**.

---

**Note:** When you add a folder containing source code, you must add the folder that contains the highest folder in your package tree. For example, for the `com.mycompany.myapp.ui` package, you add the folder that contains the `com` folder.

---

6. (Optional) In the **Test Package Folders** pane, click **Add Folder** to select the folder containing the JUnit package folders.
7. Click **Next** to advance to the **Includes & Excludes** page of the wizard.
8. (Optional) In the **Includes & Excludes** page of the wizard, enter file name patterns for any files that should be included or excluded from the project. By default, all files in your source roots are included.
9. Click **Finish**.

## 6.3 Importing an Eclipse or JBuilder Project

You can import projects created in other IDEs, such as Eclipse or JBuilder, into the NetBeans IDE.

The functionality to import Eclipse projects is included into the standard NetBeans IDE distribution. For Eclipse projects, you can import a set of dependent projects from an Eclipse Workspace or import a single project ignoring dependencies.

For JBuilder projects, you must install the JBuilder Project Importer plugin from the NetBeans Update Center. For JBuilder projects, you can import a standard Java SE project with dependencies.

### To import an Eclipse project into the NetBeans IDE:

1. Make sure that you have copy of the project that you want to import on your system.  
Typically, this project would already be in an Eclipse workspace on your system.
2. Choose **File > Import Project > Eclipse Project**.
3. In the **Workspace Location** page of the wizard, select the **Import Projects from Workspace** radio button, and specify the workspace location. Click **Next**.
4. In the **Projects to Import** page, select the projects that you want to import.
5. Choose one of the following two options for storing the NetBeans project files:
  - **Store NetBeans project data inside Eclipse project folders.** NetBeans adds folders and files within the top-level folder of the original project.
  - **Create imported NetBeans projects in a separate location.** NetBeans uses the sources and libraries in the original Eclipse project folder but creates a separate folder to hold NetBeans project metadata and build outputs.

---

**Note:** Typically, it is better to store NetBeans project data inside Eclipse project folders. In most cases, this means that the NetBeans project metadata will refer to sources and libraries with the same paths that are used by the Eclipse metadata. Therefore, checking out the project from a version control system on different machines should result in similar behavior both in NetBeans and Eclipse.

---

6. (Applicable when web applications are being imported only.) Click **Next**. In the **Servers** page, register any servers that your projects need with NetBeans IDE.
7. Click **Finish**.

After you have completed the wizard, the following dialog boxes might appear:

- **Import Issues.** The **Import Issues** dialog box provides information about discrepancies between the project structure in Eclipse and in NetBeans and points out actions that you might need to take to correct the discrepancies.

You can copy the information from this dialog and paste it elsewhere for future reference.

In most cases, you use the project's **Project Properties** dialog box to resolve those issues. See Resolving Import Problems for a guide to resolving the most common problems.

- **Resolve Reference Problems.** The **Resolve Reference Problems** dialog box alerts you to a specific reference problem with one of your project libraries. You can solve this problem after dismissing this dialog box by right-clicking the project's node and choosing **Resolve Reference Problems**.
- **Resolve Missing Server.** The **Resolve Missing Server** dialog box alerts you that the project cannot find a necessary server. You can solve this problem after dismissing this dialog box by right-clicking the project's node and choosing **Resolve Missing Server**.

After you have completed the wizard and have closed any of the above informational dialog boxes, nodes for the projects will appear in the **Projects** window.

If there are references in your project metadata to servers or other resources that NetBeans cannot resolve, the node for the project will appear in red. You can resolve these references immediately by right-clicking the project node and choosing **Resolve Reference Problems** or **Resolve Missing Server**.

For other types of project configuration adjustments, you use the **Project Properties** dialog box. You open the **Project Properties** dialog box by right-clicking the project's node and choosing **Properties**.

After you have imported the project, you will find the following folder and files on your system:

- build.xml file or nb-build.xml file. The main NetBeans build script for the project. You can customize this script according to the needs of your project. By default, this file is called build.xml. If such a file already exists in the project folder, the script is called nb-build.xml.
- nbproject folder. Contains most of the NetBeans project metadata, including resources that are called by the main NetBeans build script. If you check this folder and the build.xml file or nb-build.xml into your version control system, other users will be able to open the project in NetBeans. This folder also contains the private folder, which contains data specific to your system. This folder should not

be checked in to the version control system since its contents will vary between users. See Version Control Considerations below.

- `nbbuild` folder. When you build or run your project in NetBeans, the project's sources are compiled into this folder.
- `nbdist` folder. When you build your project in NetBeans, the project's distributable outputs are created and placed in this folder. Such outputs might be JAR files and WAR files.

### **Version Control Considerations**

If the project is checked out of a version control system, the `build` (or `nbbuild`), `dist` (or `nbdist`), and the `nbproject/private` folders should not be checked into that version control system.

If the project is under the CVS, Subversion, or Mercurial version control systems, the appropriate "ignore" files are created or updated for these directories when the project is imported.

---

**Note:** Though `nbproject/private` should be ignored, `nbproject` should be checked into the version control system. `nbproject` contains project metadata that enables other users to open the project in NetBeans without having to import the project first.

---

### **Building and Running an imported Project**

Once you have the project imported into NetBeans, you can build and run the project. All artifacts created from NetBeans build and run commands are created in the `build` and `dist` folders. NetBeans does not over-write output created from Eclipse build actions. If the Eclipse project already has `build` and `dist` folders, the NetBeans project creates folders called `nbbuild` and `nbdist` and uses those for the build outputs.

The following are some of the build and run commands available from the Run menu:

- **Run Project.** Test runs the application in the IDE.
- **Clean and Build Project.** Deletes the contents of the `build` (or `nbbuild`) and `dist` (or `nbdist`) folders and rebuilds all of the project's outputs. Uses the NetBeans build script. Similar to the Clean command in Eclipse.
- **Clean.** Deletes the contents of the `nbbuild` and `nbdist` folders.
- **Build.** Rebuilds the project's outputs. If the Compile on Save feature is enabled, the Build command is disabled.

### **Resynchronizing a Project**

The project importer features synchronization capabilities. If the classpath in the Eclipse has changed since you initially imported it, you can use the Resynchronize Eclipse Projects feature to update the classpath in the corresponding NetBeans project.

Project resynchronization is one-way from Eclipse projects to NetBeans projects. If you make changes to the project structure in NetBeans, those changes are not propagated to the Eclipse project with the resynchronization feature. If you intend to keep both Eclipse and NetBeans projects, use the Eclipse project as the "master" project.

The IDE also resynchronizes the projects automatically if the changes to the Eclipse configuration are unambiguous and do not require your input. This automatic

resynchronization occurs shortly after you open the project. If the resynchronization requires your input, you need to manually resynchronize the project.

#### To manually resynchronize NetBeans projects with Eclipse projects:

- Choose **File > Import Project > Resynchronize Eclipse Projects**.

---

**Note:** When you resynchronize a project, the resynchronization is performed on all projects that you have imported from the workspace.

---

#### Resolving Import Problems

When you import a project into NetBeans, there might be some things that can not be automatically resolved in NetBeans IDE. For some of these problems, a menu item, such as **Resolve Missing Server Problem**, appears in the contextual menu for the project. Other problems can be resolved in the Project Properties dialog box for the imported project in NetBeans IDE.

Here is a list of common import problems and their solutions.

Problem Message	Solution
Resolve Missing Server Problem	Right-click the project node and choose <b>Resolve Missing Server Problem</b> . Then navigate to the file or folder that contains the server.
Resolve Reference Problem	Right-click the project's node and choose <b>Resolve Reference Problem</b> . Then navigate to the file or folder that contains the resource that is referred to from the project.
Eclipse platform for project <i>ProjectName</i> cannot be used. It is a JRE and the NetBeans project requires a JDK. NetBeans will use the default platform.	If you would like to change the platform that NetBeans uses for the project, choose <b>Tools &gt; Platforms</b> and specify a different platform.
Eclipse project <i>ProjectName</i> claims to use JDK from the "{1}" directory. But this directory does not exist. NetBeans will use the default platform.	If you would like to change the platform that NetBeans uses for the project, choose <b>Tools &gt; Platforms</b> and specify a different platform.
NetBeans does not support source includes/excludes per source root as Eclipse does. They were merged and it is recommended that you double check them in project's properties in Source panel.	In NetBeans, includes and excludes are declared in one place for the whole project. To check the includes and excludes in the NetBeans project, right-click the project's node and the Projects window and choose Properties. In the Project Properties dialog box, select the Sources tab and then click the Includes/Excludes button.
Import failed due to .... More details can be found in IDE's log file.	You can open the IDE's log file by choosing <b>View &gt; IDE Log</b> .
Unknown project type - it cannot be imported.	You can only import the following Eclipse project types: Java Project, Java Project from Existing Ant File, Static Web, Dynamic Web, and JPA Project.

#### To install the JBuilder Project Importer plugin:

1. Choose **Tools > Plugins** from the main menu.
2. Click the Available Plugins tab and select **JBuilder Project Importer**.

3. Click **Install**.

**To import a JBuilder project:**

- Choose the JBuilder project type from the **File > Import Project** menu.

For information on how to create a Java project, see [Section 6.1, "About Creating Java Projects."](#)

For information on adding dependent projects, see [Section 6.2.3.2, "Creating Dependencies Between Projects."](#)

## 6.4 Setting the Main Project

When you develop a large application consisting of numerous source directories, it is common to split up your code into separate projects. Of these projects, one is typically the entry point for your application and contains the application's main class.

To tell the IDE which of your projects is the main entry point for your application, you set one project to be the main project. The IDE provides commands that act on the main project. For example, running the **Build Main Project** command builds both the main project and all of its required projects, thereby ensuring that you all of your compiled classes are up-to-date. Only one project can be the main project at any time.

After you set a project as the main project, the keyboard shortcuts for **Run** (F6), **Build** (F11) and **Clean and Build** (Shift+F11) apply to the main project regardless of which project is selected in the Projects window. Keyboard shortcuts for **Debug** and **Profile** also apply to the main project.

**To make a project the main project:**

- Choose **Run > Set Main Project** from the main menu and select a project from the list of open projects.

Alternatively, you can right-click the project node in the Projects window and choose **Set as Main Project**.

---

**Note:** If you create separate projects for each source root, you must set the classpath dependencies between the main project and the required projects. For more information, see [Section 6.2.3.2, "Creating Dependencies Between Projects."](#) For information on setting and modifying a project's classpath, see [Section 6.2.3.1, "Managing the Classpath."](#)

---

## 6.5 Adding Multiple Sources Roots to a Project

You can add multiple sources roots to either standard or free-form projects that require them. However, never add the same source root to more than one project because this is not supported in the IDE.

In Standard projects, it is sometimes advantageous to add additional source roots to your project's classpath such as when the project uses sources stored in a separate resources folder. Because all source directories on a standard project's classpath form a single compilation unit, they are also packaged into the application's JAR when the project is built. In Standard projects, you add source roots to the classpath using the project wizard and properties dialog.

In free-form projects, each source root can be assigned to independent compilation units providing you maximum flexibility when structuring your applications.

Free-form projects, however, require that you manage the classpath by editing your build.xml file directly. For more information, see [Section 6.2.3.1, "Managing the Classpath."](#)

---

**Note:** For both standard and free-form projects, source roots must only exist in a single project and cannot be shared with other projects, regardless of whether they are opened or not. If you have to use a library in several projects, create a special project within which to store it.

---

If you want to add a library to multiple projects, you must create a special project within which to store the library's compiled source that you want to use.

For more information on using project templates, see [Section 6.2, "Using Java Project Templates."](#)

## 6.6 Sharing a Library with Other Users

As the NetBeans IDE project system is Ant-based, NetBeans projects are generally portable between different users, whether or not they use the IDE. However, by default, the project build script's way of referring to libraries is dependent on factors specific to each user, particularly in the case of libraries defined in the Library Manager dialog box.

A potential inconvenience is that libraries are stored in a variety of locations, based on where they originate. Libraries that come with the IDE are stored in various different folders within the IDE's installation. Examples of these include the libraries for the Swing Layout Extensions, beans binding, and database drivers.

You can configure most standard Java SE, Web, and Enterprise projects in a way that makes it easy to share libraries with other users. You can specify a location for libraries on which the project relies. You can also specify how the libraries are referenced from your project.

These options make it easy for you to handle the following situations:

- You create a project and need to make it available and buildable to other users, whether they use the IDE or not. They must be able to access the project through a version control checkout or by unpacking a ZIP file that you have provided them. Then they should be able to build the application without extra configuration.
- You must start working on an existing project and adhere to a strict set of conventions concerning where project libraries are stored (and whether your build script accesses them with a relative reference or absolute reference). Other users on the team are not using NetBeans and have no plans to switch.

### 6.6.1 How to Share a Library

You can make a project's libraries sharable when you create the project in the New Project wizard. You can convert an existing project to be sharable in the Libraries tab of the Project Properties dialog box.

**To make a general Java project's libraries sharable upon project creation:**

1. Choose **File > New Project**.
2. In the Java category of the wizard, select one of the standard templates and click **Next**.

3. In the Name and Location page of the wizard, select the **Use Dedicated Folder for Sharing Libraries** checkbox.
4. In the Libraries field, select the location for the libraries to be stored.  
If the libraries are already included in the IDE, those libraries are copied to the folder that you have designated.

**To make a web or enterprise project's libraries sharable upon project creation:**

1. Choose **File > New Project**.
2. Select one of the standard templates in the Web or Enterprise category and click **Next**.
3. In the Name and Location page of the wizard, select the **Use Dedicated Folder for Sharing Libraries** checkbox.
4. In the Libraries field, select the location for the libraries to be stored.  
If the libraries are already included in the IDE, those libraries are copied to the folder that you have designated.
5. (Optional) On the Server and Settings page, select the **Copy Server JAR Files to Libraries Folder** radio button.

**To make an existing project's libraries sharable:**

1. Right-click the project's node and choose **Properties**.
2. In the Project Properties dialog box, select the Libraries node.
3. In the Libraries panel, click **Browse** to open the New Libraries Folder wizard.
4. In the Library Folder page of the wizard, enter a location for the libraries and click **Next**.

You can enter the location as a relative reference or an absolute reference.

5. In the Actions panel of the wizard, verify the selected action for each listed library. In most cases, the IDE detects the most appropriate action for that library.

The following actions are available:

- **Copy Library JAR Files to New Libraries Folder.** Use this option if the library is not in the folder that you have selected and you want to have the library JAR files placed there.
- **Use Relative Path to Library JAR Files.** Use this option if the library is not in the libraries folder and you have to access the library in its existing location using a relative path. An entry with a relative reference is added for the library in the libraries folder's nblibraries.properties file.
- **Use Absolute Path to Library JAR Files.** Use this option if the library is not in the libraries folder and you need to access the library in its existing location using an absolute path. An entry with an absolute reference is added for the library in the libraries folder's nblibraries.properties file.
- **Use Existing Library in Libraries Folder.** Use this option if there is already a copy of the library in the libraries folder and you want to use that copy of the library.

6. Click **Finish** to exit the Make Project Sharable wizard.
7. Click **OK** to exit the Project Properties dialog box.

You can also use the Libraries node of the Project Properties dialog box to change the location of the libraries folder. If you have already specified a libraries folder, click **Browse** to open a file chooser instead of the New Libraries Folder wizard.

---

**Note:** Free-form project libraries cannot be sharable in the ways described in this topic.

---

For information on creating standard Java projects, see [Section 6.2.3, "Creating Standard Projects."](#)

For information on creating free-form projects, see [Section 6.2.4, "Creating Free-Form Projects."](#)

## 6.7 Adding a Javadoc to a Project

You can make Javadoc documentation for a JAR file's class available in the IDE by associating that documentation with the JAR file.

If you want to have access to compiled JDK documentation through the IDE, see [Section 2.15.2, "How to Add the JDK Javadoc to the IDE."](#)

### 6.7.1 How to Add a Javadoc to a Project

When you add a required project to a project's classpath, the required project's Javadoc and sources are automatically added to the project as well.

#### To add Javadoc for a JAR file:

1. Choose **Tools > Libraries** from the main menu.
2. In the left pane of the Ant Library Manager, select the project library within which the JAR file you want to add Javadoc documentation to is located.  
Only libraries already registered with the IDE are listed in the Ant Library Manager's Class Libraries list.
3. If the JAR file for which you want to add Javadoc documentation has not already been added to a registered library, create a new empty library using the **New Library** button. Next, in the Classpath tab click **Add JAR/Folder** and specify the location of the JAR file containing the compiled class files.

---

**Note:** You can also associate the Javadoc with a JAR file using the project's Project Properties window. However, doing so creates the association only for that project. Open the Project Properties dialog box by right-clicking the project node and choosing **Properties**. Select the Libraries node in the Categories pane. Then select the JAR with which you want to associate the Javadoc and click **Edit**. You can then specify the sources to be associated.

---

A class library can contain multiple JAR files as well as their Javadoc documentation and source code.

4. In the Javadoc tab, click **Add ZIP/Folder** and specify the location of the Javadoc files.
5. Click **OK** to exit the Ant Library Manager.

The IDE adds the selected JAR files and Javadoc documentation ZIP files to the specified library and automatically registers the documentation in every project that has that JAR file on its classpath.

When you create a Java class library for a single JAR file, you can simply add the JAR file to the project's classpath to make the associated Javadoc and source code available. If your Java library contains multiple JAR files, however, you must add the library itself to the classpath. Adding the library to the classpath also makes it easier to share the project with other developers.

For information on setting the classpath for standard projects, see [Section 6.2.3.1, "Managing the Classpath."](#)

For information on adding source code to a JAR file or a compiled classes folder, see [Section 10.9.3, "Attaching Source Code to a JAR File."](#)

For information on how to view Javadoc in the IDE, see [Section 2.15.3, "How to View Javadoc Documentation."](#)

## 6.8 Setting the Target JDK

By default, the IDE uses the version of the Java SE platform (JDK) with which the IDE runs as the default Java platform for compilation, execution, and debugging. You can view your IDE's JDK version by choosing **Help > About** and clicking the Detail tab. The JDK version is listed in the Java field.

You can run the IDE with a different JDK version by starting the IDE with the `--jdkhome jdk-home-dir` switch on the command line or in your `IDE-HOME/etc/netbeans.conf` file. For more information, see [Section 2.7, "Setting Startup Parameters."](#)

You cannot enable JavaFX for the default registered Java platform. However, you can use the same JDK sources as the default platform in a new registered Java platform and enable JavaFX in the new platform.

### 6.8.1 How to Register a New Java Platform

In the IDE, you can register multiple Java platforms and attach Javadoc and source code to each platform. For example, if you want to work with the new features introduced in JDK 7, you would either run the IDE on JDK 7 or register JDK 7 as a platform and attach the source code and Javadoc to the platform. You can also enable JavaFX for multiple platforms.

#### To register a new Java platform:

1. Choose **Tools > Java Platforms** from the main window.
2. Click **Add Platform** and select the directory that contains the Java platform. Java platform directories are marked with a  icon in the file chooser. Click **Next**.

If support for Java ME is enabled in the IDE you must select to add either a Java Standard Edition platform or one of the Java ME platforms. For information on adding a new Java ME platform, see [Section 22.9, "Using Java ME Emulator Platforms."](#)

3. Specify the display name for the platform and the location of the sources for the Java platform and the Javadoc. Click **Finish**.

#### To register a new Java SE Embedded platform:

1. Choose **Tools > Java Platforms** from the main window.

2. Click **Add Platform** and select the **Remote Java Standard Edition** option. Click **Next**.
3. Specify the display name for the platform, the details of the remote device where an application is executed, your authentication details, and the path to the JRE on the remote device. Click **Finish**.

## 6.8.2 How to Set the Target JDK

In standard projects, you can switch the target JDK in the Project Properties dialog box. In free-form projects, you have to set the target JDK in the Ant script itself, then specify the source/binary format in the Project Properties dialog box.

### To set the default Java platform for a standard project:

1. Right-click the project's root node in the Projects window and choose **Properties**.
2. In the Project Properties dialog box, select the **Libraries** node in the left pane.
3. Choose the desired Java platform in the Java Platform combo box.

Switching the target JDK for a standard project does the following:

- Offers the new target JDK's classes for code completion.
- If available, displays the target JDK's source code and Javadoc documentation.
- Uses the target JDK's executables (`javac` and `java`) to compile and execute your application.
- Compiles your source code against the target JDK's libraries.

4. Click **Manage Platforms** to register additional Java platforms with the IDE.
5. Click **Add Platform** and navigate to the desired platform

### To switch the target JDK of a standard project:

1. Right-click the project's node and choose **Properties**.
2. Select the **Libraries** panel.
3. Change the **Java Platform** property.

### To set the target Java platform for a free-form project:

1. In your Ant script, set the target JDK as desired in the `javac`, `java`, and `javadoc` tasks.
2. Right-click the project's root node in the Projects window and choose **Properties**.
3. In the Sources panel, set the level of JDK you want your application to be run on in the Source/Binary Format combo box.

When you access Javadoc or source code for JDK classes, the IDE searches the Java platforms registered in the Java Platform Manager for a platform with a matching version number. If no matching platform is found, the IDE's default platform is used instead.

For information on setting the classpath for standard projects, see [Section 6.2.3.1, "Managing the Classpath."](#)

For information on how to declare a classpath, see [Section 6.2.4.5, "Declaring the Classpath in a Free-Form Project."](#)

### 6.8.3 How to Upload Java SE Embedded JRE to a Remote Device

When no JRE is installed on a remote device, new embedded JRE must be uploaded and deployed.

**To upload and deploy an embedded JRE on a remote device:**

1. Choose **Tools > Java Platforms** from the main window.
2. Click **Add Platform** and select the **Remote Java Standard Edition** option. Click **Next**.
3. In the **Add Java Platform** dialog, specify the display name for the platform, the details of the remote device where an application is executed and your authentication details. Click **Create**.
4. In the **Create Java SE Embedded JRE** dialog, click **Browse** and browse for the directory with JRE create on your hard drive.
5. Specify the path to the directory on a remote device where JRE must be installed and the required options. Click **Create**.

## 6.9 Moving, Copying, and Renaming a Project

When you create a project, the IDE bases much of its project metadata on the name and location of the project. These values are used in the project build scripts, the project properties, and the names and locations of the project folders.

### 6.9.1 How to Move, Copy, or Rename a Project

To safely move, copy, and rename a project, you must use the commands provided by the IDE.

**To move, copy, or rename a project:**

- Right-click the project's node in the Projects window and choose one of the following commands:
  - **Rename Project**
  - **Move Project**
  - **Copy Project**

---

**Note:** When moving or copying a project to a new location, the new folder must already exist on the computer. You can create the folder with the **Browse** button.

Moving and renaming a project can break compilation dependencies between projects. To fix the project dependencies, remove the renamed or moved project from the classpath of any other projects, then add the project again

---

## 6.10 Deleting a Project

Deleting a project is a simple procedure and you have the option of deleting all project files or only the build scripts and metadata.

**To delete a project:**

1. Right-click the project's node in the Projects window and choose **Delete Project**.
2. Specify whether to also delete the source folders under the project folder. If you do not select this option, the IDE only deletes the generated build scripts and project metadata.

The IDE does not offer to delete any source folders that are outside of the project folder.

---

# Working with Java Code

This chapter describes how to take advantage of the NetBeans editing tools and search features that help you create and modify the code for your Java applications.

This chapter contains the following sections:

- [About Working with Java Code](#)
- [Editing Java Code](#)
- [Navigating in Java Code](#)
- [Finding and Replacing Text](#)
- [Using Regular Expressions](#)
- [Using Special Code Templates](#)
- [Using Java Editor Shortcuts](#)

## 7.1 About Working with Java Code

Use the Java Source Editor to write or edit Java code. The Java Source Editor is a full-featured text editor that is integrated with the GUI Builder, the compiler, the debugger, and other parts of the IDE. It contains a set of features to enhance your coding experience. For example, you can navigate to specific areas in the source code using accelerator keys or menu commands in the Java Source Editor. You can find and replace regular expressions, Java objects such as methods and interfaces.

In addition, you can use code templates to facilitate coding commonly used sequences of reserved words and common code patterns.

## 7.2 Editing Java Code

The IDE's built-in Source Editor enables you to view, create, and edit your Java source code. Open the Source Editor window by double-clicking a node in the Projects window, Files window, or Navigator window. Alternatively, you can open the Source Editor by choosing **File > New** to create a new file.

---

**Note:** If a file is open in the GUI Builder, click **Source** in the editor's toolbar to open the Source Editor.

---

The IDE has many features to simplify coding of Java files such as:

- **Code completion.** After you press Ctrl+Space, a dialog box appears and offers possible ways of completing the expression you are typing.

- **Code templates.** You can enter common code snippets by typing abbreviations for those snippets and then pressing Tab.
- **Editor hints.** For some common coding mistakes, the Source Editor provides hints for adding the missing code and offers to add the code for you automatically. To display hints, choose **Source > Fix Code** (Alt+Enter).
- **Insert Code.** You can generate getters, setters, properties, and so on. Right-click in the Source Editor and choose **Insert Code** (Alt+Insert) to get a list of items that you can generate.
- **BeanInfo Editor.** You can generate a BeanInfo class for a bean class by right-clicking the bean class in the Project window and choosing BeanInfo Editor. You can then visually edit the BeanInfo class by clicking the **Designer** tab in the Source Editor.
- **Word match.** After you type the beginning characters of a word used elsewhere in your code and then press Ctrl+K, the Source Editor generates the rest of the word.
- **GUI Builder.** Visually design Java GUI applications.
- **Refactoring.** Change the structure of your code without changing the functionality.
- **Navigation Shortcuts.** Include keyboard shortcuts that enable you to navigate to a source file, declaration statement, or Javadoc documentation based on the location of the insertion point.
- **Code Folding.** Enables you to hide sections of code, such as Javadoc comments and method bodies.
- **Pair Completion, Smart Enter, and Smart Semicolons.** When you type a quotation mark, bracket, brace, or parenthesis, the Source Editor automatically inserts the closing character. The matching is "smart", so the closing characters are not duplicated if you type them yourself.
- **Macros.** Record macros by choosing **Edit > Start Recording Macro**.
- **Import Management.** The Ctrl+Shift+I keyboard shortcut helps you generate missing import statements for the file. See

You can find information about other Source Editor features in help topics that are specific to the type of application you are developing.

For information on working with GUI source code, see [Section 11.2.11, "How to Modify GUI Source Code."](#)

For information on working with code formatting in the editor, see [Section 2.13.2, "How to Specify Editor Formatting Options."](#)

### 7.2.1 How to Identify Java Source Files

The IDE uses various icons to identify different types of source files. The following table shows some of the high-level icons used in the Projects and Files windows to indicate Java files and objects. In general, you can find out what an icon represents by holding your cursor over the icon to display the node's tool tip.

Icon	Description
	Java source file. The following file types are represented by this icon: <ul style="list-style-type: none"> <li>▪ Classes, Entity Classes, Interfaces, Exceptions</li> <li>▪ Beans, Filters, Listeners, Servlets</li> <li>▪ Tag Handlers, Service Locators</li> <li>▪ Applets, JApplets</li> </ul>
	Java main class file
	JAR file or WAR file
	Java packages root
	Java package
	Private Java package
	Empty Java package
	Library packages root
	Package library
	Generic information file. This node icon can represent the following file types: <ul style="list-style-type: none"> <li>▪ Java Enum files</li> <li>▪ Java Annotation Type files</li> <li>▪ Java Package Info files</li> <li>▪ Empty files</li> </ul>
	Swing GUI form object. This includes: <ul style="list-style-type: none"> <li>▪ JPanel Forms</li> <li>▪ Bean Forms</li> </ul>
	Swing JInternational Frame form object
	Swing and AWT GUI form object. This includes: <ul style="list-style-type: none"> <li>▪ Frame Forms</li> <li>▪ JFrame Forms</li> </ul>
	Swing and AWT GUI dialog form object. This includes: <ul style="list-style-type: none"> <li>▪ Dialog Forms</li> <li>▪ JDialog Forms</li> </ul>
	Swing and AWT GUI applet form object. This includes: <ul style="list-style-type: none"> <li>▪ Applet Forms</li> <li>▪ JApplet Forms</li> </ul>
	Persistence Unit file
	Database schema file

Icon	Description
	JUnit file. This includes: <ul style="list-style-type: none"> <li>■ JUnit Tests</li> <li>■ Tests for Existing Class</li> <li>■ Test Suites</li> </ul>

Badge icons can be affixed to other file icons to indicate a particular state of the object. The following table lists some of the badges used in the Projects (Ctrl+1) and Files (Ctrl+2) windows.

Icon	Description
	The file needs to be compiled. Either the file has not been compiled or the source has changed since the last compilation.
	The file cannot be parsed. The file might contain an unrecoverable syntactic error, or there might have been a problem reading the file.

## 7.3 Navigating in Java Code

NetBeans provides many editing features that you can use to improve your productivity. These include features for locating and moving to the source code for your projects' classes and interfaces and their members. NetBeans provides keyboard accelerators to step from member to member in a class definition in the Java Source Editor.

For Java files, you can see a list of constructors, methods, and fields. For some kinds of XML files, you can use the Navigator to view the structure of the file. The Navigator window works with standard XML documents, XML schema files, and Ant scripts.

Choose **Window > Navigator** (Ctrl+7) to open this window.

### To display a source file in the Navigator window:

- Select the file in the Source Editor, the Projects window, the Files window, or the Favorites window.

Once a file is displayed, you can use the Navigator window to focus on a specific nodes contained in the document. To do so, click on a given node in the Navigator window, and your cursor automatically relocates to the node's position in the document.

### 7.3.1 Browsing Java Files

When a Java file is selected, a drop-down list appears at the top of the Navigator window to enable you to choose between the following two views:

- Members view that lists, alphabetically, the selected source file's members (constructors, methods, and fields).
- Bean Patterns that lists of a class's bean properties and event listeners.

You can display the Javadoc documentation for a member by hovering the mouse over the member's node. Right-click a node in the Members view to choose the following commands:

- **Go to Source.** Opens the class in the Source Editor and places the insertion point at the beginning of the constructor, method, or field that the node represents.

- **Sort by Name.** Displays the members in alphabetical order.
- **Sort by Source.** Displays the members in the order in which they appear in the code.
- **Filters.** Opens a submenu that enables you to select some common filters for the types of members to display:
  - **Show Inherited Members.** Displays members that come from classes that the current class extends
  - **Show Fields.** Displays the class's fields.
  - **Show Static Members.** Displays static fields and methods.
  - **Show Non Public Members.** Displays members that do not provide public access.

You can also activate and disable filters using the buttons at the bottom of the Navigator window.

The icons in the following table represent elements and bean patterns within the Java hierarchy. These icons are visible from the Navigator window (Ctrl+7).

Icon	Description
	Java class or inner class
	Category node for constructors
	Constructor with default access
	Private constructor
	Public constructor or nonstatic initializer
	Constructor with protected access
	Static initializer
	Category node for methods
	Method with default access (nonstatic)
	Private method (nonstatic)
	Public method (nonstatic)
	Method with protected access (nonstatic)
	Method with default access (static)
	Private method (static)
	Public method (static)
	Method with protected access (static)
	Category node for fields

Icon	Description
	Field with default access (nonstatic)
	Private field (nonstatic)
	Public field (nonstatic)
	Field with protected access (nonstatic)
	Field with default access (static)
	Private field (static)
	Public field (static)
	Field with protected access (static)

### 7.3.2 Browsing XML Files

For XML files, the Navigator view displays a tree of elements. Right-click an element node to choose the following options for the level of detail to have displayed:

- **Show Attributes.** Displays the attributes of each element.
- **Show Content.** Displays the text content of each element.

### 7.3.3 Browsing Ant Scripts

When an Ant build script is selected, a drop-down list appears at the top of the Navigator window to enable you to choose between the following two views:

- **Ant Targets.** Displays an alphabetical list of build targets.
- **XML View.** Displays a hierarchical tree view XML elements in the build script

Select the Ant Targets view and right-click a node and to choose one of the following commands:

- **Open.** Opens the file in the Source Editor and places the insertion point at the beginning of the selected target's line.
- **Run Target.** Runs the selected target.
- **Create Shortcut.** Opens a wizard that enables you to create a menu item, a toolbar item, a keyboard shortcut, and custom code for running the target.
- **Properties.** Displays properties for the target in a property sheet. These properties are not editable.

### 7.3.4 Browsing an XML File

When you select the XML View, right-click a node to choose one of the following options for the level of detail to have displayed:

- **Show Attributes.** Displays the attributes of each element.
- **Show Content.** Displays the text content of each element

### 7.3.5 How to Navigate Within Your Code

While working in NetBeans, you can navigate within your source code and browse source and Javadoc documentations.

**To go to the Java source for the identifier the insertion point is on:**

- Press Alt+O.

Alternatively, you can choose **Navigate > Go to Source**. The class is opened in the Source Editor and the insertion point is placed at the beginning of the constructor, method, or field that the node represents.

**To go to a Java file, JSP file, or tag file using hyperlinks:**

Hold down the Ctrl key and move your over one of the following identifiers and click on it:

- Java identifier (package, class, method, or variable)
- JSP identifier (include directive, <jsp:include> element, <jsp:forward> element, or tag file reference).

**To jump to the declaration for the Java method or field the insertion point is on:**

- Press Alt+G.

Alternatively, right-click and choose **Go To > Declaration** from the context menu.

**To go directly to a Java class:**

- Press Alt+Shift+O.

Alternatively, you can choose **Navigate > Go To Class**.

**To display Javadoc documentation for the entity that the insertion point is on:**

- Press Alt+F.

Alternatively, right-click and choose **Show Javadoc** from the context menu.

Note that to locate the documentation, the appropriate Javadoc library must be included in the Library manager.

**To display Javadoc documentation for the method that corresponds to the method signature that you are using in your code:**

1. Place the insertion point between the closing parenthesis of the method call and the semicolon.
2. Press Alt+F1.

Alternatively, right-click and choose **Show Javadoc** from the context menu.

If the insertion point is elsewhere in the line when you press Alt+F1, the documentation for the method with no parameters is displayed.

**To go to a specific method or field in the currently displayed Java class:**

- Double-click the method or field in the Navigator window.

**To switch Source Editor tabs:**

- Press Alt+Right or Alt+Left.

**To jump back and forth among areas of code in which you have been working:**

- Use the Alt+K and Alt+L jump list keyboard shortcuts.

**To bookmark a line of code:**

- Click anywhere in the line that you want to bookmark and press Ctrl+Shift+M.

Press Ctrl+Shift+Period or Comma to cycle forward or backwards, respectively, through your bookmarks.

Press Ctrl+Shift+M on a bookmarked line to remove the bookmark.

**To go to a specific line number:**

- Press Ctrl+G.

**To select the Projects window node for the current file in the Source Editor:**

- Press Ctrl+Shift+1.

## 7.4 Finding and Replacing Text

From the source editor, you can search for and replace text and expressions in your Java source code quickly and efficiently.

### 7.4.1 How to Find and Replace Text

You can search for text with the option of replacing it across your source file.

**To search a source file currently open in the source editor, with the option to replace text:**

1. In the Source Editor, press Ctrl+F to display the Search panel, at the bottom of the editor, or Ctrl+H to display the Replace dialog box.
2. Type the text to find and replace in the appropriate text fields.

Use the Find combo box or the Replace combo box to select from recent entries. You can open the combo box with the Alt+down arrow and close it with the Alt+up arrow.

3. Use any combination of the following options:
  - **Match Case.** Limits the search to text that has the same capitalization.
  - **Whole Words.** Matches the search text only to whole words in the file. Does not match words that contain but do not exactly equal the search text. Text separated by periods (for example, javax.swing.JPanel) is considered to be separate whole words. For example, if the search word is add and the **Match Whole Words Only** option is not selected, JPanel.add and blueSlider.addChangeListener both contain matches. If **Match Whole Words Only** is selected, only JPanel.add contains a match.
  - **Regular Expressions.** Enables you to search for and replace patterns based on regular expression constructs. For more information, see [Section 7.5, "Using Regular Expressions."](#)
  - **Wrap Around.** Continues the search from the beginning of the file when the end is reached.
  - **Search Selection.** Limits the search to the selected block of text.
  - **Search Backwards.** Searches back to the top of the file.

- **Wrap Around.** Continues the search from the beginning of the file when the end is reached.
  - **Highlight Results.** Highlights all occurrences of the search text in the file. To unhighlight the text, do another search on the same text without this option selected, or press Alt+Shift+H in the Source Editor. Alt+Shift+H toggles between highlighting and unhighlighting the text.
  - **Incremental Search.** Tries to find the text as you type it (instead of waiting until you click the Find button).
4. After the using the Find operation or Replace operation, use these keyboard shortcuts in the Source Editor:

Shortcut	Action
F3	Finds the next occurrence of the search text.
Shift+F3	Finds the previous occurrence of the search text.
Alt+Shift+H	Switches the highlight function on or off.
Ctrl+F3	Searches for the word the insertion point is on, using the last settings specified in the Find dialog box.

For Java classes, use the **Find Usages** command to display usages of a class, method, or field throughout your source code. For information on the **Find Usages** command, see [Section 8.11.2, "How to Find Class, Methods, and Field Usages."](#)

## 7.5 Using Regular Expressions

You can use regular expressions to add flexibility to searches in the IDE. Regular expressions are composed of a special syntax that enables you to express a wide range of search patterns.

In the Find in Projects dialog box, you can use regular expressions to match file names or file text. In the Find dialog box and the Replace dialog box, you can use regular expressions to help you perform search and replace operations.

### 7.5.1 Regular Expression Constructs

[Table 7–1](#) describes some of the special characters in regular expressions:

**Table 7–1 Regular Expression Constructs**

Construct	Description
	Or.
^	Matches text beginning with the subsequent characters.
\$	Matches text ending with the preceding characters.
\	Escape character. Necessary if you want to match to a period (.), bracket ([]), brace (()) or other special character.
\n	New-line character.
\r	Carriage-return character.
[]	Used to delimit a set of characters.
*	Zero or more occurrences of the previous character or set of characters.

**Table 7–1 (Cont.) Regular Expression Constructs**

Construct	Description
+	One or more occurrences of the previous character or set of characters.
.*	Wildcard.

For a more complete list of regular expression constructs and further discussion of regular expressions, see <http://java.sun.com/j2se/1.5.0/docs/api/java/util/regex/Pattern.html#sum>.

### 7.5.1.1 Sample Regular Expressions

Table 7–2 lists some examples of regular expressions that you might use in file searches:

**Table 7–2 Regular Expressions**

Regular Expression	Matches
"[^"\r\n]*"	Matches (quoted) strings in the document.
\{[^{\}]*\}	Inner block of code between braces ({}). Blocks with nested braces are not matched.
[ \t]+\$	All white space at the end of lines.

When using regular expressions in the Replace dialog box, you can use scalers in the **Replace With** field to refer to groups (constructs within parentheses) in the **Find What** field. \$1 is the scaler assigned to the first group, \$2 is assigned to the second group, and so on.

For information on finding classes, methods, and fields, see [Section 8.11.2, "How to Find Class, Methods, and Field Usages."](#)

For information on finding and replacing text, see [Section 7.4.1, "How to Find and Replace Text."](#)

## 7.6 Using Special Code Templates

Use code templates to speed up the entry of commonly used sequences of reserved words and common code patterns. For example, if you enter `forc` and press the Tab key, it expands into the following:

```
for (Iterator it = col.iterator(); it.hasNext();) {
    Object object = it.next();
```

When you create code templates, there are several constructs that you can use to customize the way the code template behaves. You can look at the default IDE code templates in the Options window, under the Tools menu, for examples from which you can learn.

In general, a code template parameter can be specified by its name and a set of optional hints. Hints serve as guidance when computing the values assigned by the infrastructure to the parameters on template expansion. Syntax for parameter definition is as follows:

```
 ${param_name hint=value hint=value ...}
```

However, boolean hints that can be written without the value part:

---

`${param_name hint}` translates to  `${param_name hint=true}`

Some parameter names are reserved by the code template infrastructure:

- `${cursor}` defines the position where the caret will be located after the editing of the code template values finishes.
- `${selection}` defines the position for pasting the content of the editor selection, which is used by the 'selection templates' that appear as hints whenever the user selects text in the editor.

Similarly, some of the hint names are reserved by the code template infrastructure:

- `${param_name default="value"}` defines the parameter's default value.
- `${param_name editable=false}` can be used to disable user's editing of the parameter.
- `${param_name instanceof="java.util.Collection"}` requires the parameter value to be an instance of the given type.
- `${param_name array}` requires the parameter value to be of an array type (including arrays of primitive data types).
- `${param_name iterable}` requires the parameter value to be of an array type or an instance of "java.lang.Iterable". Can be used in 'for-each' cycles.
- `${param type="java.util.Iterator"}` requires the parameter value to be the given type. The infrastructure tries to use short name Iterator and import `java.util.Iterator` if possible.
- `${param_name iterableElementType}` requires the parameter value to be the type of the iterable element. Can be used in 'for-each' cycles.
- `${param_name leftSideType}` requires the parameter value to be the type of the expression on the assignment's left side.
- `${param_name rightSideType}` requires the parameter value to be the type of the expression on the assignment's right side.
- `${param_name cast}` defines that the parameter value would be a type cast if necessary.
- `${param_name newVarName}` defines that the parameter value should be a 'fresh' unused variable name in the given context.

## 7.7 Using Java Editor Shortcuts

[Table 7–3](#) lists the keyboard shortcuts you can use to perform the following Java Source Editor operations:

**Table 7–3 Java Editor Keyboard Shortcuts**

Keys	Action
Ctrl+Shift+B	Goes to the source of the item the insertion point is on.
Ctrl+B	Goes to the method or variable declaration for method or variable the insertion point is on.
Ctrl+Q	Goes to the line in the file where the last change took place.
Ctrl+/	Comments/uncomments the current line or selected lines.
Alt+Shift+F	Formats the selected code or the entire file if nothing is selected.

**Table 7–3 (Cont.) Java Editor Keyboard Shortcuts**

<b>Keys</b>	<b>Action</b>
Alt+Shift+Left	Moves the selected line or lines one tab to the left.
Alt+Shift+Right	Moves the selected line or lines one tab to the right.
Alt+Shift+Up	Moves the selected line or lines one line up.
Alt+Shift+Down	Moves the selected line or lines one line down.
Ctrl+Shift+Up	Copies the selected line or lines one line up.
Ctrl+Shift+Down	Copies the selected line or lines one line down.
Ctrl+Slash	Toggles the commenting out of the current line or selected lines.
Ctrl+Space	Shows code completion box.
Alt+Insert	Pops up a context aware menu that you can use to generate common structures such as constructors, getters, and setters
Alt+Enter	Displays editor hints. The IDE informs you when a hint is available when the light bulb is displayed.
Ctrl+Shift+I	Generates the import statements required by all classes specified in the file.
Alt+Shift+I	Generates the import statements required by the class under the cursor.
Ctrl+P	Selects the next parameter. You must have a parameter selected (highlighted) for this shortcut to work.
Ctrl+Shift+Space	Shows documentation for item under the cursor.
Ctrl+Shift+K	Generates the next word used elsewhere in your code as you type its beginning characters.
Ctrl+K	Generates the previous word used elsewhere in your code as you type its beginning characters.
Ctrl+R	In place rename.

# 8

---

## Building Java Projects

This chapter provides an overview of the building features in NetBeans.

This chapter contains the following sections:

- [About Building Java Projects](#)
- [Working with Ant](#)
- [Working with Builds](#)
- [Building a Java Project](#)
- [Using a Build Server](#)
- [Compiling a Single Java File](#)
- [Building a JAR File](#)
- [Packaging an Application as a Native Installer](#)
- [Preparing a JAR File for Deployment Outside the IDE](#)
- [Using the Output Window](#)
- [Refactoring Java Projects](#)
- [Working with Maven in the IDE](#)
- [Working with Maven Repositories](#)

### 8.1 About Building Java Projects

NetBeans provides both Ant and Maven for building your Java applications. With Ant, if you are using a standard Java project, the IDE generates an Ant build script based on the options you enter in the project's Project Properties dialog box. If you are using a free-form Java project, the IDE uses your existing Ant build script.

With standard Java projects, you can customize the build process by doing any of the following:

- Enter basic options, such as classpath settings and JAR filters, in the Project Properties dialog box.
- Override IDE-generated Ant targets or create new targets in `build.xml`.

By default, the IDE compiles the classes in a standard project when you save them. This compile-on-save feature enables you to run or debug your applications in the IDE without having to wait for the projects to be built. However, the compile-on-save feature does not build your application JAR file. Before delivering your application to

users, use the Clean and Build command to generate fresh versions of the project's distributable files.

For standard projects that have a main class specified, the IDE automatically copies any JAR files on the project's classpath to the `dist/lib` folder when you build the application. The IDE also adds each of the JAR files to the Class-Path element in the application JAR's `manifest.mf` file. This simplifies running the application outside the IDE. For more information, see [Section 8.9, "Preparing a JAR File for Deployment Outside the IDE."](#)

For information on how to customize an Ant build script, see [Section 6.2.3.4, "Customizing the IDE-Generated Ant Script."](#)

For information on how to modify a build JAR file, see [Section 8.7, "Building a JAR File."](#)

Maven is a framework that provides help with managing the project lifecycle, including building and managing dependencies. Maven projects follow a set of standards that are described with a Project Object Model (POM) file. You build the project using its POM and a set of plugins that are shared by all projects using Maven to ensure consistency between projects.

When you build, the IDE executes the plugin goals, builds the project and adds the project artifacts to the local repository. Maven uses repositories (local or remote) to contain a project's build artifacts and dependencies.

By adhering to convention, Maven frees you from having to explicitly specify every build action, configure the path to source files, and so on as it uses a default setup. Even though Maven is based on convention, you can customize a project by creating a custom configuration where you can map IDE actions to Maven goals enabling you to load the project in the IDE with a specific set of dependencies and trigger specific actions.

For information on using Maven in the IDE, see [Section 8.12, "Working with Maven in the IDE."](#)

For information on configuring Maven settings, see [Section 8.12.2, "How to Configure Maven Settings."](#)

For more information on Maven repositories, see [Section 8.13, "Working with Maven Repositories."](#)

## 8.2 Working with Ant

Apache Ant is a Java-based build tool used to standardize and automate build and run environments for development. Ant build scripts are XML files that contain targets, which in turn contain tasks. Ant tasks are executable bits of code that handle the processing instruction for your source code. For example, you use the `javac` task to compile code, the `java` task to execute a class, and so forth. You can use Ant's built-in tasks, use tasks written by third parties, or write your own Ant tasks.

For more information about Ant, see <http://ant.apache.org/>.

For information on installing Ant documentation, see [Section 8.2.9, "How to Install Ant Documentation in the IDE."](#)

### 8.2.1 Using Ant with the IDE

The IDE's project system is built directly on top of Ant version 1.9.0. All of the project commands, like **Build Project** or **Run File** in Debugger, call targets in the project's Ant

script. Therefore, you can build and run your project outside the IDE exactly as it is built and run inside the IDE. For information on mapping Ant targets to debugger commands, see [Section 6.2.4.6, "Mapping an Ant Target to an IDE Command."](#)

You do not need to be familiar with Ant to work with the IDE. You can set all the basic compilation and runtime options in your project's Project Properties dialog box and the IDE automatically updates the project's Ant script. If you know how to work with Ant, you can customize a standard project's Ant script or write your own Ant script for your project. For more information on customizing an Ant script, see [Section 8.2.8, "Ant Classpaths and Custom Tasks."](#)

---

**Note:** Though the IDE is built directly on top of Ant, the parsers that usually come with Ant are not necessarily bundled with the IDE. If you require parsers not included with the IDE distribution, you can add them to Ant's classpath using the **Ant Settings > Additional Classpath** property in the Options window.

---

For more help with the IDE's Ant support, see the NetBeans Ant FAQ at <http://wiki.netbeans.org/>.

For information on modifying the classpath for a project, see [Section 6.2.3.1, "Managing the Classpath."](#)

## 8.2.2 How to Edit an Ant Script

In standard projects the IDE generates the build script based on the options you enter in the **New Project** wizard and the project's **Project Properties** dialog box. You can set all the basic compilation and runtime options in the project's **Project Properties** dialog box and the IDE automatically updates your project's Ant script. If you have additional requirements for the build process that cannot be handled in the **Project Properties** dialog box, you can modify the build script directly.

The main Ant script for a standard project is `build.xml`. The IDE calls targets in `build.xml` whenever you run IDE commands. This file contains an import statement that imports `nbproject/build-impl.xml`, which contains build targets that are generated by the IDE. In `build.xml`, you can override any of the targets from `nbproject/build-impl.xml` or write new targets. Do *not* edit `nbproject/build-impl.xml` directly, because that file is regenerated based on changes that you make in the **Project Properties** dialog box.

In addition, the build script uses the `nbproject/project.properties` which you can edit manually.

Editing an Ant script is much like editing any other XML file. All of the applicable Source Editor shortcuts are available. Double-click any of the Ant script's subnodes in the **Files** window to jump to that target's location in the Source Editor.

The IDE provides code completion for all standard Ant tasks. To enter an end tag for any empty beginning tag, type `</`.

With standard projects, you can customize the build process by doing any of the following:

- Entering basic options, like classpath settings and JAR filters, in the **New Project** wizard when you create a project, or afterwards in the **Project Properties** dialog box.

- Editing properties in nbproject/project.properties. This file stores Ant properties with important information about your project, such as the location of your source and output folders. You can override the properties in this file. Be careful when editing this file. For example, the output folder is deleted every time you clean your project. You should therefore never set the output folder to the same location as your source folder without first configuring the clean target to not delete the output folder.
- Customizing existing or creating new Ant targets by doing any of the following:
  - Add instructions to be processed before or after an Ant target is run. Each of the main targets in nbproject/build-impl.xml also has a -pre and -post target that you can override in build.xml. For example, to get RMI working with regular projects, type the code from [Example 8-1](#) in build.xml.

**Example 8-1**

```
<target name="-post-compile">
  <rmic base="${build.classes.dir}" includes="**/Remote*.class"/>
</target>


- Change the instructions in an Ant target. Copy the target from nbproject/build-impl.xml to build.xml and make any changes to the target.
- Create new targets in build.xml. You can also add the new target to the dependencies of any of the IDE's existing targets. Override the existing target in build.xml and then add the new target to the existing target's depends property. For example, code from Example 8-2 adds the new-target target to the run target's dependencies.

```

**Example 8-2**

```
<target name="new-target">
  <!-- target body... -->
</new-target>

<target name="run" depends="new-target,myprojname-impl.run"/>
You do not need to copy the body of the run target into build.xml.
```

The following table lists some common tasks for redefining a JAR file that you may find useful.

To perform this task	Follow these steps
Specify which files are added to a JAR file.	Right-click the project node in the <b>Projects</b> window and choose <b>Properties</b> . Click the <b>Packaging</b> subnode (under <b>Build</b> ) and configure the filter and compression settings using the <b>Exclude from JAR File</b> field.
Change a JAR file's name and location.	In the <b>Files</b> window, double-click the project's nbproject/project.properties file to open it in the Source Editor. Enter the full path to the JAR file in the dist.jar property.
Specify the manifest file for a JAR file.	In project.properties, type the name of the manifest file in the manifest.file property. The file name must be specified relative to the project's build.xml file. Note that if you are using the Java Application template, the IDE creates a manifest file for you.

---

To perform this task	Follow these steps
Disable the generation of a JAR file for a project.	In the <b>Files</b> window, open your project folder and open <code>build.xml</code> . Override the jar target to have no contents and no dependencies. For example, add the following to <code>build.xml</code> :

```
<target name="jar" />
```

---

The IDE automatically recognizes Ant scripts and displays them as Ant script nodes ( ) rather than as normal XML files. You can right-click Ant scripts in the **Projects** window, **Files** window, or **Favorites** window to access a context menu of commands. You can also expand the Ant script node to see an alphabetical list of subnodes representing the Ant script's targets. Each of these subnodes also has a context menu of commands.

In the **Projects**, **Files**, and **Favorites** windows, an Ant script's subnodes are flagged in the following ways:

---

Icon	Meaning
	<b>Emphasized Ant target.</b> These targets include a description attribute, which is displayed as a tooltip. You define the target's description attribute in the Source Editor.
	<b>Normal Ant target.</b> A target without a description attribute.

---

Double-click any of the Ant script's subnodes to jump to that target's location in the Source Editor. All of the normal XML search tools, selection tools, and keyboard shortcuts are available for editing Ant scripts, and the IDE provides code completion for all standard Ant tasks.

When you create a target that you want to run from the command line, give the target a description attribute. Then, if you forget the names of the targets or what they do, you can run the `ant -projecthelp <script>` command from the command line. With this command, Ant lists only those targets that have a description attribute, together with their descriptions. Especially when there are many targets in your Ant build script, emphasizing some and de-emphasizing others can be a useful way to distinguish between those that you use a lot and those that you use less often.

The font style of a subnode's label in the Projects, Files, and Favorites windows indicates the following:

- **Normal.** A target that is defined within the current Ant script.
- **Italics.** A target that is imported from another Ant script.
- **Greyed out.** An internal target that cannot be run directly. Internal targets have names beginning with `-`.
- **Bold.** The default target for the script, if there is one. The default target is declared as an attribute of the project, together with other project attributes, such as its name. You define the project's default attribute in the Source Editor.

Targets that are imported from another script but are overridden in the importing script are not listed. Only the overriding target is listed.

### 8.2.2.1 Writing Custom Ant Tasks

You can use custom Ant tasks to expand on the functionality provided by Ant's built-in tasks. Custom tasks are often used to define properties, create nested elements, or write text directly between tags using the `addText` method.

#### To create a custom Ant task in the IDE:

1. Right-click the package where you would like to place the task and choose **New > Other**.
2. Select the **Other** category and the **Custom Ant Task** file type.
3. Complete the wizard.

When you create the custom Ant task file, the template opens in the Source Editor. The template contains sample code for many of the common operations performed by Ant tasks. After each section of code, the template also shows you how to use the task in an Ant script.

## 8.2.3 How to Run an Ant Script

The IDE runs targets in your project's Ant script any time you run commands on your project. You can also manually run a target in any Ant script from the Ant script's node in the **Files** window or **Favorites** window.

#### To run a target in an Ant script:

- Right-click the Ant script node and choose the target you want to run from the **Run Target** submenu.

Targets are sorted alphabetically. Only emphasized targets are listed.

Choose **Other Targets** to run a target that has not been emphasized with a `description` attribute. Internal targets are excluded from these lists because they cannot be run independently.

- Select the Ant script node to display its targets in the **Navigator** window. Then right-click the target node and choose **Run Target**.

The **Output** window displays compilation errors generated by Ant scripts as normal Java code compilation errors. You can double-click any error message to jump to the location in the source where the error occurred.

---

**Note:** For free-form projects, you need to modify the Ant script to map IDE commands to Ant targets. For more information, see [Section 6.2.4.6, "Mapping an Ant Target to an IDE Command."](#)

---

#### To stop a running Ant script:

- Choose **Run > Stop Build/Run** from the main menu.

For resources on learning Ant, see <http://ant.apache.org/resources.html>.

#### To install the Ant manual into the IDE help system by using the Plugins Manager:

- Choose **Tools > Plugins** and install the Ant Documentation module.

## 8.2.4 How to Debug an Ant Script

You can run an Ant script in the IDE's debugger like you would a Java class. Debugging an Ant script is handy when you need to diagnose a problem in the Ant script's execution or examine how the script works.

### To debug an Ant script:

- Right-click the Ant script node in the Projects window, Files window, or Favorites window and choose any target from the **Debug Target** menu.

When you start debugging an Ant script, the IDE opens the Debugger windows, goes to the first line of the target, and stops. You can perform all of the following operations:

- Use the **Step Into** (F7), **Step Over** (F8), **Step Out** (Ctrl+F7) commands to go through the Ant script one line at a time.
- View the values of instantiated properties in the Local Variables window.
- Set a watch on a property by choosing **Run > New Watch** (Ctrl+Shift+F7).
- Set a line breakpoint by clicking in the Source Editor's left margin and use the **Continue** (F5) command to run the Ant script to the line.
- Set a conditional breakpoint by choosing **Run > New Breakpoint** (Ctrl+Shift+F8).
- See the hierarchy of nested calls in the Call Stack window.

For information on using the debugger window, see [Chapter 10, "Running and Debugging Java Application Projects."](#)

## 8.2.5 How to Create a Shortcut to a Target

You can create a mini-script that serves as a shortcut to a commonly used target in an Ant script. You can also customize the way the target is run. The shortcut can be saved as a menu item, toolbar button, or keyboard shortcut.

### To create a shortcut to a target:

1. Select the build script node in the Files or Favorites window to display its targets in the Navigation window.
2. Right-click the target node and choose **Create Shortcut**.
3. Specify where you want to place the shortcut and whether you want to customize its Ant code and click **Next**.
4. Use the remaining pages of the wizard to assign the shortcut to a toolbar, menu, or keyboard shortcut. The wizard only shows those steps that you selected in Step 3.  
To ensure you do not overwrite any existing keyboard shortcuts, you can view existing keyboard shortcuts by choosing **Tools > Options** and clicking **Keymap**.
5. In the Customize Script page, add or change any of the shortcut script's elements, such as tasks or properties. This page only appears if you selected the Customize generated Ant code checkbox in Step 3.
6. When you have specified all of the necessary information, click **Finish**. The shortcut is created in the specified locations.

## 8.2.6 How to Configure Ant Settings

You can use the Options window to configure Ant Settings that effect the behavior of Ant in the IDE.

### To configure Ant Settings:

1. From the main window, choose **Tools > Options**.
2. Click the Miscellaneous category and then click the **Ant** tab.
3. Modify the properties as desired.

You can set the following properties:

- **Ant Home.** Displays the installation directory of the Ant executable used by the IDE. You can change Ant versions by typing the full path to a new Ant installation directory in this property. You can only switch between versions 1.5.3 and higher of Ant.

The Ant installation directory must contain a lib/ subdirectory which contains the ant.jar binary. For example, for the standard Ant 1.5.4 release, the Ant installation directory is ant/lib/apache-ant-1.5.4. If you enter a directory that does not match this structure, the IDE gives you an error.

- **Save Files.** If selected, saves all unsaved files in the IDE before running Ant. It is recommended to leave this property selected because modifications to files in the IDE are not recognized by Ant unless they are first saved to disk.
- **Verbosity Level.** Sets the amount of compilation output. You can set the verbosity lower to suppress informational messages or higher to get more detailed information.
- **Always Show Output.** If selected, fronts the Output window tab if the Ant output requires user input or contains a hyperlink. Output that contains hyperlinks usually denotes an error or warning. If not selected, the IDE always fronts the Output window for all Ant processes.
- **Reuse Output Tabs.** If selected, writes Ant output to a single Output window tab, deleting the output from the previous process. If not selected, opens a new tab for each Ant process.
- **Properties.** Configures custom properties to pass to an Ant script each time you call Ant. Click the ellipsis button to open the property editor. This property is similar to the Ant command-line option, -Dkey=value.

The default property, \${build.compiler.emacs}, is available. If you are compiling using Jikes (build.compiler=jikes), setting this property to true enables Emacs-compatible error messages. It is recommended that you leave this property set to true even if you are not using Jikes, since the IDE prefers Emacs-compatible error messages.

- **Classpath.** Contains binaries and libraries that are added to Ant's classpath. For more information, see [Chapter 8.2.8, "Ant Classpaths and Custom Tasks."](#)

## 8.2.7 How to Switch Versions of Ant

The IDE comes bundled with Ant version 1.9.0 and uses this installation to run Ant scripts. You can change the version of Ant that the IDE uses by switching the Ant installation directory in Ant Settings. You can only switch between versions 1.5.3 and higher of Ant.

**To switch the IDE's Ant version:**

1. Choose **Tools > Options** from the main window.
2. Click **Miscellaneous** in the left panel of the window and expand the **Ant** node. The **Ant Home** section displays the current Ant location and version.
3. Click **Manage Ant Home** and select a new Ant installation folder.

The Ant installation directory must contain a `lib/` subdirectory which contains the `ant.jar` binary. If you enter a directory that does not match this structure, the IDE gives you an error.

## 8.2.8 Ant Classpaths and Custom Tasks

By default, the IDE ignores your environment's `CLASSPATH` variable whenever it runs Ant. For your build script to use custom tasks, you must add the tasks to Ant's classpath in the IDE.

**To add custom tasks to Ant's classpath within the IDE:**

- Configure the Ant classpath settings in the Options window.
- Provide an explicit classpath to the tasks in your build script. This is the recommended method.

### 8.2.8.1 Adding Binaries to Ant's Classpath in the IDE

If you cannot declare a classpath in your build script, or you are using third-party build scripts that you cannot alter, you can add the tasks to Ant's classpath in the IDE. Open the Options window, click **Miscellaneous** in the left panel of the window, and expand the **Ant** node. Use the **Classpath** section to manage the Ant classpath.

Only add items to the Ant classpath that are needed to run custom tasks. Do not use the **Classpath** settings to manage the compilation or runtime classpath of your project source folders. For information on managing the project classpath, see [Chapter 6.2.3.1, "Managing the Classpath."](#) For free-form projects, you must declare the classpath in the project settings as the IDE is unaware of project metadata. For more information, see [Chapter 6.2.4.5, "Declaring the Classpath in a Free-Form Project."](#)

### 8.2.8.2 Build Scripts With an Explicit Classpath

Using an explicit classpath is the recommended method as it ensures that your build scripts are fully portable. You can write your tasks and include instructions to compile them and produce a JAR file in the build file. To use these tasks, include the long form of `taskdef`, which includes a classpath. Here is a simple example of such a task:

***Example 8–3 Sample of a Task with an Explicit Classpath***

```
<project name="test" default="all" basedir=".">
    <target name="init">
        <javac srcdir="tasksource" destdir="build/taskclasses"/>
        <jar jarfile="mytasks.jar">
            <fileset dir="build/taskclasses"/>
        </jar>
        <taskdef name="customtask" classname="com.mycom.MyCustomTask">
            <classpath>
                <pathelement location="mytasks.jar"/>
            </classpath>
        </taskdef>
    </target>
</project>
```

The advantage of this method is that no special preparation is needed to begin using the script. The script is entirely self-contained and portable. This method also makes it easier to develop your tasks within the IDE as the script compiles them for you automatically.

To make your build scripts even more robust, use a property instead of a hard-coded location to specify the classpath to your tasks. You can store the property in the build script itself or in a separate `ant.properties` file. You can then change the classpath setting throughout your script by simply changing the value of the specified property.

For information on customizing Ant scripts, see [Chapter 6.2.3.4, "Customizing the IDE-Generated Ant Script."](#)

### 8.2.9 How to Install Ant Documentation in the IDE

Use the Ant documentation plugin to install the Ant manual in the IDE.

**To install the Ant documentation plugin:**

1. Choose **Tools > Plugins** from the main window.
2. Select the **Available Plugins** tab and locate the Ant documentation plugin.
3. Select the **Install** checkbox for the Ant documentation plugin and click **Install**.
4. Follow the wizard instructions to complete the installation of the plugin.

If you receive the `Unable to Connect to the Update Center Server` error message, click **OK** in the dialog box to close it. Click the **Proxy Settings** button in the **Settings** tab of the Plugin manager and set make sure the proxy settings are correct. Click **OK** and try to connect to the update center again.

The official release of the Ant documentation plugin does not always coincide with the release of the IDE. If you do not see the Ant documentation plugin on the NetBeans Update Center, it may be posted on the NetBeans Beta Update Center.

## 8.3 Working with Builds

Before building a project, set the compilation classpath (for details, see [Section 6.2.3.1, "Managing the Classpath."](#)) After the project is built, fix any build errors that occurred. Build errors are displayed in the IDE's Output window. Once all errors in the code have been corrected, clean the `build` and `dist` directories by deleting their contents.

You can examine the build output by viewing compiled classes in the project's `build` directory and the Javadoc files and built libraries, such as JAR and WAR files, in the project's `dist` directory from the Files window.

## 8.4 Building a Java Project

When you build a project, the IDE calls the corresponding target in the project's Ant build script. The IDE compiles the source files and generates the packaged build output, such as a JAR file or WAR file. You can build a project and all of its required projects, or build any project individually.

You do not need to build the project or compile individual classes to run the project in the IDE. By default, the IDE automatically compiles classes when you save them.

These incrementally compiled files are stored in a cache in your user directory and are copied to your project's build folder when you run or debug your project. This

incremental compilation can save you a lot of time when you are editing and testing your project. However, you need to build the project to generate distributable files for the project, such as JAR files.

### 8.4.1 How to Build a Java Project

By default, the **Build Project** command is not enabled since most of that command's functions are handled by the incremental compilation. However, you can use the **Clean and Build** command to create a fresh build. When you clean and build a project, all previous build outputs are deleted and new versions of the build outputs are created.

#### To build a project and its required projects:

1. Select the project that you want to build in the Projects window.
2. Choose **Run > Clean and Build Project** (Shift+F11).

Alternatively, right-click the project's node in the Projects window and choose **Clean and Build**.

The IDE displays the Ant output and any compilation errors in the Output window. Double-click any error to go to the location in the source code where the error occurred.

---

**Note:** If you are building a project often, you can set it as the main project by right-clicking on it in the Projects window and choosing **Set as Main Project** or by choosing **Run > Set Main Project** from the main menu and selecting the project in the sub-menu. After you set a project as the main project, the keyboard shortcuts for **Run** (F6), **Build** (F11) and **Clean and Build** (Shift+F11) apply to the main project regardless of which project is selected in the Projects window. Keyboard shortcuts for **Debug** and **Profile** also apply to the main project.

---

#### To stop building a project:

- Choose **Run > Stop Build/Run** from the main menu.

For standard projects that have a main class specified, the IDE automatically copies any JAR files on the project's classpath to the `dist/lib` folder when you build the project. The IDE also adds each of the JAR files to the `Class-Path` element in the application JAR's `manifest.mf` file. This simplifies running the application outside the IDE. For more information, see [Section 8.9, "Preparing a JAR File for Deployment Outside the IDE."](#)

#### To turn off incremental compilation for a project:

1. Right-click the project's node and choose **Properties**.
2. In the Project Properties dialog box, select the Compiling node and clear the **Compile On Save** checkbox.

You can compile individual files as well entire project files. For information on compiling individual files, see [Section 8.6, "Compiling a Single Java File."](#)

If you need to modify the main class or specify runtime arguments, see [Section 10.6, "Setting the Main Class and Runtime Arguments."](#)

## 8.5 Using a Build Server

The IDE supports creating and starting build jobs using the Hudson build server. Hudson is an open-source server that you can use to build your applications. You can use the Hudson server as part of your continuous integration (CI) process for automated building, verification and testing.

For more information about setting up and using a Hudson build server, see the following documents: <http://hudson-ci.org> and <http://wiki.hudson-ci.org/display/HUDSON/Meet+Hudson>.

The Jenkins build server also works with the IDE, though this is not as well tested.

### To add a Hudson instance:

1. Right-click the Hudson Builders node in the Services window and choose **Add Hudson Instance**.
2. Type the name for the instance to be displayed under the Hudson Builders node.
3. Specify the server URL, the auto-refresh setting.
4. Click **Add**.

After you add a Hudson instance, a node for the instance is added below the Hudson Builders node. You can expand the node to view the status of builds on that instance.

### 8.5.1 Creating a Build

You can specify the application that you want the server to build by creating a new build. When you create the build, you specify the repository containing the sources for the application. A node for the build is added under the node of the target Hudson instance in the Services window. To view additional details about the status of builds, right-click the node for the build and choose **Open** in Browser.

#### To set up a new build job:

1. Choose **Team > Create Build Job** from the main menu.

Alternatively, in the Services window, right-click the Hudson instance you want to use and choose **New Build**.

2. Select the build server instance from the dropdown list.
3. Specify the name for the build job.
4. Select the project from the dropdown list.

The build server will use the sources in the project's repository.

5. Click **Create**.

After you supply the details for the build, you can start the build process on the server by right-clicking the build that you want to start and choosing **Start Job**. When a job is building, the node for the job is displayed as running. You can expand the node for the job to view past builds and build artifacts.

---

**Note:** To create builds and start jobs you must have access to a Hudson server.

---

## 8.6 Compiling a Single Java File

By default, you do not need to manually compile the files to run your application in the IDE. Files in a standard Java project are compiled automatically when you save the files.

If you have turned off the **Compile on Save** feature for a project, you can compile files by building the project or by compiling individual files.

### To compile an individual file:

- Select the file in the Projects window, Files window, or in the Source Editor and choose **Run > Compile File (F9)**.

If you are using a free-form project, you need an Ant target for this command. The IDE offers to generate a target the first time you choose the command. In the generated target, specify the directory where to put the compiled class. You can do so by specifying a value for the `build.classes.dir` property in the generated target. For example, you might add the following line to the line above the `<target name="compile-selected-files-in-src">` entry:

```
<property name="build.classes.dir" value="build"/>
```

Alternatively, you can replace the value of the provided `build.classes.dir` or rewrite the target entirely.

For information on managing the classpath, see [Section 6.2.3.1, "Managing the Classpath."](#)

## 8.7 Building a JAR File

In standard project, the IDE builds a JAR file from your project sources every time you run the **Build** command or the **Clean and Build** command. The JAR file is generated to the `dist` directory of your project folder.

For standard projects where the main class is specified, the IDE automatically copies any JAR files on the project's classpath to the `dist/lib` folder when you build the project. The IDE also adds each of the JAR files to the `Class-Path` element in the application JAR's `manifest.mf` file. This simplifies running the application outside the IDE. For more information, see [Section 8.9, "Preparing a JAR File for Deployment Outside the IDE."](#)

When you create a JAR file or a WAR file, you usually want to include just the compiled `.class` files and any other resource files located in your source directory, such as resource bundles or XML documents. The default filter does this for you by excluding all `.java`, `.nbattrs`, and `.form` files from your output file.

In addition to the default expressions, you can create additional filters using regular expressions to control the output files.

Regular Expression	Description
<code>\.html\$</code>	Exclude all HTML files
<code>\.java\$</code>	Exclude all Java files
<code>(\.\html\\$)   (\.\java\\$)</code>	Exclude all HTML and Java files
<code>(Key)   (\.\gif\\$)</code>	Exclude all GIF files and any files with Key in their name

**To specify which files are added to the JAR file:**

1. Right-click the project node in the Projects window and choose **Properties**.
2. Select the **Packaging** node in the dialog's left pane.
3. Specify the files to exclude and any additional settings in the right pane.

**To disable generation of a JAR file for a project:**

1. In the Files window, open your project folder and open build.xml.
2. Override the jar target to have no contents and no dependencies. For example, add the following to build.xml:

```
<target name="jar" />
```

Alternatively, if you deselect the **Build JAR after Compiling** option in the Packaging category of the Properties window the Java compiler will produce only .class files, without building JAR files.

---

**Note:** In free-form projects, JAR file creation is controlled by your Ant script.

---

See [Section 6.2.3.3, "Editing IDE-Generated Ant Scripts"](#) for a description of standard project Ant scripts and properties files. For information on customizing an Ant build script, see [Section 6.2.3.4, "Customizing the IDE-Generated Ant Script."](#)

## 8.8 Packaging an Application as a Native Installer

NetBeans IDE supports native packaging for standard Java SE and JavaFX projects. A native package is a wrapper for your project that turns the project into a self-contained, platform-specific installation package. The package contains all the artifacts (source code, Javadoc, Java Runtime and/or JavaFX runtime, a native application launcher, and so on) needed to install and run the application.

Native packaging does not affect the deployment model of a Java SE or a JavaFX project. It enables your project to be packaged with the Java runtime to produce an installer that is common for the operating system you are working in.

Some of the benefits to using native packaging are:

- Negates runtime version conflicts since the specific JRE is included in the bundle
- Allows for easy deployment of the application using enterprise deployment tools
- Can be distributed as a .zip file or packaged as a platform-specific installer

Before opting for native packaging, however, be aware of the following considerations:

- Users might have to go through several steps to download and install the package
- Due to the inclusion of the Java runtime (JavaFX runtime for JavaFX applications), the size of the application is significantly larger than a standard packaged application
- Application formats are platform-specific (to create a bundle for different operating systems, you must build your project on each platform)
- No autoupdate support exists for bundled applications

The following types of packages can be built for the following platforms:

- Windows
  - bundle image
  - EXE installer (requires Inno Setup 5 or later)
  - MSI installer (requires WiX 3.0 or later)
- MAC OS X:
  - bundle image
  - DMG installer
- Linux:
  - bundle image
  - rpm installer (requires rpmbuild)

Note that native packaging might require external tools. For instance, if you want to package a Java application to distribute to a non-Windows environment, you need the `appbundler` tool, which is not included in JDK7update 6.

Deployment of a Java SE applet (or application) intended to be launched from a web browser and deployment of a JavaFX application are similar in that when a Java SE applet or a JavaFX application are built, both a Java Network Launching Protocol (JNLP) file and a sample html version of the applet are generated. The WebStart engine reads the JNLP file and executes the applet accordingly. The html file defines how the applet is embedded in the web page.

For web page deployment, Java SE and JavaFX applications require the following packaging tools:

- `bin/javafxpackager` (command-line utility to produce Java SE or JavaFX packages)
- `lib/ant-javafx.jar` (set of Ant tasks to produce Java SE or JavaFX packages)

These packaging tools are available with JDK version 7 update 6 and later.

For more information on deploying a Java Web Start Application, see Deploying a Java Web Start Application at

<http://docs.oracle.com/javase/tutorial/deployment/deploymentInDepth/createWebStartLaunchButtonFunction.html>.

For information on using JavaFX packaging tools, see Deploying JavaFX Applications: <http://docs.oracle.com/javafx/2/deployment/jfxpub-deployment.htm>.

For information on creating native packaging of a Java SE project in NetBeans, see Packaging a Distributable JavaApp at <http://wiki.netbeans.org/PackagingADistributableJavaApp>.

For more information on native packaging, see the Native packaging for JavaFX page at the following URL:

[https://blogs.oracle.com/talkingjavadeployment/entry/native\\_packaging\\_for\\_javafx](https://blogs.oracle.com/talkingjavadeployment/entry/native_packaging_for_javafx)

#### To enable native packaging actions in the project context menu:

1. Right-click the project node in the Projects window and select **Properties** from the context menu.
2. In the Project Properties dialog, choose the Deployment category.
3. Select the **Enable Native Packaging Actions** option.

**4. Click OK.**

A Package as command is added to the project's context menu. When you are ready to package your application, right-click the project, choose Package as, and select one of the packaging types:

- **All Artifacts** - Creates a package that contains only the project artifacts
- **All Installers** - Creates a package that contains the application image and all applicable installers
- **Image Only** - Creates a package that contains the application image only
- **EXE Installer** - (Windows only) Creates a package as an executable .exe installer
- **MSI Installer** - (Windows only) Creates a package as an executable .msi installer

---

**Note:** Contents of the package types is dependent on the platform on which the IDE is running.

---

When you select a packaging type, a subdirectory is created under the project's dist directory where the package is stored.

Be aware that the packaging process can take some time to complete and, depending on the tools you are using, progress might not be indicated.

Note that the contents of the submenu are dependent on the operating system you are running on. For example, if running on Linux, an RPM installer is available instead of EXE or MSI. If running on MAC OS, a DMG installer is available.

## 8.9 Preparing a JAR File for Deployment Outside the IDE

Whenever you build a standard Java project for which a main class is specified, the IDE automatically copies any JAR files on the project's classpath to the dist/lib folder. The IDE also adds each of the JAR files to the Class-Path element in the application JAR's manifest.mf file. This simplifies running the application outside the IDE.

Though the IDE copies the necessary files to the dist/lib directory automatically, the following special cases should be kept in mind:

- If two JAR files on the project classpath have the same name, only the first JAR file is copied to the lib folder.
- If you added a folder of classes or resources to the classpath (as opposed to a JAR file or project), none of the classpath elements are copied to the dist folder.
- If a library on the project's classpath also has a Class-Path element specified in its manifest, the content of the Class-Path element must be on the project's runtime path.

### 8.9.1 Running an Application JAR Outside of the IDE

Once you have distributed the archive of your application, the application can be run outside of the IDE from the command line.

**To run an application JAR file from the command line:**

1. Navigate to the project's dist folder.
2. Type the following:

```
java -jar <jar_name>.jar
```

When you run the **jar** command, the JAR tool uses the JAR manifest to determine the application entry point and the paths to the dependent binaries that are specified in the `manifest.mf` file.

## 8.10 Using the Output Window

The Output window is a multi-tabbed window that displays messages from the IDE. This window is displayed automatically when you encounter compilation errors, debug your program, generate Javadoc documentation, and so on. You can also open this window by choosing **Window > Output** (Ctrl+4).

One function of the Output window is to notify you of errors found while compiling your program. The error message is displayed in blue underlined text and is linked to the line in the source code that caused the error. The Output window also provides links to errors found when running Ant build scripts and when checking and validating XML documents. For information on validating XML documents, see [Section 18.2.5, "How to Validate an XML Document."](#)

If the file that contains the error is open, the Source Editor jumps to the line containing each error as you move the insertion point into the error in the Source Editor. You can also use the F12 and Shift+F12 keyboard shortcuts to move to the next and previous error in the file.

Every action that is run by an Ant script, such as compiling, running, and debugging files, sends its output to the same **Output window** tab. If you need to save some output, you can copy and paste it to a separate file.

**To set Ant to print the command output for each new target to a new Output window tab:**

1. Choose **Tools > Options**.
2. Click Miscellaneous in the left panel of the window, expand the Ant node, and select the checkbox in the Reuse Output Tabs property.

When you run a program that requires user input, a new tab appears in the Output window. This tab includes a cursor. You can enter information in the Output window as you would on a command line.

## 8.11 Refactoring Java Projects

*Refactoring* is the use of small transformations to restructure code without changing any program behavior. Just as you factor an expression to make it easier to understand or modify, you refactor code to make it easier to read, simpler to understand, and faster to update. And just as a refactored expression must produce the same result, the refactored program must be functionally equivalent with the original source.

Some common motivations for refactoring code include:

- Making the code easier to change or easier to add a new feature.
- Reducing complexity for better understanding.
- Removing unnecessary repetition.
- Enabling use of the code for other needs or more general needs.
- Improving the performance of your code.

The IDE's refactoring features simplify code restructuring by evaluating the changes that you want to make, showing you the parts of your application that are affected, and making all necessary changes to your code. For example, if you use the Rename operation to change a class name, the IDE finds every usage of that name in your code and offers to change each occurrence of that name for you.

For information on how to refactor an Enterprise Bean, see [Section 8.11.22, "How to Refactor an Enterprise Bean."](#)

### 8.11.1 How to Undo Refactoring Changes

You can undo any changes that you made using the commands in the Refactor menu. When you undo a refactoring, the IDE rolls back all the changes in all the files that were affected by the refactoring.

**To undo a refactoring command:**

1. Go to the Refactor menu in the main menu bar.
2. Choose **Undo**.

The IDE rolls back all the changes in all the files that were affected by the refactoring.

If any of the affected files have been modified since the refactoring took place, the Refactoring Undo is not available.

### 8.11.2 How to Find Class, Methods, and Field Usages

Use the **Find Usages** command to determine everywhere a class, method, or field is used in your project's source code.

**To find where a class, interface, method, or field is used in your project:**

1. In the Projects window or the Source Editor window, right-click the code element and choose **Find Usages** (Alt+F7).
2. In the Find Usages dialog box, select options for the scope of the search:
  - Classes and interfaces
  - Methods
  - Fields
3. Click **Find**.

The Usages window displays the file name and the line of code for each usage found in that file.

**To jump to a specific occurrence of the code element:**

- Double-click a file name in the Usages window to open the file.
- Double-click a line of code to open the file and to position the cursor on that line of code.

#### 8.11.2.1 Classes and Interfaces

For classes and interfaces, the **Find Usages** command displays all the code lines that:

- Use the type, such as creating a new instance, importing, extending, implementing, casting, or throwing.

- Use the type's members and static variables

After you choose the Find Usages command on a class or interface, the Find Usages dialog box might give you additional options:

- **Find All Subtypes** checkbox. If selected, only usages of subtypes of the class are displayed.
- **Find Direct Subtypes Only** checkbox. If selected, only usages of direct subtypes are displayed. Subtypes of those subtypes are ignored.

### 8.11.2.2 Methods

For methods, the **Find Usages** command displays the code lines that:

- Call the method
- Override the method

After you choose the Find Usages command on a method, the Find Usages dialog box might give you additional options:

- **Include overloaded methods** checkbox. If selected, any occurrences of overloaded methods are displayed.
- **Search from Base Class** checkbox. If selected, the output shows every usage of that base method. This option only appears if the method that you are finding usages for overrides another method.

### 8.11.2.3 Fields

For fields, the **Find Usages** command displays the code lines that:

- Set the field to a value
- Get the value of a field

### 8.11.2.4 Additional Find Mechanisms

Other IDE tools that enable you to search for all the places where specific text is used in a project include:

- **Finding and Replacing Text.** Searches for all the places where specific text is used in a source file that is open in the Java Editor. Choose **Edit > Find** to open the Find dialog box, or choose **Edit > Replace** to open the Replace dialog box. These commands finds all matching strings, regardless of whether the string is a Java element.
- **Find in Projects.** As with the **Find** command, the **Find in Projects** command searches for matching strings, regardless of whether the string is a class name. Choose **Edit > Find in Projects** to open the Find in Projects dialog box and then type the string of text that you are looking for.

---

**Note:** To find where a method is declared in a source file, you can either double-click the method in the Projects window or Navigator window. If the method is declared in a different source file, right-click the method and choose **Go To > Declaration** from the context menu.

---

## 8.11.3 How to Rename a Class or Interface

Use the refactoring **Rename** command to rename a class or interface. When you rename a class or interface through refactoring, all instances of that class or interface

are renamed throughout the project. If you do not want to enact a global change, you can simply select the desired class or interface and edit its name.

**To rename a class or interface and update references to it throughout your project:**

1. In the Projects window or the Source Editor window, right-click the class or interface and choose **Refactor > Rename** from the contextual menu.
2. In the Rename dialog box, type the new name of the class or interface.
3. Click **Next**.
4. In the Refactoring window, review the lines of code that are affected by the change and clear the checkbox of any code that you do not want changed.
5. Click **Do Refactoring** to apply the selected changes.

Always perform a clean build after completing any refactoring commands. Clean a build by right-clicking the project's node in the Projects window and choosing **Clean and Build Project**.

**To rename the class or interface without refactoring:**

1. In the Projects window, rename the class's node inline.

You can rename a class this way by selecting the node of the class and then pressing F2 to make the name editable.

2. In the Rename dialog box, select the **Rename Without Refactoring** checkbox.
3. Click **Next**.

If you need to back out a refactoring change, see [Section 8.11.1, "How to Undo Refactoring Changes."](#)

#### 8.11.4 How to Rename a Field or Method

Use the refactoring **Rename** command to rename a field or method. As with renaming a class or interface, the scope of a renaming operation is the full scope of the element in the project. Field and method usages are replaced everywhere they appear in the project. Parameters and variables are renamed only in the lexical scope of their definitions. Other elements with the same name are not modified.

**To rename a field or method and update references to it throughout your project:**

1. In the Source Editor, right-click the field or method and choose **Refactor > Rename** from the context menu
2. In the Rename dialog box, type the new name of the field or method.
3. (Optional) Click **Preview**. In the Refactoring window, at the bottom of the Source Editor, review the lines of code that are affected by the change and clear the checkbox of any code that you do not want changed.
4. Click **Do Refactoring** to apply the selected changes.

For quick in-place renaming, place the cursor in the item that you want to rename, and press Ctrl+R. Type the new name. Press **Escape** to finish renaming.

Always perform a clean build after completing any refactoring commands. You can do a clean build by right-clicking the project's node in the Projects window and choosing **Clean and Build Project**.

For information on how to undo refactoring, see [Section 8.11.1, "How to Undo Refactoring Changes."](#)

### 8.11.5 How to Move a Class to Another Java Package

Use the refactoring **Move** command to move a class to another package. All references to the class are updated the next time you build the project.

**To move a class to another package and to change the code that references that class:**

1. In the Projects window or the Source Editor window, right-click the class and choose **Refactor > Move** from the context menu.
2. In the Move Class dialog box, select the package from the To Package drop-down list or type the fully qualified package name, such as com.myCom.myPkg.
3. Click **Refactor** or **Preview**.

If you click **Refactor**, the IDE applies the changes automatically and skips the remaining steps.

If you click **Preview**, the Refactoring window displays the lines of code that are affected. Review the list and clear the checkbox of any code that you do not want changed. If the class that you are pushing members from has multiple subclasses and you do not want the members to be pushed to all of them, be sure to clear the checkboxes for the corresponding subclasses. Click **Do Refactoring** to apply the selected changes.

You can also initiate the moving of a class by dragging a class's node in the Projects window to another package's node or by cutting and pasting a class's node.

Always perform a clean build after completing any refactoring commands. Clean a build by right-clicking the project's node in the Projects window and choosing **Clean and Build Project**.

**To move a class without doing refactoring:**

1. In the Projects window, manually move the class to another package.  
You can cut and paste the class or drag and drop it into another package.
2. In the Move Class dialog box, select the **Move Without Refactoring** checkbox.
3. Click **Next**.

You can also use the **Move** command to move the methods and fields of a selected class to another package.

**To move the members of a class:**

1. In the Projects window or the Source Editor window, right-click the method and choose **Refactor > Move** from the context menu.
2. In the Move Members dialog box, select the package from the drop-down list or type the fully qualified package name, such as com.myCom.myPkg.
3. Enter the class name or select the target class from the drop-down list.
4. Select the members you want to move.
5. Select the visibility level.
6. Select whether to keep the Javadoc as it is or update it.

7. Check the option if you want to keep the original method signatures in the source class and let them call the new methods in the target class.
8. Check the option if you want the original methods in the source class to be deprecated.
9. Click Refactor or Preview.

If you click **Refactor**, the IDE applies the changes automatically and skips the remaining steps.

If you click **Preview**, the Refactoring window displays the lines of code that are affected. Review the list and clear the checkbox of any code that you do not want changed. If the class that you are pushing members from has multiple subclasses and you do not want the members to be pushed to all of them, be sure to clear the checkboxes for the corresponding subclasses. Click **Do Refactoring** to apply the selected changes.

### 8.11.6 How to Move an Inner Class One Level Up

Use the **Move Inner to Outer Level** command to do move an inner class one level up in hierarchy. For example, if the selected class is directly nested in a top-level class, a new top-level class is created. If the selected class is nested in an inner class, the selected class is moved to the level of the inner class in which it was nested.

#### To use the Move Inner to Outer Level operation:

1. In the Source Editor, place the insertion point in the inner class that you want to convert.
  2. Choose **Refactor > Move Inner to Outer Level**.
- The Move Inner to Outer Level dialog box appears.
3. In the Class Name field, change the name of the class, if necessary.
  4. (Optional) Select the **Declare Field for the Current Outer Class** checkbox if you want to generate an instance field for the current outer class and pass the outer class to the constructor. If you select this checkbox, type a name for the outer class' instance field.
  5. Click Refactor or Preview.

If you click **Refactor**, the IDE applies the changes automatically and skips the remaining steps.

If you click **Preview**, the Refactoring window displays the lines of code that are affected. Review the list and clear the checkbox of any code that you do not want changed. If the class that you are pushing members from has multiple subclasses and you do not want the members to be pushed to all of them, be sure to clear the checkboxes for the corresponding subclasses. Click **Do Refactoring** to apply the selected changes.

Always perform a clean build after completing any refactoring commands. You can do a clean build by right-clicking the project's node in the Projects window and choosing **Clean and Build Project**.

### 8.11.7 How to Move a Class Member to a Superclass

Use the **Pull Up** command to move methods and fields to a class that their current class inherits from.

**To move a class member to a superclass:**

1. In the Source Editor or Projects window, select the class that contains members that you want to move.
2. Choose **Refactor > Pull Up**.

The Pull Up dialog box appears and displays a list of the class's members and any interfaces that the class implements.

3. In the Destination Supertype drop-down list, select the class to which you want to move the members.
4. Select the checkbox for the member or members that you want to move.

If the current class implements any interfaces, there are checkboxes for these interfaces. If you select a checkbox for an interface, the implements statement for that interface is moved to the superclass.

5. (Optional) If you want to make a method abstract, select the **Make Abstract** checkbox for the method. If you select this checkbox, the method will be declared in the superclass as an abstract method and overridden in the current class. The method will be assigned the protected access modifier.

6. Click **Refactor or Preview**.

If you click **Refactor**, the IDE applies the changes automatically and skips the remaining steps.

If you click **Preview**, the Refactoring window displays the lines of code that are affected. Review the list and clear the checkbox of any code that you do not want changed. If the class that you are pushing members from has multiple subclasses and you do not want the members to be pushed to all of them, be sure to clear the checkboxes for the corresponding subclasses. Click **Do Refactoring** to apply the selected changes.

Always perform a clean build after completing any refactoring commands. You can do a clean build by right-clicking the project's node in the Projects window and choosing **Clean and Build Project**.

### 8.11.8 How to Move a Class Member to a Subclass

Use the **Push Down** command to move inner classes, methods, and fields to all subclasses of their current class.

**To move a class member to a subclass:**

1. In the Source Editor or Projects window, select the class member or class members that you want to move.
2. Choose **Refactor > Push Down**.

The Push Down dialog box appears and displays a list of the class's members. Make sure the checkbox for the member that you want to move is selected.

3. (Optional) Select the **Keep Abstract** checkbox for any abstract methods that you want to keep defined in the current class and have implemented in the subclass. The checkbox in the left column must also be checked for the class definition to be copied to the subclass.
4. Click **Refactor or Preview**.

If you click **Refactor**, the IDE applies the changes automatically and skips the remaining steps.

If you click **Preview**, the Refactoring window displays the lines of code that are affected. Review the list and clear the checkbox of any code that you do not want changed. If the class that you are pushing members from has multiple subclasses and you do not want the members to be pushed to all of them, be sure to clear the checkboxes for the corresponding subclasses. Click **Do Refactoring** to apply the selected changes.

Always perform a clean build after completing any refactoring commands. You can do a clean build by right-clicking the project's node in the Projects window and choosing **Clean and Build Project**.

### 8.11.9 How to Copy a Class

Use the refactoring **Copy** command to copy a class.

**To copy a class, either to the same or to another package, and to change the code that references that class:**

1. In the Projects window or the Source Editor window, right-click the class and choose **Refactor > Copy** from the contextual menu.
2. In the Copy Class dialog box, select the package from the To Package combo box or type the fully qualified package name, such as `com.myCom.myPkg`.
3. Click **Refactor** or **Preview**.

If you click **Refactor**, the IDE applies the changes automatically and skips the remaining steps.

If you click **Preview**, the Refactoring window displays the lines of code that are affected. Review the list and clear the checkbox of any code that you do not want changed. If the class that you are pushing members from has multiple subclasses and you do not want the members to be pushed to all of them, be sure to clear the checkboxes for the corresponding subclasses. Click **Do Refactoring** to apply the selected changes.

You can also initiate the copying of a class by dragging a class's node in the Projects window to another package's node or by cutting and pasting a class's node.

Always perform a clean build after completing any refactoring commands. You can do a clean build by right-clicking the project's node in the Projects window and choosing **Clean and Build Project**.

**To copy a class without doing refactoring:**

1. In the Projects window, manually copy the class to another package.  
You can cut and paste the class, or you can use drag and drop.
2. In the Copy Class dialog box, select the **Copy Without Refactoring** checkbox.
3. Click **Next**.

### 8.11.10 How to Encapsulate a Field

*Field encapsulation* is the act of restructuring your code so that a field is accessed only by a pair of accessor methods. Accessor methods are also referred to as read/write methods or getters and setters.

Typically when you encapsulate a field, you change the field's access modifier to private so that the field can not be directly referenced from outside of the class. For other classes to reference the field, they have to use the accessor methods.

Use the **Encapsulate Fields** command to:

- Generate accessor methods for fields. The names of these methods take the form of `getfield-name` and `setfield-name`.
- Adjust the access modifier for the fields.
- Replace direct references in your code to the fields with calls to the accessor methods.

**To encapsulate a field:**

1. In the Source Editor, right-click a field or a reference to the field and choose **Refactor > Encapsulate Fields** from the context menu.
2. In the List of Fields to Encapsulate table in the Encapsulate Fields dialog, make sure the checkbox for the field that you want to encapsulate is selected. You can select multiple fields.
3. (Optional) Set the field's visibility.
4. (Optional) Set the accessors' (getter and setter) visibility.
5. (Optional) If you do not want the IDE to replace code to use the accessor methods, clear the **Use Accessors Even When Field is Accessible** checkbox.

This option only has an impact if both of the following are true:

- You have direct references to the field in your code.
  - You have set the field's accessor modifier so that the field is visible to the classes with these references.
6. Click **Refactor or Preview**.

If you click **Refactor**, the IDE applies the changes automatically and skips the remaining steps.

If you click **Preview**, the Refactoring window displays the lines of code that are affected. Review the list and clear the checkbox of any code that you do not want changed. If the class that you are pushing members from has multiple subclasses and you do not want the members to be pushed to all of them, be sure to clear the checkboxes for the corresponding subclasses. Click **Do Refactoring** to apply the selected changes.

Always perform a clean build after completing any refactoring commands. You can do a clean build by right-clicking the project's node in the Projects window and choosing **Clean and Build Project**.

### 8.11.11 How to Change a Method's Signature

Use the **Change Method Parameters** command to alter the signature of a method and have those changes propagated in all of the code that calls this method. Specifically, you can:

- Add parameters to a method.
- Reorder the parameters in a method signature.
- Change the access modifier for the method.

**To add a parameter to a method:**

1. Right-click the method in the Source Editor and choose **Refactor > Change Method Parameters** from the contextual menu.

2. In the Change Method Parameters dialog, click the **Add** to add a parameter.
3. In the Parameters table, modify the name and type of the parameter that you have added. Then add a default value for the parameter in the Value column. You need to double-click a cell to make it editable.
4. Click **Refactor** or **Preview**.

If you click **Refactor**, the IDE applies the changes automatically and skips the remaining steps.

If you click **Preview**, the Refactoring window displays the lines of code that are affected. Review the list and clear the checkbox of any code that you do not want changed. If the class that you are pushing members from has multiple subclasses and you do not want the members to be pushed to all of them, be sure to clear the checkboxes for the corresponding subclasses. Click **Do Refactoring** to apply the selected changes.

Always perform a clean build after completing any refactoring commands. You can do a clean build by right-clicking the project's node in the Projects window and choosing **Clean and Build Project**.

#### To reorder a parameter in a method signature:

1. Right-click the method in the Source Editor and choose **Refactor > Change Method Parameters** from the context menu.
  2. Select a parameter that you want to move and click **Move Up** or **Move Down** to change its position in the list.
  3. Click **Next**.
- If you click **Refactor**, the IDE applies the changes automatically and skips the remaining steps.
4. The Refactoring window displays the lines of code that are affected. Review the list and clear the checkbox of any code that you do not want changed.
  5. Click **Do Refactoring** to apply the selected changes.

#### To change a method's access modifier:

1. Right-click the method in the Source Editor and choose **Refactor > Change Method Parameters** from the contextual menu.
  2. Choose a modifier from the Visibility Modifier combo box.
  3. Click **Refactor** or **Preview**.
- If you click **Refactor**, the IDE applies the changes automatically and skips the remaining steps.
- If you click **Preview**, the Refactoring window displays the lines of code that are affected. Review the list and clear the checkbox of any code that you do not want changed. If the class that you are pushing members from has multiple subclasses and you do not want the members to be pushed to all of them, be sure to clear the checkboxes for the corresponding subclasses. Click **Do Refactoring** to apply the selected changes.
- Always perform a clean build after completing any refactoring commands. You can do a clean build by right-clicking the project's node in the Projects window and choosing **Clean and Build Project**.

### 8.11.12 How to Invert a Boolean Method or Variable

When testing for a logical condition, it can be helpful to replace a Boolean method or variable with an opposite definition.

**To invert a Boolean method or variable:**

1. Select the method or variable in the Source Editor.
2. Choose Refactor > Invert Boolean.
3. Modify the name of the method or variable in the New Name field.
4. Click Refactor.

If you click **Refactor**, the IDE applies the changes automatically and skips the remaining steps.

If you click **Preview**, the Refactoring window displays the lines of code that are affected. Review the list and clear the checkbox of any code that you do not want changed. If the class that you are pushing members from has multiple subclasses and you do not want the members to be pushed to all of them, be sure to clear the checkboxes for the corresponding subclasses. Click **Do Refactoring** to apply the selected changes.

Always perform a clean build after completing any refactoring commands. You can do a clean build by right-clicking the project's node in the Projects window and choosing **Clean and Build Project**.

### 8.11.13 Replacing a Constructor

You can replace a public constructor with references to a newly created builder class. Use the **Replace Constructor With Builder** command to introduce a builder class that calls the hidden constructor and replaces existing calls to the constructor with calls to the builder class.

**To replace a constructor with a builder class:**

1. Select a constructor in the Source Editor.
2. Choose **Refactor > Replace Constructor with Builder**
3. Edit the setter name and default value as needed by double-clicking the appropriate column.
4. Check the Optional Setter option if you want to omit a setter method in the builder invocation if the default value of a field matches the parameter value in the constructor invocation.
5. Specify the name of the builder class.
6. Click **Do Refactoring**.

You can also replace a public constructor with a factory method. Use the Replace Constructor with Factory command to introduce a new static factory method that calls the constructor and replaces existing calls to the constructor with calls to the new factory method. After replacing a public constructor with a factory method, the constructor is made private.

**To convert a constructor into a factory method:**

1. Select a constructor in the Source Editor.
2. Choose **Refactor > Replace Constructor with Factory**.

3. Specify a factory method name.
4. Click **Do Refactoring**.

For either operation, you can preview the changes before initiating refactoring. Click **Preview** and check the items displayed in the Refactoring window. Clear the checkboxes for the parts of the code the you do not want changed. Click **Do Refactoring** to make the changes.

#### 8.11.14 How to Introduce a Variable, Constant, Field, or Method

When you introduce a variable, constant, field, or method in the IDE, you change a selected code fragment into a variable, constant, field, or method. Typically you do this when you want to separate a piece of code into smaller, more meaningful fragments. Creating smaller fragments can increase the reusability of your code as you can separate the parts of your code that may need to be updated more often. By giving your new method a meaningful name, you can increase the comprehensibility of your code.

For example, when you introduce a method in the IDE, you replace statements in a class with a call to a method. Before statements are replaced, the IDE opens the Introduce Method dialog box where you specify the parameters and modifiers for the method. The IDE searches your open projects for occurrences of the statements you specified and replaces the occurrences with the method call.

##### To introduce a variable, constant, field, parameter, or method:

1. In the Source Editor, select the statements you want to introduce as a new variable, constant, field, or method.
2. Choose **Refactor > Introduce**.  
Choose the appropriate menu item, such as Introduce Method.
3. Type the name for your new item in the **Name** field and choose the access type.
4. Provide the information required for the item.  
For example, to introduce a method, you must choose an access type.
5. Select the **Replace Also Other Occurrences** to replace all occurrences of the selected item with the new reference
6. Click **OK** to apply the changes to the selected files.

In situations where you need to add multiple methods to a class but cannot modify the class, you can create a new class that contains these methods. You can this class as a local extension that is either a subclass or wrapper of the original class.

##### To introduce a local extension:

1. In the Source Editor, select the class for which you want an extension.
2. Choose **Refactor > Introduce > Local Extension**.  
The project and source folder name are automatically entered in the dialog box. Use the drop-down list to change the values if needed.
3. Change the name of the new class to be easily identified.
4. Specify the name of package to store the extension class.
5. Select whether the extension class is a wrapper or subclass of the original class.
6. Select the Equality type if the wrapper option is chosen.

7. (Optional) Replace all usages of the original class with the extension class throughout the source code.

### Troubleshooting

If you encounter an error message when introducing a method, check to see that the statements you selected meet the following criteria:

- Selections cannot have more than one output parameter.
- Selections cannot contain a break or continue statement if the corresponding target is not part of the selection.
- Selections cannot contain a return statement that is not the last statement of the selection. The selected code is not allowed to return conditionally.

## 8.11.15 How to Inline a Variable, Method, or Constant

Inline refactoring allows you to replace references to a variable, method, or constant with the values assigned to the variable, the implementation of the method, or the constant, respectively.

### To inline a variable, constant, or method:

1. Select a local variable (temp) method, or constant in the Source Editor.
2. Choose **Refactor > Inline**.
3. Click **Do Refactoring** in the Refactoring window.

Inline Temp replaces all references to a local variable with its initial value and removes the declaration.

Inline Method replaces all method calls with the method's body and removes the declaration.

Inline Constant replaces all constant references with its defined value and removes the declaration.

## 8.11.16 How to Extract a Superclass

Use the refactoring **Extract Superclass** command to extract the common features of classes to create a new superclass. When you extract a superclass, the IDE does the following operations:

- Creates a new class with the selected methods and fields in the selected class. You can also have the new class implement interfaces that are implemented in the selected class.
  - If the selected class extends a class, the new class also extends the same class.
- The selected class is modified so that it extends the new superclass. All selected interfaces are removed from the implements clause.
- Removes the selected public and protected fields from the base class.

Click **Preview** in the Extract Interface dialog box to preview the files that are affected. A list of the files to be modified is displayed in the Refactoring window. If you do not want certain occurrences to be changed, you can clear the checkbox for that occurrence in the Refactoring window. Double-clicking on an occurrence opens that file in the Source editor, and the caret is placed in the line containing the occurrence.

**To extract a superclass:**

1. Open the class containing the methods or fields you want to move to the new superclass.
2. In the Source editor, right-click in the file and choose **Refactor > Extract Superclass**.  
The Extract Superclass dialog box opens.
3. Type the name for your new superclass in the **Superclass Name** text field.
4. Select the members you want to extract to the new superclass.
5. (Optional) If you want to make a method abstract, select the **Make Abstract checkbox** for the method. If you select this checkbox, the method is declared in the superclass as an abstract method and overridden in the current class. The method will be assigned the protected access modifier.
6. Click **Refactor** or **Preview**.

If you click **Refactor**, the IDE applies the changes automatically and skips the remaining steps.

If you click **Preview**, the Refactoring window displays the lines of code that are affected. Review the list and clear the checkbox of any code that you do not want changed. If the class that you are pushing members from has multiple subclasses and you do not want the members to be pushed to all of them, be sure to clear the checkboxes for the corresponding subclasses. Click **Do Refactoring** to apply the selected changes.

Always perform a clean build after completing any refactoring commands. You can do a clean build by right-clicking the project's node in the Projects window and choosing **Clean and Build Project**.

### 8.11.17 How to Extract an Interface

Use the **Extract Interface** command to create a new interface from the selected public non-static methods in a class or interface. Because an interface does not restrict how its methods are implemented, interfaces can be used in classes that have different functions. Creating interfaces can increase the reusability of your code as you can have multiple classes implementing the same interface. If necessary, you can then modify the interface instead of making modifications in multiple classes.

When you extract an interface, the IDE does the following things:

- Creates a new interface with the selected methods in the same package as the current class or interface.
- Updates the implements or extends clause of the current class or interface to include the new interface. Any interfaces that the new interface extends are excluded.

**To extract an interface:**

1. Open the class or interface containing the methods you want to move to an interface.
2. In the Source editor, right-click in the file and choose **Refactor > Extract Interface**.  
The Extract Interface dialog box opens.
3. Type the name for your interface in the **Interface Name** text field.

4. In the **Members to Extract** list, select the members that you want to extract to the new interface.

If the class from which you are extracting an interface already implements an interface, there is also an item for that implemented interface. If you select the checkbox for that interface, the implements clause for that new interface is moved to the new interface that you are extracting.

5. Click **Refactor** or **Preview**.

If you click **Refactor**, the IDE applies the changes automatically and skips the remaining steps.

If you click **Preview**, the Refactoring window displays the lines of code that are affected. Review the list and clear the checkbox of any code that you do not want changed. If the class that you are pushing members from has multiple subclasses and you do not want the members to be pushed to all of them, be sure to clear the checkboxes for the corresponding subclasses. Click **Do Refactoring** to apply the selected changes.

Always perform a clean build after completing any refactoring commands. You can do a clean build by right-clicking the project's node in the Projects window and choosing **Clean and Build Project**.

### 8.11.18 How to Use a Supertype Where Possible

Use the **Use Supertype Where Possible** command to replace references to a type with references to one of the type's supertypes. Before replacing those references, the IDE checks to make sure that supertype and its members are accessible to all of the code that would be changed to reference them.

#### To initiate the **Use Supertype Where Possible** operation:

1. In the Source Editor or Projects window, select the type to which you want to apply the operation. The type can be a class, interface, or enumeration.
2. Choose **Refactor > Use Supertype Where Possible**.
3. In the Select Supertype to Use list, select the supertype that you want to be referenced in place of the type your code currently uses.
4. Click **Refactor** or **Preview**.

If you click **Refactor**, the IDE applies the changes automatically and skips the remaining steps.

If you click **Preview**, the Refactoring window displays the lines of code that are affected. Review the list and clear the checkbox of any code that you do not want changed. If the class that you are pushing members from has multiple subclasses and you do not want the members to be pushed to all of them, be sure to clear the checkboxes for the corresponding subclasses. Click **Do Refactoring** to apply the selected changes.

Always perform a clean build after completing any refactoring commands. You can do a clean build by right-clicking the project's node in the Projects window and choosing **Clean and Build Project**.

### 8.11.19 How to Convert an Anonymous Inner Class to a Regular Inner Class

Use the **Convert Anonymous Class to Inner** command to convert an anonymous class to an inner class that contains a name and constructor. When you use this operation, a

new inner class is created and the anonymous inner class is replaced with a call to the new inner class.

**To use the Convert Anonymous Class to Inner operation:**

1. In the Source Editor, place the insertion point in the anonymous inner class that you want to convert.
2. Press Alt+Enter and choose **Convert Anonymous Class to Inner** from the menu that appears.

### 8.11.20 How to Safely Delete Java Code

Use the **Safely Delete** command to have the IDE check for references to a code element before you delete that code element. You can use this command on classes, methods, and fields.

When you apply the **Safely Delete** command to a code element, the Safe Delete dialog box opens and helps walk you through the process.

**To initiate the Safely Delete operation:**

1. Select the code element that you want to delete and choose **Refactor > Safely Delete**.
2. In the Safe Delete dialog box, make sure that the IDE has identified the right element to be deleted and click **Next**.
  - If the code element is not referenced by other code, the Safe Delete dialog box closes and the code element is deleted.
  - If the code element to be deleted is referenced by other code, a warning and a **Show Usages** button appear in the Safely Delete dialog box. See the following section for information on completing or canceling the operation.

#### 8.11.20.1 Handling Deletions When The Code Element is Referenced

When the message beginning with References to selected elements were found appears in the Safely Delete window, you can proceed in one of the following ways:

- Remove the references to the code to be deleted and then continue the Safely Delete operation.
- Click **Cancel** to cancel the command.

If you mistakenly delete a code element that is still referenced by other code, you can reverse the deletion with the Refactor > Undo command.

**To remove references to code and continue with the deletion of the class element:**

1. In the Safely Delete dialog box, click **Show Usages**.

A list of the references to the code that you want to delete is displayed in the Usages window.

2. Double-click a node for code that references the class to be deleted.  
The referencing class opens in the Source Editor.
3. Remove the reference to the code that you want to delete.

Use the **Safely Delete** command to remove this reference. If there are references to that code as well, you can click **Show Usages** to open a new tab in the Usages window.

4. Repeat steps 2 and 3 until all references to the code that you want to delete are removed.
5. In the Usages window, click **Rerun Safe Delete**.

The Safely Delete command is run again. If there are any references that you have not removed, a warning appears and you can click **Show Usages** to resume the process of resolving the references.

6. Click **Preview** in the Safe Delete window, then click **Next**.
7. Click **Do Refactoring** in the Refactoring window to proceed with the deletion.

### 8.11.21 Using Hints in Source Code Analysis and Refactoring

In the Source Editor, a hint is displayed when the IDE performs source code analysis and detects common syntax errors or problems.

An editor hint is available if a light bulb icon appears in the left margin of the Source Editor while you are developing your code. Click the light bulb icon or press Alt+Enter to read the hint. Click the hint or press Enter to generate the code suggested by the hint.

---

**Note:** You can select which hints to display when you type in the Source Editor in the **Hints** tab of the Editor panel in the Options dialog box. You can access it by choosing **Tools > Options** from the main IDE's menu, and then clicking the Editor category.

---

You can use hints when running source code inspections on a selected file, package, or project and refactoring your sources.

#### To initiate the **Inspect** operation:

1. Choose **Source > Inspect** from the main IDE's menu.
2. In the Scope drop-down list of the Inspect dialog box, select a file, package, or project(s) to be inspected.
3. Select either of the following:
  - In the Configuration drop-down list, select **NetBeans Java Hints** to choose all hints available in the IDE to be used in the source code analysis.  
Alternatively, click **Manage** to open the Configurations dialog box and specify a set of hints to be used in the source code analysis. Click **OK**.
  - Select **Single Inspection** and choose a single hint to be used in the source code analysis.
4. Click **Inspect** to perform the source code analysis.

After the Inspect operation is completed, the hints that can be applied to your code are displayed in the Inspector Window below the Source Editor.

#### To initiate the **Inspect and Transform** operation:

1. Choose **Refactor > Inspect and Transform** from the main IDE's menu.

2. In the Inspect drop-down list of the Inspect and Transform dialog box, select a file, package, or projects to be inspected.

Alternatively, click the button to the right to open the Custom Scope dialog box and specify the custom code to be inspected.

3. Select either of the following to use:

- In the Configuration drop-down list, select a predefined configuration of hints available in the IDE (for example, Convert to JDK 7) to be used in the source code analysis.

Alternatively, click **Manage** to open the Manage Inspections dialog box and group a set of hints into a configuration to be used in the source code analysis. Click **OK**.

- Select **Single Inspection** and choose a single hint to be used in the source code analysis.

4. Click **Refactor** to run the source code analysis.

After the Inspect and Transform operation is completed, the results are displayed in the Inspect and Transform dialog box. Click **Preview** to display the source code refactorings proposed by the IDE upon conducting the source code analysis with the hints you specified.

Check the complete list of hints available in the IDE at the *Java Hints* wiki page at [http://wiki.netbeans.org/Java\\_Hints](http://wiki.netbeans.org/Java_Hints).

For information on how to create a NetBeans module that provides one or more Java hints, see the *NetBeans Java Hint Module Tutorial* at <https://platform.netbeans.org/tutorials/nbm-java-hint.html>.

### 8.11.22 How to Refactor an Enterprise Bean

Refactoring is especially useful in EJB modules, since changing the name of one method often means you have to update the name in all of the related interfaces, deployment descriptors, and dependent classes and servlets.

#### To refactor an EJB module:

- Right-click a piece of code in the Source Editor and choose from the Refactor submenu in the pop-up menu.

You cannot move an enterprise bean or any of its classes and interfaces to a different project. The **Move Class** command only lets you move bean classes and interfaces to different packages in the same EJB module.

You cannot rename mandatory EJB infrastructure methods such as ejbCreate.

When you rename a Web service endpoint interface, the IDE updates the corresponding WSDL file.

When deleting a code element, you can use the **Safely Delete** refactoring command to help check for references to that element before making the changes.

## 8.12 Working with Maven in the IDE

When using Maven as the build infrastructure for Java projects, Maven uses conventions and patterns to provide a uniform build system. All Maven projects use a shared set of plugins that are retrieved from the Maven repository and Maven executes a defined series of tasks as part of the lifecycle when building the project.

Unlike Ant, you do not need to explicitly specify all the tasks required for building a project.

Maven support in the IDE includes the following features:

- Maven is bundled with IDE and is used to run builds
- Project creation from archetypes in the New Project wizard
- Repository browser for managing Maven repositories
- Code-completion for POM in editor
- Configuring of custom Maven goals

For information about developing projects in the IDE using Maven, see the Maven best practices page at <http://wiki.netbeans.org/MavenBestPractices>.

For more information about using Maven, see the Maven documentation at <http://maven.apache.org>.

### 8.12.1 How to Create a New Maven Project

The New Project wizard enables you to create a new Maven project (referred to as an *artifact*) based on an archetype in your local or a remote repository. An *archetype* is a Maven project template. When you create the new artifact, the wizard prompts you to specify the project coordinates (artifactId, groupId, versionId).

#### To create a new Maven project:

1. Choose **File > New Project** from the main menu to open the New Project wizard.
2. Choose the Maven category and then choose a Maven project template.
3. Click **Next**.
4. Specify the project name, location and project details. Click **Finish**.

The IDE creates the Maven project and displays the node for the project in the Projects window.

When using Maven, be aware of the following considerations:

- If you have only recently installed Maven, the first time you create a project the IDE might need to download the most recent artifacts into your local repository. This process can take some time.
- If you choose an Enterprise Application template, the IDE automatically creates an assembly module for packaging the web application module and EJB module. You can select the modules that you want to create in the New Project wizard.
- If you choose a NetBeans Application template, you can choose to use OSGI bundles if the version of NetBeans you are using supports it and whether you want the IDE to create a NetBeans module and configure it as dependency of the project.
- If you choose the Project from Archetype project template, you need to choose an archetype. The wizard displays the available archetypes contained in your local or registered remote repository.
- If you have an existing Maven project that contains a pom.xml file, you do not need to use the New Project wizard. You can open the project by choosing Open Project (Shift+Ctrl+O) from the main menu.

After you create the project, you can configure additional project properties by right-clicking the project node in the Projects window and choosing **Properties**.

## 8.12.2 How to Configure Maven Settings

Configure Maven settings at the IDE level and at the project level. At the IDE level, you can configure Maven installation and repository details and define global Maven goals. At the project level, you can create configurations to activate Maven profiles and bind IDE actions to Maven goals for a project.

### 8.12.2.1 Configuring Maven Settings for the IDE

Configure Maven settings that affect the behavior of Maven in the IDE in the Options window. You can specify a local Maven installation, the location of the local Maven repository, and settings for updating the repository.

#### To modify Maven settings in the IDE:

1. From the main window, choose **Tools > Options**.
2. Click the Java option and then click the **Maven** tab.
3. Modify the properties as desired.
4. Click **OK**.

### 8.12.2.2 Configuring Maven Settings for Projects

By using the project's Properties window, you can configure Maven settings for an individual project. You can create custom configurations and then bind IDE actions to Maven goals and assign the binding to a configuration.

#### To configure Maven settings for a project:

1. Right-click the project node in the Projects window and choose **Properties**.
2. Click the category in the left pane and modify the properties in the right pane.

### 8.12.2.3 Using Project Configurations

The Configurations category of the Project Properties window of a Maven project enables you to create and select custom configurations for your project. For example, you can create custom configurations to do the following:

- Load the project in the IDE with a customized set of dependencies or submodules.
- Trigger actions and activate any associated profiles.
- Customize IDE actions to map to Maven goals.

#### To create a custom configuration for a project:

1. Open the Configurations category of the Properties window by right-clicking the project node in the Projects window and choosing **Set Configuration > Customize**.

Alternatively, right-click the project node and choose **Properties** and select **Configurations** in the Project Properties window.

2. Click **Add** to open the Add Configuration dialog box.

Alternatively, select an existing configuration and click **Clone** to create a copy of the configuration. The name of the clone configuration is the name of the original

configuration with the suffix `_clone`. You cannot rename the clone but you can set profiles and properties.

3. Type a name for the new configuration.
4. Select **Keep private to this IDE** instance if you do not want to share the configuration with other instances of the IDE.  
By default this option is deselected and configurations are shared between instances of the IDE when the project is opened, for example when you open the project in an updated version of the IDE.
5. Type any profiles or settings that you want to enable for the configuration. Click **OK** to close the Add Configuration dialog box.
6. Select the new configuration in the list of configurations and click **Activate**.  
The name of the configuration that is currently active is in bold.
7. Modify additional options for the configuration in the other categories in the Properties window.
8. Click **OK**.

When you click **OK** the IDE creates a configuration file in the root directory of the project with the name `nbactions-myconfigurationname.xml`.

To activate a configuration for a project, right-click the project node and choose **Set Configuration** and then choose the configuration name in the context menu. You can also use the drop-down list in the toolbar to switch between configurations for the project that is currently selected in the Projects window. If a project is set as the main project, the dropdown list displays the configurations for the main project.

#### 8.12.2.4 Binding IDE Actions to Maven Goals

Use the Actions category in the Properties window to customize IDE actions by binding the action to Maven goals. The Actions pane lists the IDE's default project actions that can be mapped to Maven goals. Assign the mapping to a project configuration if you activated configurations for the project.

After you select an IDE action, modify the Maven goals and properties of the action by typing in the textfield. Actions are displayed in bold if the default values for the action have been modified. The following textfields display the goals and properties associated with the selected action:

- **Execute Goals.** This field displays the Maven goals that are associated with the selected IDE action.
- **Activate Profiles.** This field displays any profiles that are activated when the action is invoked.
- **Set Properties.** This field displays the properties that are set on the command line. Properties are generally used to customize the behavior of the executed goal.

Type in the text area to add a property or click the **Add** button to view and add one of the default properties.

When you select an action you can select the **Build with Dependencies** checkbox if you want to run the **Build with Dependencies** command as part of the action.

#### 8.12.2.5 Creating Custom Actions

Click **Add Custom** in the Actions category of the Properties window to create custom actions for a project. After you create the new action, specify the goals or properties for

the new action. Right-click the project node in the Projects window and choose the action under Custom in the context menu to invoke the new action.

Click **Edit Global Custom Goal Definitions** in the **Maven** tab in the Options window to create global custom goals for Maven projects. The global goal is then listed in the context menu under Custom when you right-click a Maven project in the Projects window.

### 8.12.3 How to Work with the Maven POM

The Project Object Model (POM) is the basic unit of work in Maven. The POM is described in the `pom.xml` XML file and contains project and configuration details, such as the artifact coordinates, project dependencies, and build profiles that Maven uses to execute goals and build the project.

For more information about POM elements, see [Introduction to POM at `http://maven.apache.org/guides/introduction/introduction-to-the-pom.html`](#).

When you use the New Project wizard to create an artifact, the IDE automatically generates the `pom.xml` file and adds the artifact details. Open `pom.xml` in the XML editor to edit the file. The following features are available in the XML editor to help you edit `pom.xml`:

- **Code completion.** The IDE's code completion can help you add and modify elements. The code completion hints offered in the editor are based on the contents of the repository.
- **Insert Code.** The IDE can generate XML elements for you based on the details you specify in a dialog box. When you right-click in the editor window and choose **Insert Code** (Ctrl-+), a context menu appears where you can choose to add any of the following types of elements to the POM:
  - Dependency
  - Dependency Exclusion
  - Plugin
  - Profile
  - License

After you select the type of element a dialog box opens that enables you to specify the details about the code that you want the IDE to insert into the POM:

- **Dependency Graph.** The Maven Dependency Graph is a visual representation of the direct and transitive dependencies specified in `pom.xml`. Click the **Graph** tab in the XML editor toolbar to open the graph. You can select any element in the graph to view a representation of the dependencies of that element. You can right-click in the graph view and choose different layout schemes in the popup menu.
- **Effective POM.** The **Effective POM** tab displays a read-only version of the POM that Maven uses to build the project after other factors that affect the build such as any active profiles and inherited parent POMs are incorporated. The left column of the tab displays the source that generates each line in the effective POM when the source can be determined. Right-click the source and choose **Go to Source** to navigate to the line in the source where the element is specified.

In the Effective POM tab you can click Show Diff in the toolbar to compare the effective POM to a POM for an alternate project configuration or profile. Click Show Diff to open a dialog box where you can select the alternate POM. The diff of the POMs opens in a new tab in the IDE.

## 8.12.4 How to Manage Maven Project Dependencies

To make libraries and projects available to your project during compilation, testing, and execution, you need to declare the projects or libraries as dependencies. All dependencies need to be available as artifacts in your local repository when you build the project.

You declare a dependency by modifying the POM to add the coordinates of the artifact you want to declare as a dependency. All Maven artifacts are defined by a unique coordinate that consists of a group identifier (`groupId`), artifact identifier (`artifactId`) and version. Adding a dependency to the POM is similar to adding libraries to the classpath in an Ant-based project.

### To add a dependency to a Maven project:

- **Edit the POM in the editor.** Open `pom.xml` in the XML editor and use code completion to help you write the dependency elements and artifact coordinates.
- **Use the Add Dependency dialog.** Type or search for the artifact in the Add Library dialog and click **OK**. When you click **OK**, the IDE adds the dependency elements and artifact coordinates to `pom.xml`.

### To open the Add Dependency dialog:

- Right-click the Dependencies node in the Projects window and choose **Add Dependency**.
- Right-click in `pom.xml` in the editor and choose **Insert Code** (Ctrl+I) and choose **Dependency**.
- Right-click an artifact under the Maven Repositories node in the Services window and choose **Add as Dependency** and select a project in the Add Dependency dialog box.

After you add a dependency to `pom.xml`, a node for the artifact  appears under the project's Dependencies node. Artifacts that are not in your local repository are marked with a badge. You need to have a copy of the artifact in your local repository if you want to use it in your project. This applies to libraries and to other projects that are described as dependencies.

---

**Note:** If a required artifact is available in a remote repository, Maven automatically downloads the artifact and any transitive dependencies from the repository when you build the project. Maven uses the latest available version of the artifact if no version is specified. If the required artifact is not available in a remote repository, right-click the artifact node and install the artifact manually.

---

It is recommended that you store Maven software library JARs and project artifacts in a local repository and keep the source code for projects under a version control system.

## 8.12.5 How to Work with Maven Artifacts

The project dependencies that are displayed under a project's Dependencies node are Maven *artifacts*. Each Maven artifact has a unique coordinate (`groupId`, `artifactId` and `version`) that is used to identify the artifact in a project's POM. An artifact can also have dependencies. The artifacts available to a project can be viewed under the Maven Repositories node in the Services window.

Right-click an artifact in the Projects window or the Services window to invoke a popup menu with commands for downloading the sources and Javadoc of an artifact, finding usages and viewing artifact details. Use the Artifact Viewer to view details about individual artifacts.

**To open the Artifact Viewer, perform either of the following steps.**

- In the Projects window, right-click an artifact under the Dependencies node and choose **View Artifact Details**.
- In the Services window, expand the Maven Repositories node, right-click an artifact and choose **View Details**.

The Artifact Viewer displays details in the following tabs:

- **Basic.** Displays the artifacts coordinates and version details.
- **Project.** Displays metadata contained in the artifact's POM such as the description of the artifact, the URL for the project and issue tracking and version control details.
- **Classpath.** Displays the artifact's compilation, runtime and test dependencies.
- **Graph.** Displays a visualization of the artifact's primary and secondary dependencies.
- **POM.** Displays the pom.xml file of the artifact.

[Table 8–1](#) describes the commands available from the toolbar of the Artifact Viewer:

**Table 8–1   Artifact Viewer Toolbar Commands**

Command	Description
Add as Dependency 	Opens the Add Dependency dialog box from which you can choose a project.
Checkout Sources 	Opens the Checkout Sources wizard in which you can specify a location for a local repository for the sources. Use this command only if the sources of the artifact are available in a repository
Create Library 	Creates a NetBeans library from the artifact. The new library is available in the Ant Library Manager.

#### 8.12.5.1 Visualizing Dependencies

The **Graph** tab of the Artifact Viewer enables you to easily visualize dependencies. The visualizer also provides tools to help you find and resolve potential problems resulting from dependency version conflicts.

**To use the graph view to resolve dependency conflicts:**

1. Open the artifact in the Artifact Viewer and click the **Graph** tab.  
Alternatively, you can open pom.xml file in the editor and click the **Graph** tab.
2. Locate any artifact boxes that have a red background.  
In the dependency graph, a dependency with a red background indicates a potential version conflict.
3. Click the suggestion icon to open a dialog box with a description of the conflict and options for resolving the conflict.

## 8.12.6 How to Build a Maven Project

The Maven build lifecycle has defined phases with goals that are executed when building and distributing the project. When you execute a goal, Maven will also execute all the preceding goals in the build lifecycle. You can invoke phase goals and plugin goals by mapping the goals to IDE actions, or you can create custom goals and invoke the goals individually. When you execute a goal, the Maven output with the plugin and goal identifiers are displayed in the Output window.

When you want to generate distributable files for the project, such as JAR files or WAR files, you need to build the project. When you build a Maven project using the IDE's Build command, by default the IDE executes the plugin goals in the install phase of the Maven lifecycle (builds the project and adds the artifact to the local repository).

---

**Note:** The IDE can automatically compile classes when you save them if you enable the **Compile on Save** option in the Compile category of the project's Properties window. When **Compile on Save** is enabled, you do not need to build the project or compile individual classes to run or test the project in the IDE. The incrementally compiled files are stored in a cache in your user directory and are copied to your project's build folder when you run or debug your project.

---

**Compile on Save** is enabled by default for WAR, EJB and EAR projects.

### To build a Maven project:

1. Select the project node in the Projects window.
2. Choose **Run > Build Project (F11)**.

When you build a project the IDE displays the Maven output and any compilation errors in the Output window. You can double-click any error to go to the location in the source code where the error occurred.

You can click **Show Build Overview** () in the Output window sidebar to open the Build Execution Overview window to view additional details about the executed goal or goals. Click Show Phase in the Build Execution Overview window to view the goals organized by the phase in the build lifecycle. You can right-click a goal in the Build Execution Overview window to open a popup menu where you can choose to view the goal definition in the POM, the source code of the plugin or to debug the plugin.

You can modify the goals that are executed by the Build command and other IDE commands by modifying the Maven settings in the project's Properties window.

### To build an individual project and its required projects:

- Right-click the project's node in the Projects window and choose **Build with Dependencies**.

---

**Note:** You can change the default Build action to **Build with Dependencies** by selecting the Build action in the Actions category of the project's Properties window and then selecting the **Build with Dependencies** checkbox.

---

Modify the project dependencies by adding libraries in the Projects window or by editing pom.xml in the editor.

### 8.12.6.1 Executing Maven Goals in the IDE

IDE actions in the popup menu are mapped to Maven phases and goals. For example, by default the Build action in the IDE is mapped to the install phase in the Maven lifecycle. When you right-click the project node and choose **Build**, the IDE executes the goals described in the install phase to package the project and executes all the goals in the preceding phases in the build lifecycle.

You can modify the mapped Maven goals that are invoked by IDE actions by configuring actions in the project's Properties window or in the **Maven** tab in the Options window. For more information, see [Section 8.12.2, "How to Configure Maven Settings."](#)

### 8.12.6.2 Customizing the Build Process

To customize the build for a Maven project, modify the POM to add or reconfigure plugins and dependencies. You can use the code completion in the editor to help you when adding details to `pom.xml`.

#### To customize a build:

1. Open `pom.xml` in the XML editor.
2. Choose **Source > Insert Code** (**Ctrl+I**) from the main menu and select **Plugin** from the popup menu to open the Add New Plugin dialog box.
3. Type a query term to search for the plugin.
4. Select the plugin from the list of available plugins.
5. Select the goals that you want to be executed. Click **OK**.

When you click **OK**, the IDE adds the plugin description to `pom.xml` and the goals to run as part of the build process. If you add a plugin that is not in the local repository, the required artifact will be automatically downloaded from a remote repository when required.

### 8.12.6.3 Executing Individual Goals

If you select the project node or `pom.xml` in the Projects window you can execute goals that are defined in the `pom.xml` from the Navigator window.

#### To execute a goal:

1. Open the Navigator window.
2. Select the project node in the Projects window.

Alternatively, select `pom.xml` in the Projects window and choose **Related goals** in the drop-down list at the top of the Navigator window.

3. Right-click a goal in the Navigator window and choose **Execute Goal**.

Alternatively, select **Execute Goal** with **Modifiers** to open the Run Maven dialog box and modify the default run settings for the goal.

You can view the results of the goal in the Output window.

## 8.13 Working with Maven Repositories

Software library archives (JARs), build artifacts and dependencies that are used to build Maven projects are stored in repositories. There are two types of repositories:

- **Local repositories.** A cache of a remote repository that is stored on the local machine. Maven projects are built against the local repository. The local repository usually only stores a subset of the files available in the remote repository and any temporary build artifacts.
- **Remote repositories.** A repository that contains all the Maven artifacts and plugins. The remote repository may be a third-party repository (for example, <http://repo.maven.apache.org/>), or it may be a private internal repository.

### 8.13.1 How to Work with Maven Repositories

The IDE indexes the contents of local and remote Maven repositories using the Nexus indexing engine. The IDE uses the repository indexes for some Maven-related functions such as code completion. You can browse and manage Maven repository indexes in the Services window.

#### To browse Maven repositories:

- Choose **Window > Services** from the main menu and expand the Maven Repositories node to view the contents of the repositories.

The Maven Repositories node in the Services window lists the Maven repositories registered with the IDE and enables you to add, remove and update Maven repositories. The local Maven repository is indicated by a yellow repository node ( ) and remote repositories are indicated by blue repository nodes ( ).

By default, the IDE includes the central Maven repository in the list of remote repositories. When a project requires build artifacts that are not stored in the local repository, Maven downloads the required artifacts to the local repository from a remote repository. The files in the local Maven repository are then shared by all of your Maven projects.

Expand the repository nodes to view the indexed artifacts grouped by GroupId and ArtifactId, the version of the sources, and the type of packaging. Depending on the metadata available, right-click an artifact to perform the following actions:

- **View Details.** Opens the **Artifact Viewer** tab in the editor window.
- **Add As Dependency.** Adds the library to an open project as a dependency. The IDE automatically modifies the project's POM.
- **Find Usages.** Displays open projects and repository artifacts that use the selected library as a dependency.
- **View JavaDoc.** Displays the downloaded JavaDoc in a browser.
- **Open.** Opens the project POM in the IDE.
- **Download.** Downloads the artifact to the local Maven repository.
- **Download Sources.** Downloads the source JAR for the binary. After downloading, if you click on a class file, the Java file is displayed.
- **Download JavaDoc.** Downloads the Javadoc.
- **Copy.** Copies the *dependency.xml* snippet of the artifact to the clipboard, which you can paste into the project *pom.xml* file.

#### To locate artifacts in the repositories:

1. Right-click the Maven Repositories node and choose **Find**.
2. Enter the search term for the artifact in the Find in Repositories dialog box (a groupID, for example) and select any additional criteria.

**3. Click OK.**

When you click **OK** the IDE creates a node for the search term ( ) under the Maven Repositories node. Expand the search node to view a list of the artifacts that matched your search. Remove a search node by right-clicking the node and choosing **Delete**.

**To add a repository:**

1. Right-click the Maven Repositories node and choose **Add Repository**.
2. Specify the details for the new repository and click **Add**.

**To update the index of a repository:**

- Expand the Maven Repositories node, right-click the repository node you want to update and choose **Update Index**.

---

# Testing and Profiling Java Application Projects

This chapter describes how to use the tools provided by the IDE to run unit tests and to profile Java applications.

This chapter contains the following sections:

- [About Testing and Profiling Java Application Projects](#)
- [Testing Java Application Projects with Unit Tests](#)
- [Creating a Unit Test](#)
- [Running a Unit Test](#)
- [Debugging a Unit Test](#)
- [Configuring Unit Test Settings](#)
- [Creating a Selenium Test](#)
- [Configuring Selenium Server Settings](#)
- [Starting a Profiling Session](#)
- [Selecting a Profiling Task](#)
- [Attaching the Profiler](#)
- [Attaching the Profiler to a Remote Application](#)
- [Profiling a Free-form Project](#)
- [Taking and Accessing Snapshots of Profiling Data](#)
- [Taking a Heap Dump](#)
- [Setting a Profiling Point](#)
- [Profiling Telemetry](#)
- [Profiling Methods](#)
- [Profiling Objects](#)
- [Profiling Threads](#)
- [Profiling Locks](#)
- [SQL Queries Profiling](#)
- [Additional Functions when Running a Profiling Session](#)

## 9.1 About Testing and Profiling Java Application Projects

The IDE provides tools for creating and running unit tests and for profiling Java applications. Unit tests enable you to test the code in Java applications. Profiling is the process of examining an application to locate memory or performance-related issues.

When profiling a Java application, you can monitor the Java Virtual Machine (JVM) and obtain data about application performance, including method timing, object allocation and garbage collection. You can use this data to locate potential areas in your code that can be optimized to improve performance.

You can use the IDE to test and to profile the following types of Java applications:

- Java SE projects
- Java EE and Web applications
- Java Free-form projects
- NetBeans Modules and Module suites

The following profiling capabilities are available:

- **Telemetry**—profiles CPU, memory usage, number of threads and loaded classes. See [Section 9.17, "Profiling Telemetry"](#)
- **Methods**—profiles methods execution times and invocation count, including call trees. See [Section 9.18, "Profiling Methods."](#)
- **Objects**—profiles size and count of allocated objects including allocation paths. See [Section 9.19, "Profiling Objects."](#)
- **Threads**—profiles threads time and state. See [Section 9.20, "Profiling Threads."](#)
- **Locks**—profiles locks content data. See [Section 9.21, "Profiling Locks."](#)

## 9.2 Testing Java Application Projects with Unit Tests

The IDE provides built-in support for generating and executing unit tests based on the JUnit, TestNG, Mocha frameworks and Selenium tool suite.

The IDE supports JUnit 3 and JUnit 4 unit testing. For more information about JUnit, see <http://www.junit.org>

For more information about TestNG, see <http://testng.org/doc/index.html>

For more information about Mocha, see <http://mochajs.org/>

The IDE supports Selenium 2.0 testing. For more information about Selenium, see <http://www.seleniumhq.org/docs/index.jsp>

### 9.2.1 Test Types in the IDE

You can use the IDE to create the following:

- **Empty Tests.** Test skeletons without testing methods, for which no class has been designated to be tested.
- **Tests for Existing Classes.** Classes containing the actual testing methods which mirror the structure of the sources being tested.
- **Test Suites.** Groups of test classes clustered to permit testing of an entire application or project.

You can generate tests and test suites by selecting any class or package node in the Projects window and choosing **Tools > Create/Update Tests** from the main menu.

## 9.2.2 Unit Test Structure

The IDE represents unit tests as subtrees which mirror the project's Java package structure. By default, when the IDE generates tests each test class has the name of the class it is testing appended by the word `Test` (for example, `MyClassTest.java`).

Each standard project has a default test folder that is used to store the unit tests and test suites. This folder is displayed as the `Test Packages` node in the Projects window. You can add any number of test folders to your project. Test files and the source files they test cannot be located in the same source tree.

## 9.3 Creating a Unit Test

The unit test generator enables you to create unit test suites and compilable test classes for use as skeletons in your unit tests. You can create unit tests for single classes and entire packages, as well as empty test skeletons to be used with sources you create later.

JUnit 4.x is backwards-compatible with JUnit 3.x. If you decide to downgrade from JUnit 4.x to JUnit 3.x, tests using unsupported features such as annotation descriptions and static imports are ignored.

---

**Note:** The JUnit 4.x option is only available if your project is running on the Java 5.0 platform or higher.

---

### 9.3.1 Changing the JUnit Version

You can update support for the JUnit versions by updating the JUnit libraries in the project's `Test Libraries` node. View the project's JUnit libraries by expanding the project's `Test Libraries` node in the Projects window.

To add a new library, right-click the `Test Libraries` node in the Projects window and choose **Add Library** and select the library in the Add Library dialog box. To remove a library, right-click the library and choose **Remove**.

### 9.3.2 How to Create a Unit Test

Generated tests are distinguished by appending `Test` to the tested classes' names (for example, `MyClassTest.java`).

---

**Note:** Depending on the version of the unit testing framework you are using, the name of the test class is not required to end with `Test`.

---

To create a test, the project must have a test directory for the tests. The IDE creates a `Test Packages` directory by default when you create a project. If no test directory exists in your project or you want to specify a different directory for your test, you can specify a test directory in the project properties dialog.

#### To create a test for a single class:

1. Right-click the class in the Projects window and choose **Tools > Create/Update Tests** (`Ctrl+Shift+U`) to open the Create Tests dialog box.

You can modify the default name of the test class, but some of the navigation in the IDE between classes and test classes might not function if the name of the test class does not contain the name of the class that is tested.

2. Select a unit test framework.
3. Select the desired code generation options and click **OK**.

Alternatively, you can create a test by choosing **File > New File**, selecting the Unit Tests category, and selecting JUnit Test or TestNG Test Case in the File Types pane.

**To create a test suite:**

1. Right-click the package node containing the source files for which you want to generate the test suite in the Projects window and choose **Tools > Create Tests** (Ctrl+Shift+U) from the context menu.
2. Select a unit test framework.
3. Select the **Generate Test Suites** checkbox.
4. Select the desired code generation options and click **OK**.

The IDE generates test classes for all enclosed classes and stores them in the project's Test Packages node. If any of the test classes already exist, those classes are updated. By default the IDE includes all the test classes as part of the generated test suite.

Alternatively, you can create a test suite by choosing **File > New File**, selecting the Unit Test category, and selecting Test Suite or TestNG Test Suite in the File Types pane.

**To create an empty test:**

1. Choose **File > New File** from the main menu.
2. In the New File wizard, select the Unit Test category and JUnit Test or TestNG Test Case in the File Types pane. Click **Next**.
3. Specify the test class name, folder and package.
4. Select the desired code generation options and click **Finish**.

When you create an empty test the IDE generates a test class that does not contain any skeleton methods. The IDE creates the test suite class based on the parameters you have specified and opens the class in the editor. The test suite class then appears in the Projects window under the *package-name* node in the test folder.

**To create a new test for an existing class:**

1. Choose **File > New File** from the main menu.
2. In the New File wizard, select the Unit Test category and Test for Existing Class in the File Types pane. Click **Next**.
3. Specify the test class name and folder.
4. Select the desired code generation options and click **Finish**.

When you create a test for an existing class, you specify the class that you want to test and the IDE generates a test class that contains skeleton methods based on the methods in the specified class. The IDE opens the class in the editor. The test class then appears in the Projects window in the test folder. The IDE creates the test class under the *package-name* node that replicates the package structure of the tested class.

**To specify a test directory:**

1. Right-click the project node in the Projects window and choose **Properties**.

2. In the Properties window, select Sources in the Categories pane.
3. Define the properties of the test packages folder in the Test Package Folders list.

You can add or remove the folders that are used for test packages and modify the names of the test packages folder as it appears in the Projects window.

4. Click Close.

Your project must have a test package folder to generate unit tests. If the test packages folder for your project is missing or unavailable, create a new folder in your project and then designate the new folder as the test packages folder in the project's Properties window.

## 9.4 Running a Unit Test

Once you have created a test or test suite, use the **Run Test** command to initiate execution of the test. **Run Test** commands are available on source nodes only. After you run a test, you can rerun individual test methods executed during the test and displayed in the Test Results window.

### 9.4.1 How to Run a Unit Test

You can run unit tests for a specific class or method or for a project.

#### To run tests for an entire project:

1. Select any node or file in the project you want to test in the Projects or Files window.
2. From the main menu, choose **Run > Test Project *project\_name*** (Alt+F6).

The IDE executes all of the project's tests.

If you want to run a subset of the project's tests or run the tests in a specific order, you can create test suites that specify the tests to run as part of that suite. After creating a test suite you run the suite in the same way you run a single test class.

You can select multiple projects in the Projects window and run tests for them at once by choosing **Run > Test *number\_of\_selected\_projects* Projects** (Alt+F6) from the main IDE's menu. If you want to run a subset of the project's tests or run the tests in a specific order, you can create test suites that specify the tests to run as part of that suite. After creating a test suite you run the suite in the same way you run a single test class.

#### To run a test for a single class:

1. Select the node of the class for which you want to run a test in the Projects or Files window.
2. From the main menu, choose **Run > Test File** (CtrlF6).

You can also run a class's test by right-clicking the test class node itself in the Projects window and choosing **Test File** (Ctrl+F6).

#### To run a single test method:

1. Run the test class or suite containing the test method.
2. In the Test Results window, right-click the test method and choose **Run Again**.

To run a single test method the method must be listed in the Test Results window.

## 9.4.2 Working with Unit Test Output

When you run a test, the IDE shows the following test results in two panes in the Test Results window:

- A summary of the passed and failed tests and the description of failed tests are displayed in the left pane of the window. You can use the filter icons in the left side of the window to filter the test results.
- The textual output from the unit tests themselves is displayed in the right pane of the window.

The output from the process that builds and runs the test is displayed in the Output window. You can double-click any error to jump to the line in the code where the error occurred.

After you run a test class, you can right-click any test method displayed in the Test Results window and choose **Run Again** from the context menu to run the individual test method again or **Debug** to debug the test.

## 9.5 Debugging a Unit Test

You can debug a test class as you would any class. After you set your breakpoints in the test class you can run your unit test in the debugger and step through the code to locate problems in your test classes.

**To debug a unit test:**

1. Place your breakpoints in your test class.
2. Select the node for the class whose test you wish to debug.
3. Choose **Debug > Debug Test File** (Ctrl+Shift+F6) from the main menu.

When you choose **Debug Test File** the IDE starts the test in the debugger and opens the Debugging window. For information on using the Debugging window, see [Section 10.9, "Using the Debugger Windows."](#)

## 9.6 Configuring Unit Test Settings

The IDE enables you to customize the process of test generation as you create tests. You can also edit the list of sources the IDE references when compiling tests.

### 9.6.1 How to Edit Unit Test Settings

Modify the test settings to create a custom tests.

**To edit unit test settings as you create tests:**

1. Right-click the sources for which you wish to create tests.
2. Choose **Tools > Create Tests** (Ctrl+Shift+U).
3. Select a unit test framework.
4. In the Create Tests dialog box, select the Code Generation options you require for the tests.

The IDE creates the tests with the specified options.

## 9.6.2 How to Edit the Classpath for Compiling or Running Tests

Edit the classpath as needed to include specific projects, libraries, or JAR files.

**To edit the classpath for compiling or running tests:**

1. Right-click the project's Test Libraries node and choose one of the following:
  - **Add Project.** The build output, source files, and Javadoc files of another IDE project.
  - **Add Library.** A collection of binary files, source files, and Javadoc files.
  - **Add JAR/Folder.** A JAR file or folder somewhere on your system.
2. Click **OK**.

The IDE adjusts and stores the classpath priorities based on the new settings. For information on setting the classpath for a project, see [Section 6.2.3.1, "Managing the Classpath."](#)

## 9.7 Creating a Selenium Test

You can create a Selenium test by choosing **File > New File**, selecting the Selenium Tests category, and selecting Selenium Test Case in the File Types pane.

**To create a Selenium Test:**

1. Choose **File > New File** from the main menu.
2. Select the Selenium Tests category.
3. Select Selenium Test Case in the File Types pane.
4. Click **Next**.
5. Provide the required information in the Name and Location panel of the wizard.
6. Click **Finish**.

A Selenium Mocha/Protractor test case with a default test method is created.

**To create a Selenium Test for a single class (applies to Maven java related project types only):**

1. Right-click the class in the Projects window and choose **Tools > Create/Update Tests** to open the Create Tests dialog box.

---

**Note:** You can modify the default name of the test class, but some of navigation between classes and test classes might be disabled in the IDE if the name of the test class does not contain the name of the class that is tested.

---

2. Select Selenium as the test framework.
3. Click **OK**.

## 9.8 Configuring Selenium Server Settings

Configure the Selenium server settings as needed to test.

**To configure Selenium server:**

1. Right-click the Selenium Server node in the Services tab.
2. Provide the required information.
3. Click OK.

## 9.9 Starting a Profiling Session

If you have a project that is targeted to run on your local machine, you can profile the project without any additional configuration. When you profile a local project, you launch the project and start the profiling session from within the IDE.

If you want to profile a local application but you cannot or do not want to start the application from the IDE, you can profile the application by attaching the IDE to the application.

To profile a local project, the project must be open in the IDE. You can start the profiling session after you select the profiling task.

**Table 9–1 Profiling Commands**

Command	Description
<b>Profile &gt; Profile Project</b>	The selected task is run on the project that is selected in the Projects window or on the main project if a main project is set.
<b>Profile &gt; Profile File</b>	The profiling task is run on the selected file. The selected file must have a main method.
<b>Profile &gt; Profile Test File</b>	If a test file is selected the profiling task is run on the test file. If the selected file is not a test file, the profiling task is run on the test file associated with the selected file.
<b>Profile &gt; Attach to Project</b>	Opens the Profile window. In the Profile window, you select the profiling task.
<b>Profile &gt; External Process</b>	Opens the Profile window. In the Profile window, you select the profiling task to profile remote applications or applications that you start outside the IDE.

---

**Note:** Before you can use the profiler in the IDE, you must calibrate the profiler. You only have to calibrate the profiler once. For information, see [Section 9.9.2, "How to Calibrate the Profiler."](#)

---

### 9.9.1 How to Profile a Project

**To profile an individual project:**

1. Right-click a project in the Projects window and choose **Profile**. Alternatively, you can select a project in the Projects window and choose **Profile > Profile Project** from the main menu.
2. On the main window click the **Configure Session** button and select a profiler mode by clicking on it. You can change the profiler mode at any point by clicking the **Profile** drop-down arrow.
3. On the main window, click the **Profile** button. The application and the profiling session are started.

If a project is set as the main project, the **Profile Project** command in the **Profile** menu always profiles the main project.

**To profile a specific class:**

1. In the Projects window, select the class that you want to profile. (This class must contain a runnable method.)
2. Choose **Profile > Profile File** from the main menu.
3. On the main window, click the **Profile** button. The profiling session is started.

For information on profiling an application that you started outside the IDE, see [Section 9.11, "Attaching the Profiler."](#)

For information on monitoring an application thread activity, see [Section 9.20, "Profiling Threads"](#). For information on examining CPU performance and memory usage, see [Section 9.17, "Profiling Telemetry"](#).

## 9.9.2 How to Calibrate the Profiler

You must calibrate the IDE before you can use the IDE to profile an application. You must run the calibration process for each JDK that you use for profiling. You do this because instrumenting the bytecode of the application imposes some overhead, and the time spent in code instrumentation needs to be "factored out" to achieve more accurate results.

You only have to calibrate the IDE once for each JDK that you use. However, you should run the calibration process again when anything changes on your local or remote configuration that could affect system performance. The following could affect system performance:

- Any hardware upgrade
- Any significant change or upgrade of the operating system
- An upgrade of the Java platform used for profiling

**To calibrate the IDE to the local system:**

1. Close any other programs that are running.

The IDE runs the calibration if other applications are running, but running any CPU-intensive programs when performing the calibration might affect the accuracy of profiling results.

2. Go to **Tools > Options > Java > Profiler**.
3. Select the **General** category and click **Manage** in the Profiling specific options. The **Manage Calibration Data** dialog box displays.
4. Select the Java Platform to be used for profiling. Click **Calibrate**.

You can click **Java Platforms** to open the Java Platform Manager to add a new Java platform. The **Manage Calibration Data** dialog box displays the date that the most recent calibration was performed.

When you click **Calibrate**, the IDE collects calibration data on the selected Java platform. When the calibration process is complete you can start using the IDE to profile your applications.

Do not share calibration data between various computers or systems.

You cannot perform a calibration of a remote platform from the Manage Calibration Data dialog box. Calibration of a remote platform is performed the first time that you profile an application on the remote platform.

**To calibrate the IDE to a remote system:**

- To calibrate the IDE to a remote system, see the NetBeans FAQ at <http://wiki.netbeans.org/wiki/view/FaqProfilerCalibration>.

### 9.9.3 Understanding the Toolbar Icons

Table 9–2 contains the various icons used when profiling. These icons are available from areas in the NetBeans user interface, including Profile window with Methods, Profile window with Threads, Profile window with Objects, Profile window with Telemetry, and Profile window with Locks.

**Table 9–2 Profiling Methods Icons**

Icon	Name	Description
	Pause live results	Allows to stop updating live results. Pressed automatically when a context menu is displayed for the profiling results.
	Update live results	Is enabled when the Pause live results button is pressed, allows to manually update paused live results.
	Show delta/absolute values	Controls switching between absolute and incremental values.
	Forward calls	Shows or hides the Forward calls view (visible by default).
	Hot spots	Shows or hides the Hot spots view (hidden by default).
	Reverse calls	Shows or hides the Reverse calls view (visible by default).
	Select threads	Shows a list of threads to be included or excluded from the displayed results.
	Take snapshot	Saves the currently available profiling data to a file which can be reopened later for offline analysis or compared with another snapshot.
	Reset results	Clears the currently available profiling data, enabling to start collecting new data from a defined point.
	Take thread dump	Takes snapshot of all threads in the profiled JVM including their call stacks and displays it in a separate view.
	Take heap dump	Saves a snapshot of all objects currently stored in the heap memory in a .hprof format and optionally opens it in a heap viewer.
	Request garbage collection	Requests garbage collection in the profiled JVM. There's no guarantee when or if at all the garbage collection will be performed. To make sure GC will be run, the action should be invoked several times.

## 9.10 Selecting a Profiling Task

When you start a profiling session, you first select and configure the profiling task. You choose the profiling task according to the type of profiling results you want to obtain from the session.

Each profiling task has default settings that are sufficient in most situations. You can configure the basic settings of each profiling task to modify the profiling results that are returned.

**Table 9–3 Profiling Tasks and Results**

Profiling Task	Results
Telemetry	Choose this to monitor CPU, memory usage, number of threads and loaded classes.
Methods	Choose this to profile methods execution times and invocation count, including call trees.
Objects	Choose this to profile size and count of allocated objects including allocation paths.
Threads	Choose this to profile threads time and state.
Locks	Choose this to profile locks content data.

### 9.10.1 How to Select a Profiling Task

Each time you start or modify a profiling session, you first select a profiling task. You cannot run more than one profiling session at one time. You can profile a project or an individual file if the file has a runnable method. You can also profile test classes.

#### To select a profiling task:

1. Select a project or file in the Projects window and choose **Profile > Profile Project** from the main menu.  
Alternatively, you can right-click a project node in the Projects window and choose **Profile**, or right-click a file and choose **Profile File** or **Profile Test File**.
2. (Required when you run a profiling session for the first time only.) Select a profiling task by clicking one of the tasks in the **Configure Section** drop-down list.
3. Click **Profile**.

### 9.10.2 Using a Load Generator Script

The IDE supports using a load generator to replay previously saved load testing scripts. By using a load testing script, you can simulate a heavy load on your server or application to achieve more accurate results when testing application performance.

The IDE supports the Apache JMeter load testing tool. To work with the Apache JMeter load testing tool in the IDE, use the Plugins manager download and install the JMeter plugin from the Update Center by choosing **Tools > Plugins** from the main menu. This plugin installs additional Load Generator and Profiler/Load Generator Bridge plugins. A sample JMeter script is downloaded as a part of the JMeter plugin. The sample script is available in `userdir/modules/jmeter/extras/Test.jmx`.

Once you install the JMeter integration, a new **Plugins** section with a **Load Generator** item appears in the **Profile** or **Attach** drop-down menu. Choose **Load Generator** to start a JMeter script when starting the profiling session.

**To create a new load testing script:**

1. Choose **File > New File** from the main menu.
2. Select **Load Testing Scripts > JMeter Plans** in the **Categories** list and **New JMeter Plan** in the **New File** dialog and click **Next**.
3. In the **Name and Location** panel of the dialog specify the name of the new test jmx file and its location. Click **Finish**.

For more about using Apache JMeter, see  
<http://jakarta.apache.org/jmeter/index.html>.

## 9.11 Attaching the Profiler

The easiest way to profile your application is to use the **Profile Project** command. However, if you must start your application outside of the IDE, you can use the attach mode to profile an application. For example, you might want to use the attach mode in the following cases:

- Your application needs to be started from the command line or uses a complex launch script
- You want to obtain profiling data on the startup of the application or target JVM
- You want to profile the application without restarting the application

You must use the attach mode in the following cases:

- You want to profile an application running on a remote JVM, such as a remote application server
- You want to profile a Java EE or web application and the target server is not Tomcat, GlassFish, WebLogic or JBoss

The Attach mode enables you to attach the profiling tool to an application that is already running, or just about to start on a local or remote machine. Using the attach mode is useful for profiling long-running applications, and in particular for profiling web or enterprise applications running on top of an application server. You can also use the Attach mode to profile the code of the application server itself.

### 9.11.1 How to Configure the Attach Settings

Before you can attach to an application, the startup options for the target application or server must be configured. You use the **Attach Settings** dialog box to specify the attachment settings for a project. You only have to configure the attachment settings once. The attachment settings are project specific, which means that each project has its own settings. You can open the **Attach Settings** dialog box at any time if you want to review or modify any of the attachment settings.

**To configure the attach settings:**

1. Choose **Profile > Attach to Project** or **Profile > Attach to External Process** () from the main menu to open the **Profile** window.
2. Click the **Attach** button to display the **Attach Settings** dialog box and make the appropriate selections.

Based on the attach mode you choose in the Profile drop-down list, the **Attach Settings** dialog box provides you with a set of instructions on how to configure the application so that you can attach the profiler.

3. Click **OK**.

The attach mode you choose depends on the type of profiling data you want to obtain and the details of the target JVM platform. [Table 9–4](#) provides an overview of the attach modes:

**Table 9–4 Attach Modes**

Attach Mode	Description
Already running local Java process	<p>Use this mode if you want to obtain profiling data on a local application. To use this mode, the application needs to be running on JDK 1.6, 1.7, or 1.8.</p> <p>When you use this mode, you can attach to and detach from the application without restarting application.</p>
Manually started local Java process	<p>Use this mode if you want to obtain profiling data on the startup of a local application.</p> <p>When you use this mode, the target JVM and application wait until you attach the profiler before starting. After you detach from the application, you must restart the application to start another profiling session.</p>
Manually started remote Java process	<p>Use this mode if you want to obtain profiling data on an application running on a remote JVM. To use this mode, you must install the Profiler Remote Pack on the remote target machine. You specify the details of the remote system and generate the Profiler Remote Pack using the Attach Settings dialog box.</p> <p>When you use this mode, the remote application starts after the profiler is attached. This mode enables you to obtain profiling data on the startup of the target JVM.</p>

**To profile an applet using the Local Direct or Remote attach mode:**

- Specify the agent parameters in the Java Control Panel and restart the browser.

### 9.11.2 How to Attach the Profiler to a Local Application

Use the attach mode to profile a local application when you must start the application outside of the IDE. For example, you may want to use the attach mode in the following cases:

- Your application needs to be started from the command line or uses a complex launch script.
- You want to obtain profiling data on the startup of the application or target JVM.
- You want to profile the application without restarting the application (Already running local Java process attach mode, requires JDK 1.6, 1.7, or 1.8).

To attach to a local application, you must first configure the attachment settings using the Attach Settings dialog box. These settings are associated with the project you are profiling. You only have to configure the attach settings for your project once, but you can modify the settings at any time in the Attach Settings dialog box.

When you configure the settings for attaching to a local application, you can choose from the following attachment modes:

- **Already running local Java process.** This mode allows you to detach from and attach to the application at any time without stopping the application. This mode requires JDK 1.6, 1.7, or 1.8 but does not require any additional configuration.
- **Manually started local Java process.** This mode enables you to obtain profiling data on the startup of the local application. This mode requires you to modify the startup script and start the application when you want to attach the profiler.

Once the connection is established and the profiler is attached to the target application, you can change the profiling task you are running on the target application without stopping or restarting the application.

**To attach to a local application:**

1. Choose **Profile > Attach to Project** () from the main menu to open the Profile window.
2. Click the **Attach** button to open the **Attach Settings** dialog box.
3. Select either **Already running local Java process** or **Manually started local Java process** as the attach mode.
4. Follow the instructions in the **Attach Settings** dialog box for configuring the application, if necessary. Click **OK**.

A profiling session tracking the running application is started.

**To finish profiler session:**

- Choose **Profile > Finish Profiler Session** from the main menu or click the **Finish profiler** button in the Profiler window.

When you finish profiler session, the connection to the target JVM is closed. You can re-attach to the application by choosing **Profile > Attach to Project** from the main menu.

**To change the profiling task without finishing profiler session:**

Click the **Attach** drop-down arrow to change the profiling mode at any point.

The target application is instrumented again without starting or stopping the application.

## 9.12 Attaching the Profiler to a Remote Application

You can profile an application that is running on a remote system by attaching the profiling tool to the application. When you use this attach mode, the remote application starts after the profiler is attached. This mode enables you to obtain profiling data on the startup of the target JVM.

To attach the profiling tool, you use the Attach Settings dialog box to specify the attachment settings for your project. In the Attach Settings dialog box, specify the location, OS and JVM of the remote target. Based on the details that you provide, the Attach Settings dialog box provides you with a set of instructions on how to configure the remote application to support profiling.

### 9.12.1 How to Attach to a Remote Application

To attach the profiler to a remote application you must configure the application to load some profiler libraries on startup to enable the profiling tool to attach to the application. You use the **Attach Settings** dialog box to specify the details of the remote

system and to generate a Remote Profiler Pack archive that contains the necessary profiler libraries. You must copy the contents of the Remote Profiler Pack archive to the remote system and then configure the remote application to load the profiler libraries.

After you configure the remote application according to the instructions, you can start the remote application and attach the profiler. You only have to configure the attach mode once. The attachment settings are associated with that project. You can open the Attach Settings dialog box at any time to change the attachment settings.

#### **To attach to a remote application:**

1. Choose **Profile > Attach to External Project** from the main menu. The **Profile External Process** window displays.
2. Select the **Configure Session** button and select **Setup Attach to Process**. The **Attach Settings** dialog box displays.
3. Set the profiler mode to **Manually started remote Java process** in the **Profile** drop-down list.
4. Specify the **Hostname** and select the OS and JVM from the drop-down list.
5. Follow the detailed instructions in the **Attach Settings** dialog to start profiling.
6. Click **OK**.

After the IDE is attached to the remote application you can do the following:

- Detach from the remote application.

When you detach from the remote application, the remote application does not stop but you stop receiving profiling data about the remote application. To attach to the remote application, use the startup options provided by the **Attach Settings** dialog box and start the remote application again.

- Modify the profiling session.

You can modify the profiling session without detaching from the remote application. For example, you can change the profiling task to monitoring to reduce the profiling overhead, and then modify the task again later. This way you do not have to re-attach and restart the remote application.

#### **9.12.2 Attaching to a Remote Server**

Similar to attaching the profiler to a remote application, you must copy the profiler libraries in the Remote Profiler Pack to the remote system to attach the profiling tool to a remote server. You must also modify the server configuration files to specify the path to the JDK and to specify the path to the profiler agent. When you start the server using the modified startup script, the server waits until the profiler attaches to the server.

You can retrieve the path to the profiler agent when you use the Attach Settings dialog box to configure your attach settings. The path to the profiler agent is similar to the following path.

```
-agentpath:<remote>\lib\deployed\jdk16\windows\profilerinterface.dll=\lib,
5140
```

The placeholder *<remote>* refers to the full path to the root directory containing the profiler libraries that you copied to the remote system. The number 5140 is the Communication Port that the profiling tool uses to connect to the application. Modify the port number in the Profiler tab in the Java category in the Options window.

**Table 9–5** identifies the startup scripts and the parameters that must be modified to specify the paths to the JDK and the profiler libraries:

**Table 9–5 Startup Scripts**

Server	File	Modification
Tomcat 7.x and 8.0.27	catalina.bat/catalina.sh	Set JAVA_HOME path to JDK For server, modify CATALINA_OPTS to include profile -agentpath parameter
GlassFish Server Open Source Edition 4.1.1	asenv.bat/asenv.conf domain.xml	Set AS_JAVA path to JDK For server, add <i>jvm-options</i> to include profile -agentpath parameter
WebLogic 12c	startWebLogic.cmd/startWebLogic.sh	Set JAVA_HOME path to JDK For server, modify JAVA_OPTIONS to include profile -agentpath parameter
JBoss AS 7.x	standalone.conf.bat/standalone.conf	Set JAVA_HOME path to JDK For server, modify JAVA_OPTS to include profile -agentpath parameter

For the GlassFish server, set the path to the JDK in the asenv.bat/asenv.conf file and the path to the profiler agent in domain.xml.

For more details about configuring the servers for attaching the profiler tool, see the following NetBeans FAQ:

<http://wiki.netbeans.org/wiki/view/FaqProfilerAttachRemoteServer>

For more details about modifying server startup scripts, consult the documentation for the server.

## 9.13 Profiling a Free-form Project

In a free-form project, you have to create Ant targets to be able to profile a file or project. You generally want one target in your free-form project's build script for profiling a project and one target for profiling individual files.

If you do not have a profile target written for your project the IDE will offer to generate a basic target for you when you first try to profile the project. You can then inspect the target and customize it according to the specific requirements of the project.

The first time that you choose the **Profile Project** or **Profile File** command on a free-form project you must create the targets. The IDE can create the targets for you based on the information for the target that is mapped to the **Run** command for the project. You only have to generate the targets once.

---

**Note:** Before the profile target is generated it is recommended that you first confirm that you have a target mapped to the Run Project command. When the IDE generates a profile target the IDE looks for the information in the target that is mapped to the Run Project command to determine details such as the run classpath and the project's main class. If a target is already mapped to the Run Project command there is a good chance that the generated profile target will work without further customization.

---

### To profile a free-form project:

1. Set the free-form project as the main project by choosing **Run > Set Main Project** in the main menu and selecting the project.
2. Choose **Profile > Profile Main Project** in the main menu.
3. Click the **Configure Session** button and select the required profiling mode.
4. Click the **Profile** button to start profiling.

Before profiling your free-form project, be aware of the following conditions:

- The project output must be set before you can profile a free-form application.
- If you click **Generate** in the **Profile Project** dialog box to generate the targets the IDE creates a build script named `ide-targets.xml` and generates a target named `profile-nb`. Verify that the generated `profile-nb` target properly takes into account all of the elements of your project. In some cases, you might have to modify the `classpath` argument in the target if it does not include all of the items in your run classpath.
- The IDE also modifies `project.xml` to map the `profile-nb` target to the **Profile Project** command in the IDE. If you write the target from scratch, you must also create this mapping yourself.

For example, the `profile-nb` target that the IDE generates in `ide-targets.xml` might look similar to the following example:

#### **Example 9-1 Sample profile-nb Target**

```
<?xml version="1.0" encoding="UTF-8"?>
<project basedir=".." name="YourProjectName">
    <import file="../build.xml"/>
    <target name="-profile-check">
        <startprofiler freeform="true"/>
    </target>
    <!-- TODO: edit the following target according to your needs -->
    <target name="profile-nb" if="profiler.configured"
depends="-profile-check">
        <java classname="${mainclass}" dir="." fork="true">
            <classpath>
                <pathelement path="ClasspathSpecifiedInYourRunTarget" />
            </classpath>
            <jvmarg line="${agent.jvmargs}" />
        </java>
    </target>
</project>
```

In this example the IDE attempted to guess the runtime classpath for the project.

- If you do not have a run target mapped or the IDE otherwise cannot determine the project's classpath or main class, the generated profile target includes "TODO" placeholders for you to fill in these values.

### 9.13.1 Profiling Free-form Web Projects

Typically the server is started using a startup script, not the **java** command directly. Therefore you cannot use the `<jvmarg ... />` element to pass additional argument to it. Instead, you have to modify the server startup script to configure it for profiling.

The recommended approach is to create a new script for starting the server for profiling and use it in the profile target. The Attach Settings dialog box can provide

steps to help you modify the startup script. If your target server does not support automatic integration, you can create the script by following the integration steps described in the Attach Settings dialog box. You always have to set up the integration for Local Direct attach.

The other steps for profiling a free-form web project are the same as those above for profiling a standard J2SE projects.

### 9.13.1.1 A Typical Free-Form Project Profile Target

The generated Ant target does the following:

- Starts the profiler with the startprofiler task. Setting the freeform attribute to true will force displaying the profiling session configuration dialog.
- The previous task sets the profiler.configured to true if the configuration was confirmed. It also stores the profiler agent JVM arguments in the agent.jvmargs property.
- Establishes the runtime classpath. If the IDE is not able to determine your runtime classpath the IDE adds placeholders to the script which you need to fill in yourself.
- Runs the application in profile mode. Setting fork="true" ensures the process is launched in a separate virtual machine.

---

**Note:** You can add any additional JVM arguments or program arguments in the java task as well.

---

A generated profile target where the IDE is able to guess the runtime classpath will look similar to the following (where the italicized items would have values specific to your project).

```
<?xml version="1.0" encoding="UTF-8"?>
<project basedir=".." name="YourProjectName">
    <import file="../build.xml"/>
    <target name="-profile-check">
        <startprofiler freeform="true"/>
    </target>
    <!-- TODO: edit the following target according to your needs -->
    <!-- (more info:
https://netbeans.org/kb/articles/freeform-config.html#profilej2se) -->
    <target name="profile-nb" if="profiler.configured"
depends="-profile-check">
        <java classname="${mainclass}" dir="." fork="true">
            <classpath>
                <path element path="ClasspathSpecifiedInYourRunTarget" />
            </classpath>
            <jvmarg line="${agent.jvmargs}" />
        </java>
    </target>
</project>
```

If you do not have a run target mapped or the IDE otherwise cannot determine the project's classpath or main class, the generated profile target includes "TODO" placeholders for you to fill in these values as in the example below.

```
<?xml version="1.0" encoding="UTF-8"?>
<project basedir=".." name="YourProjectName">
    <target name="-profile-check">
        <startprofiler freeform="true"/>
```

```

        </target>
        <!-- TODO: edit the following target according to your needs -->
        <!-- (more info:
            https://netbeans.org/kb/articles/freeform-config.html#profilej2se) -->
        <target depends="-profile-check" if="profiler.configured"
name="profile-nb">
            <path id="cp">
                <!-- TODO configure the runtime classpath for your project here:
-->
            </path>
            <!-- TODO configure the main class for your project here: -->
            <java classname="some.main.Class" fork="true">
                <classpath refid="cp"/>
                <jvmarg line="${agent.jvmargs}"/>
            </java>
        </target>
    </project>

```

To specify the runtime classpath, insert `pathelement` elements within the `path` element and point them to the directories that contain the items in your classpath. For example, you can use the `location` attribute of `pathelement` to specify the location of the classpath items relative to your project directory. The project directory is usually the directory that contains the project's `build.xml` file. Below is an example of using the `pathelement` attributes.

```

<path id="cp">
    <pathelement location="libs">
    <pathelement location="build">
</path>

```

### 9.13.1.2 Writing a Target to Profile a Selected File

The process is basically the same for writing targets to debug and run a single file. The `profile-selected-files` target looks similar to the following:

```

<target name="-profile-check">
    <startprofiler freeform="true"/>
</target>
<!-- TODO: edit the following target according to your needs -->
<!-- (more info: https://netbeans.org/kb/articles/freeform-config.html#profile_
sing) -->
<target depends="-profile-check" if="profiler.configured"
name="profile-selected-file-in-src">
    <fail unless="profile.class">Must set property 'profile.class'</fail>
    <path id="cp">
        <pathelement location="build"/>
    </path>>
    <java classname="\${profile.class}" fork="true">
        <classpath refid="cp"/>
        <jvmarg line="\${agent.jvmargs}"/>
    </java>
</target>

```

- This is basically the same as the `profile` target. Instead of passing the program main class to `java` you pass the `profile.class` property, which is set by the IDE to the currently selected file.

Then you map the `profile-selected-files` target to the `profile.single` action.

```

<action name="profile.single">
    <target>profile-selected-files</target>
    <context>
        <property>profile.class</property>

```

```
<folder>${src.dir}</folder>
<pattern>\.java$</pattern>
<format>java-name</format>
<arity>
    <one-file-only/>
</arity>
</context>
</action>
■ <property> now stores the context in the profile.class property.
■ Because java can only take a single file, you set <arity> to <one-file-only>.
■ Setting <format> to java-name and making it relative to src.dir creates a
fully-qualified class name for the currently selected file.
```

---

**Note:** The IDE does not define the \${src.dir} property for you. You need to define the property or import the .properties file that Ant is using in project.xml. See [Section 6.2.4.9.1, "Using Properties in the project.xml File"](#) for more information.

---

## 9.14 Taking and Accessing Snapshots of Profiling Data

A snapshot captures profiling data at a specific point in time and allows you to access them via the Snapshot window. See [Section 9.14.4, "Accessing Snapshots"](#).

A snapshot differs from live profiling results in the following ways:

- Snapshots can be examined when no profiling session is running.
- Snapshots can be easily compared.

There are two options for taking snapshots:

- While the profiling session is in progress. See [Section 9.14.1, "Taking Snapshots During a Profiling Session"](#)
- At the end of the profiling session. See [Section 9.14.2, "Taking Snapshots at the End of a Profiling Session"](#)

### 9.14.1 Taking Snapshots During a Profiling Session

You may take a snapshot of the profiling data at any time during the profiling session by clicking the **Snapshot** icon shown in the figure below.

To control how the snapshots functionality behaves during a session, go to **Tools > Options > Java > Snapshots > Profiler** and click the **When taking snapshots** drop-down menu to see the following options:

**Open Snapshot**—it opens the snapshot right after clicking the Snapshot icon

**Save Snapshot**—it saves a new snapshot every time you click the Snapshot Icon

**Save and open snapshot**—it saves and opens a snapshot right after clicking the Snapshot icon.

You may take multiple snapshots during a profiling session and you will also be prompted to save a "final" snapshot at the end of the session.

You can only take a snapshot while a profiling session is in progress. You can take a snapshot manually or set a profiling point to take a snapshot automatically at a precise point in your application. When you use a profiling point to take a snapshot, you

locate the point in your source code where you want to take a snapshot. For example, you may want to take a snapshot when a thread enters a specific method.

**Perform the following steps to take a snapshot manually:**

1. Start a profiling session.

For information, see [Section 9.9, "Starting a Profiling Session."](#)

2. Do one of the following:

- Click the **Take Snapshot** of Collected Results button in the Profiler window.
- Select **Profile > Take Snapshot of Collected Results** from the main menu.

When you take a snapshot, the snapshot opens in the main editor window. You can then save the snapshot to your project.

**To take a snapshot using a profiling point:**

1. Open the source file containing the code where you want to place the profiling point.
2. Right-click in the line of code and choose **Profile > Insert Profiling Point**.
3. Select one of the snapshot options in the **New Profiling Point** wizard and click **Next**.
4. Modify any of the profiling point properties in the **Customize Properties** page of the wizard. Click **Finish**.

If you use a profiling point to take a snapshot, you can choose from the following snapshot options:

- **Load Generator.** Choose this if you want the IDE to take a snapshot to start and stop a load generator script at the given source code location.
- **Reset Results.** Choose this if you want the IDE to take a snapshot to reset currently collected profiling results.
- **Stopwatch.** Choose this if you want the IDE to take a snapshot to measure time between start and stop locations to obtain the execution time of a method fragment.
- **Take Snapshot.** Choose this if you want the IDE to take a snapshot when an application thread enters or leaves the specified line of code.
- **Timed Take Snapshot.** Choose this if you want the IDE to take a snapshot at a specific time and date. You can also configure the IDE to take snapshots at a specified interval.
- **Triggered Take Snapshot.** Choose this if you want the IDE to take a snapshot when the specified condition is met. You specify the condition by choosing one of the available triggers.

For each snapshot option you can choose to save the snapshot to your project or you can specify a location.

**To save a snapshot:**

- Click **Save Snapshot to Project** in the snapshot toolbar.

You can also use the **Options** window to configure the IDE behavior when taking a snapshot.

You can save a snapshot to your project or you can save the snapshot to a location on your local file system. When you save a snapshot to your project, the snapshot is listed under Snapshots in the **Snapshots** window.

To distinguish a snapshot from other profiling snapshots you have taken of the project, you can rename it. Select a snapshot in the Snapshots section of the Snapshots window and click **Rename selected snapshot** to open the **Rename Snapshot** dialog box and enter a new name.

### 9.14.2 Taking Snapshots at the End of a Profiling Session

When closing a profiled application, or if it finishes on its own, while the profiling session is in progress, the profiler asks you whether to take a snapshot of the results collected so far by displaying the **Application Finished** dialog.

Click **Yes** to save the snapshot.

### 9.14.3 Starting and Stopping the Application Finished Dialog

When the **Application Finished** dialog appears at the end of the profiling session, if you select the **Do not show this message again** checkbox the dialog would not display again. If at a later time you want to reactivate the display of this dialog, go to **Tools Options > Java > Snapshots > Profiler > General > Miscellaneous** and click the **Reset** button.

### 9.14.4 Accessing Snapshots

You may access your profiling session snapshots by going to **Window > Profiling > Snapshots**. The **Snapshots** window appears.

At the bottom of the **Snapshots** window there are icons that allow you to export, open, rename, and delete selected snapshots.

## 9.15 Taking a Heap Dump

A *heap dump* captures profiling data at a specific point in time. You can take a heap dump when a profiling session is in progress. When you take a heap dump, you are prompted to save the heap to your project or local file system. After you save a heap dump you can load the heap dump at any time and browse the objects on the heap, locate references to individual objects and compare heap dumps to view the differences between the snapshots. It is not necessary to run a profiling session to load and browse the heap dump.

---

**Note:** To take a heap dump, the application must be running on a version of JDK 1.5.0\_12 or later.

---

### 9.15.1 How to Take a Heap Dump

You can take a heap dump manually or set a profiling point to take a heap dump automatically at a certain point in your application. In the Options window, you can also set the IDE to automatically take a heap dump on OutOfMemory error.

#### To take a heap dump manually:

1. Start a profiling session.
2. Choose **Profile > Take Heap Dump** in the main menu.

When you take the heap dump, you are prompted to specify where you want to save the heap dump. You can save the heap dump snapshot to your project or to any location on your local file system.

**To take a heap dump using a profiling point:**

1. Open the source file containing the code where you want to place the profiling point.
2. Right-click in the line of code where you want to place the profiling point and choose **Profile > Insert Profiling Point**.
3. In the New Profiling Point wizard, select one of the following snapshot options and click **Next**.
  - **Take Snapshot**
  - **Timed Take Snapshot**
  - **Triggered Take Snapshot**
4. In the **Customize Properties** page of the wizard, select **Heap Dump** as the type of snapshot and modify any additional settings. Click **Finish**.

When you use a profiling point to take a heap dump, you specify the point in your source code where you want to place the profiling point. For example, you may want to take a heap dump when a thread enters a specific method.

**To take a heap dump on OutOfMemory error:**

1. Choose **Tools > Options** from the main menu, click the **Java** category and then click the **Profiler** tab.
2. Choose the **Snapshots** category.
3. In the **On OutOfMemoryError** list, select an option from the drop-down list to specify what the IDE does when an **OutOfMemoryError** is encountered.

The default behavior is to save the heap dump to the profiled project.

### 9.15.2 How to Analyze a Heap Dump Using Object Query Language (OQL)

OQL is a SQL-like query language to query a Java heap that enables you to filter/select information wanted from the Java heap. While pre-defined queries such as "show all instances of class X" are already supported by the tool, OQL adds more flexibility. OQL is based on JavaScript expression language.

When you load a Java heap in the **Heap** window, you can click the **OQL Console** tab of the window to open the OQL editor. The OQL Console contains an OQL editor, a saved OQL queries window and a window that displays the query results. You can use any of the sample OQL queries or create a query to filter and select heap data to locate the information that you want from the Java heap. After you choose or write a query, you can run the query against the Java heap and view the results.

An OQL query is of the following form:

```
select <JavaScript expression to select>
[ from [instanceof] <class name> <identifier>
[ where <JavaScript boolean expression to filter> ] ]
where class name is fully qualified Java class name (example: java.net.URL) or array
class name. char[] (or [C] is char array name, java.io.File (or [Ljava.io.File;) is
name of java.io.File[] and so on. Note that fully qualified class name does not always
uniquely identify a Java class at runtime. There may be more than one Java class with
```

the same name but loaded by different loaders. So, class name is permitted to be id string of the class object. If instanceof keyword is used, subtype objects are selected. If this keyword is not specified, only the instances of exact class specified are selected. Both from and where clauses are optional.

In select and (optional) where clauses, the expression used in JavaScript expression. Java heap objects are wrapped as convenient script objects so that fields may be accessed in natural syntax. For example, Java fields can be accessed with `obj.field_name` syntax and array elements can be accessed with `array[index]` syntax. Each Java object selected is bound to a JavaScript variable of the identifier name specified in from clause.

### 9.15.2.1 OQL Examples

Select all Strings of length 100 or more:

```
select s from java.lang.String s where s.count >= 100
```

Select all int arrays of length 256 or more:

```
select a from int[] a where a.length >= 256
```

Show content of Strings that match a regular expression:

```
select {instance: s, content: s.toString()} from java.lang.String s  
      where /java/(s.toString())
```

Show path value of all File objects:

```
select file.path.toString() from java.io.File file
```

Show names of all ClassLoader classes:

```
select classof(cl).name  
      from instanceof java.lang.ClassLoader cl
```

Show instances of the Class identified by given id string:

```
select o from instanceof 0xd404b198 o
```

0xd404b198 is id of a Class (in a session). This is found by looking at the id shown in that class's page.

### 9.15.2.2 OQL built-in objects and functions

#### Heap object

The heap built-in object supports the following methods:

- `heap.forEachClass` - calls a callback function for each Java Class

```
heap.forEachClass(callback);
```

- `heap.forEachObject` - calls a callback function for each Java object

```
heap.forEachObject(callback, clazz, includeSubtypes);
```

`clazz` is the class whose instances are selected. If not specified, defaults to `java.lang.Object`. `includeSubtypes` is a boolean flag that specifies whether to include subtype instances or not. Default value of this flag is true.

- `heap.findClass` - finds Java Class of given name

```
heap.findClass(className);
```

where `className` is name of the class to find. The resulting Class object has following properties:

- `name` - name of the class.

- superclass - Class object for super class (or null if java.lang.Object).
- statics - name, value pairs for static fields of the Class.
- fields - array of field objects. field object has name, signature properties.
- loader - ClassLoader object that loaded this class.

Class objects have the following methods:

- isSubclassOf - tests whether given class is direct or indirect subclass of this class or not.
- isSuperclassOf - tests whether given Class is direct or indirect superclass of this class or not.
- subclasses - returns array of direct and indirect subclasses.
- superclasses - returns array of direct and indirect superclasses.
- heap.findObject - finds object from given object id  
`heap.findObject(stringIdOfObject);`
- heap.classes - returns an enumeration of all Java classes
- heap.objects - returns an enumeration of Java objects  
`heap.objects(clazz, [includeSubtypes], [filter])`

`clazz` is the class whose instances are selected. If not specified, defaults to `java.lang.Object`. `includeSubtypes` is a boolean flag that specifies whether to include subtype instances or not. Default value of this flag is true. This method accepts an optional filter expression to filter the result set of objects.

- heap.finalizables - returns an enumeration of Java objects that are pending to be finalized.
- heap.livepaths - return an enumeration of paths by which a given object is alive. This method accepts optional second parameter that is a boolean flag. This flag tells whether to include paths with weak reference(s) or not. By default, paths with weak reference(s) are not included.

```
select heap.livepaths(s) from java.lang.String s
```

Each element of this array itself is another array. The later array is contains an objects that are in the 'reference chain' of the path.

- heap.roots - returns an Enumeration of Roots of the heap.

Each Root object has the following properties:

- id - String id of the object that is referred by this root
- type - descriptive type of Root (JNI Global, JNI Local, Java Static, etc.)
- description - String description of the Root
- referrer - Thread Object or Class object that is responsible for this root or null

## Examples

- Access static field 'props' of class `java.lang.System`

```
select heap.findClass("java.lang.System").statics.props
select heap.findClass("java.lang.System").props
```

- Get number of fields of `java.lang.String` class

```
select heap.findClass("java.lang.String").fields.length

■ Find the object whose object id is given
select heap.findObject("0xf3800b58")

■ Select all classes that have name pattern java.net.*
select filter(heap.classes(), "/java.net./(it.name) ")
```

### Functions on individual objects

- allocTrace function

Returns allocation site trace of a given Java object if available. allocTrace returns array of frame objects. Each frame object has the following properties:

- className - name of the Java class whose method is running in the frame.
- methodName - name of the Java method running in the frame.
- methodSignature - signature of the Java method running in the frame.
- sourceFileName - name of source file of the Java class running in the frame.
- lineNumber - source line number within the method.

- classof function

Returns class object of a given Java object. The resulting object supports the following properties:

- name - name of the class
- superclass - class object for super class (or null if `java.lang.Object`)
- statics - name, value pairs for static fields of the class
- fields - array of field objects. Field objects have name, signature properties
- loader - ClassLoader object that loaded this class.

Class objects have the following methods:

- isSubclassOf - tests whether given class is direct or indirect subclass of this class or not
- isSuperclassOf - tests whether a given class is direct or indirect superclass of this class or not
- subclasses - returns array of direct and indirect subclasses
- superclasses - returns array of direct and indirect superclasses

### Examples

- Show class name of each Reference type object

```
select classof(o).name from instanceof java.lang.ref.Reference o
```

- Show all subclasses of `java.io.InputStream`

```
select heap.findClass("java.io.InputStream").subclasses()
```

- Show all superclasses of `java.io.BufferedInputStream`

```
show all superclasses of java.io.BufferedInputStream
```

- **forEachReferrer function**

Calls a callback function for each referrer of a given Java object.

- **identical function**

Returns whether two given Java objects are identical or not, for example:

```
select identical(heap.findClass("Foo").statics.bar,
    heap.findClass("AnotherClass").statics.bar)
```

- **objectid function**

Returns String id of a given Java object. This id can be passed to `heap.findObject` and may also be used to compare objects for identity. For example:

```
select objectid(o) from java.lang.Object o
```

- **reachables function**

Returns an array of Java objects that are transitively referred from the given Java object. Optionally accepts a second parameter that is comma separated field names to be excluded from reachability computation. Fields are written in `class_name.field_name` pattern.

### Examples

- Print all reachable objects from each `Properties` instance.

```
select reachables(p) from java.util.Properties p
```

- Print all reachables from each `java.net.URL` but omit the objects reachable via the fields specified.

```
select reachables(u, 'java.net.URL.handler') from java.net.URL u
```

- **referrers function**

Returns an enumeration of Java objects that hold reference to a given Java object. This method accepts optional second parameter that is a boolean flag. This flag tells whether to include weak reference(s) or not. By default, weak reference(s) are not included.

### Examples

- Print number of referrers for each `java.lang.Object` instance

```
select count(referrers(o)) from java.lang.Object o
```

- Print referrers for each `java.io.File` object

```
select referrers(f) from java.io.File f
```

- Print URL objects only if referred by 2 or more

```
select u from java.net.URL u where count(referrers(u)) > 2
```

- **referees function**

Returns an array of Java objects to which the given Java object directly refers to. This method accepts optional second parameter that is a boolean flag. This flag tells whether to include weak reference(s) or not. By default, weak reference(s) are not included. For example, to print all static reference fields of `java.io.File` class:

```
select referees(heap.findClass("java.io.File"))
```

- **refers** function  
Returns whether first Java object refers to second Java object or not.
- **root** function  
If the given object is a member of root set of objects, this function returns a descriptive Root object describing why it is so. If given object is not a root, then this function returns null.
- **sizeof** function  
Returns size of the given Java object in bytes, for example:  

```
select sizeof(o) from int[] o
```
- **retainedsize** function  
Returns size of the retained set of the given Java object in bytes. **Note:** Using this function for the first time on a heap dump may take significant amount of time.  
The following is an example usage of the retainedsize function:  

```
select rssizeof(o) from instanceof java.lang.HashMap o
```
- **toHtml** function  
Returns HTML string for the given Java object. Note that this is called automatically for objects selected by select expression. But, it may be useful to print more complex output. For example, to print a hyperlink in bold font:  

```
select "<b>" + toHtml(o) + "</b>" from java.lang.Object o
```

### 9.15.2.3 Selecting Multiple Values

Multiple values can be selected using JavaScript object literals or arrays.

For example, show the name and thread for each thread object

```
select { name: t.name? t.name.toString() : "null", thread: t }  
from instanceof java.lang.Thread t
```

### array/iterator/enumeration manipulation functions

These functions accept an array/iterator/enumeration and an expression string [or a callback function] as input. These functions iterate the array/iterator/enumeration and apply the expression (or function) on each element. **Note:** JavaScript objects are associative arrays. So, these functions may also be used with arbitrary JavaScript objects.

- **concat** function  
Returns whether the given array/enumeration contains an element the given boolean expression specified in code. The code evaluated can refer to the following built-in variables.
  - **it** - currently visited element
  - **index** - index of the current element
  - **array** - array/enumeration that is being iterated

For example, to select all Properties objects that are referred by some static field some class:

```
select p from java.util.Properties p
where contains(referrers(p), "classof(it).name == 'java.lang.Class'")
```

- **count function**

Returns the count of elements of the input array/enumeration that satisfy the given boolean expression. The boolean expression code can refer to the following built-in variables.

- it - currently visited element
- index - index of the current element
- array - array/enumeration that is being iterated

For example, print the number of classes that have a specific name pattern:

```
select count(heap.classes(), "/java.io./(it.name)")
```

- **filter function**

Returns an array/enumeration that contains elements of the input array/enumeration that satisfy the given boolean expression. The boolean expression code can refer to the following built-in variables.

- it - currently visited element
- index - index of the current element
- array - array/enumeration that is being iterated
- result -> result array/enumeration

### Examples

- Show all classes that have `java.io.*` name pattern

```
select filter(heap.classes(), "/java.io./(it.name)")
```

- Show all referrers of URL object where the referrer is not from java.net package

```
select filter(referrers(u), " ! /java.net./(classof(it).name)")
from java.net.URL u
```

- **length function**

Returns number of elements of an array/enumeration.

- **map function**

Transforms the given array/enumeration by evaluating given code on each element. The code evaluated can refer to the following built-in variables.

- it - currently visited element
- index - index of the current element
- array - array/enumeration that is being iterated
- result -> result array/enumeration

Map function returns an array/enumeration of values created by repeatedly calling code on each element of input array/enumeration.

For example, show all static fields of `java.io.File` with name and value:

```
select map(heap.findClass("java.io.File").statics, "index + '=' + toHtml(it))
```

- max function

Returns the maximum element of the given array/enumeration. Optionally accepts code expression to compare elements of the array. By default numerical comparison is used. The comparison expression can use the following built-in variables:

- lhs - left side element for comparison
- rhs - right side element for comparison

**Examples**

- Find the maximum length of any string instance

```
select max(map(heap.objects('java.lang.String', false), 'it.count'))
```

- Find string instance that has the maximum length

```
select max(heap.objects('java.lang.String'), 'lhs.count > rhs.count')
```

- min function

Returns the minimum element of the given array/enumeration. Optionally accepts code expression to compare elements of the array. By default numerical comparison is used. The comparison expression can use the following built-in variables:

- lhs - left side element for comparison
- rhs - right side element for comparison

**Examples**

- Find the minimum size of any vector instance

```
select min(map(heap.objects('java.util.Vector', false),  
'it.elementAt.length'))
```

- Find vector instance that has the maximum length

```
select min(heap.objects('java.util.Vector'), 'lhs.elementAt.length <  
rhs.elementAt.length')
```

- sort function

Sorts a given array/enumeration. Optionally accepts code expression to compare elements of the array. By default numerical comparison is used. The comparison expression can use the following built-in variables:

- lhs - left side element for comparison
- rhs - right side element for comparison

**Examples**

- Print all char[] objects in the order of size.

```
select sort(heap.objects('char[]'), 'sizeof(lhs) - sizeof(rhs)')
```

- Print all char[] objects in the order of size but print size as well.

```
select map(sort(heap.objects('char[]'), 'sizeof(lhs) - sizeof(rhs)'), '{  
size: sizeof(it), obj: it }')
```

- top function

Returns top N elements of the given array/enumeration. Optionally accepts code expression to compare elements of the array and the number of top elements. By default the first 10 elements in the order of appearance is returned. The comparison expression can use the following built-in variables:

- lhs - left side element for comparison
- rhs - right side element for comparison

#### Examples

- Print 5 longest strings

```
select top(heap.objects('java.lang.String'), 'rhs.count - lhs.count', 5)
```

- Print 5 longest strings but print size as well.

```
select map(top(heap.objects('java.lang.String'), 'rhs.count - lhs.count', 5), '{ length: it.count, obj: it }')
```

- sum function

Returns the sum of all the elements of the given input array or enumeration. Optionally, accepts an expression as second param. This is used to map the input elements before summing those.

For example, return the sum of sizes of the reachable objects from each Properties object:

```
select sum(map(reachables(p), 'sizeof(it)'))
from java.util.Properties p
```

```
// or omit the map as in ...
select sum(reachables(p), 'sizeof(it)')
from java.util.Properties p
```

- toArray function

Returns an array that contains elements of the input array/enumeration.

- unique function

Returns an array/enumeration containing unique elements of the given input array/enumeration.

The following example selects a unique char[] instances referenced from strings. Note that more than one string instance can share the same char[] for the content.

```
// number of unique char[] instances referenced from any String
select count(unique(map(heap.objects('java.lang.String'), 'it.value')))

// total number of Strings
select count(heap.objects('java.lang.String'))
```

#### 9.15.2.4 Other Examples

The following example prints a histogram of each class loader and number of classes loaded by it.

`java.lang.ClassLoader` has a private field called `classes` of type `java.util.Vector` and `Vector` has a private field named `elementCount` that is number of elements in the vector. The query selects multiple values (`loader, count`) using JavaScript object literal and `map` function. It sorts the result by `count` (i.e., number of classes loaded) using `sort` function with comparison expression.

```
select map(sort(map(heap.objects('java.lang.ClassLoader'),
'{ loader: it, count: it.classes.elementCount }')), 'lhs.count < rhs.count'),
'toHtml(it) + "<br>"')
```

The following example shows the parent-child chain for each class loader instance.

```
select map(heap.objects('java.lang.ClassLoader'),
function (it) {
    var res = '';
    while (it != null) {
        res += toHtml(it) + ">>";
        it = it.parent;
    }
    res += "null";
    return res + "<br>";
})
```

Note that the parent field of `java.lang.ClassLoader` class is used and the example walks until the parent is null using the callback function to map call.

The following example prints the value of all System properties. Note that this query (and many other queries) may not be stable - because private fields of the Java platform classes may be modified or removed without any notification (implementation detail). But using such queries on user classes may be safe, given that you have control over the classes.

```
select map(filter(heap.findClass('java.lang.System').props.table, 'it != null &&
it.key != null && it.value != null'),
function (it) {
    var res = it.key.toString() + ' = ' + it.value.toString();
    return res;
});
```

- `java.lang.System` has static field by name '`props`' of type `java.util.Properties`.
- `java.util.Properties` has field by '`table`' of type `java.util.Hashtable$Entry` (this field is inherited from `java.util.Hashtable`). This is the hashtable buckets array.
- `java.util.Hashtable$Entry` has `key`, `value` and `next` fields. Each entry points the next entry (or null) in the same hashtable bucket.
- `java.lang.String` class has a `value` field of type `char[]`.

## 9.16 Setting a Profiling Point

A profiling point is a marker in your source code that can invoke specific profiling actions. You set a profiling point in your code by using the context menu in the Source Editor or by using the toolbar in the Profiling Points window.

You can set the following types of profiling points:

- Reset Results
- Stopwatch
- Take Snapshot
- Load Generator
- Timed Take Snapshot
- Triggered Take Snapshot

You can use a profiling point to reset profiling results, take a snapshot or record the timestamp or execution time of a code fragment.

You can also use a profiling to stop and start a load generator script (requires the load generator plugin).

### 9.16.1 How to Set Profiling Points

Once you set a profiling point it becomes part of the project until you delete it. You can view, modify and delete the Profiling Points in your projects in the Profiling Points window. The Profiling Points window displays all profiling points in open projects. You can select a profiling point and then use the toolbar to edit, remove, enable and disable the profiling point.

**To open the New Profiling Points window:**

- Choose **Profile > Insert Profiling Points** in the main menu

**To set a profiling point:**

1. Locate the class where you want to add the profiling point and open the class in the Source Editor.
2. In the Source Editor, right-click in the line where you want to add the profiling point and choose **Profiling > Insert Profiling Point** to open the New Profiling Point wizard.
3. Select a profiling point type and the project and click **Next**.
4. Customize the properties of the profiling point, if necessary and click **Finish**.

When you click **Finish**, an icon representing the profiling point type appears in the sidebar of the Source Editor next to the line where you inserted the profiling point.

**To enable and disable a profiling point:**

1. Locate the class containing the profiling point and open the class in the Source Editor.
2. In the Source Editor, right-click in the left margin of the line containing the profiling point and choose **Profiling Point > Disabled** or **Enabled**.

**To enable, disable, or customize the settings of a profiling point:**

- Choose **Window > Profiling > Profiling Points**.

**To view details about a profiling point:**

You can view information about a profiling point in the Profiling Points Report window. The information displayed is dependent on the profiling point type. Right-click a profiling point in the **Profiling Points** window and choose **Show Report** in the context menu to open this window.

### 9.16.2 How to Reset Profiling Results

You can use a profiling point to reset the collected profiling results each time that a thread in the application hits the profiling point. You can then use this profiling point to gather results deltas when combined with the Take Snapshot profiling point.

**To reset a profiling point:**

1. Click **Add Profiling Point** or **Edit Profiling Point** in the Profiling Points window.

2. Select the **Reset Results** profiling point type and select the desired project.
3. Click **Next**.
4. Specify a name that identifies that profiling point.
5. Specify the line number containing the profiling point and whether the profiling point is hit when a thread hits the beginning or end of that line (the file name is automatically filled in).
6. Click **Finish**.

### 9.16.3 How to Set a Stopwatch Profiling Point

You can obtain a timestamp each time that a profiling point is hit instead of calling the `System.currentTimeMillis()` method. Setting a Stopwatch profiling point lets you measure the time between the start and stop locations to obtain the execution time of a method fragment.

**To set a Stopwatch profiling point:**

1. Choose **Window > Profiling > Profiling Points**.
2. Click **Add Profiling Point** in the **Profiling Points** window.
3. Select the Stopwatch profiling point type and select the desired project.
4. Click **Next**.
5. Specify a name that identifies that profiling point.
6. Select whether you want to obtain a timestamp or a timestamp and the execution duration.
7. Specify the file in which to set the profiling point.
8. Specify the line number containing the profiling point at which point time measurement begins (the file name is automatically filled in).
9. If you selected Timestamp and duration, specify the line at which to stop the measurement (the file name is automatically filled in).
10. Click **Finish**.

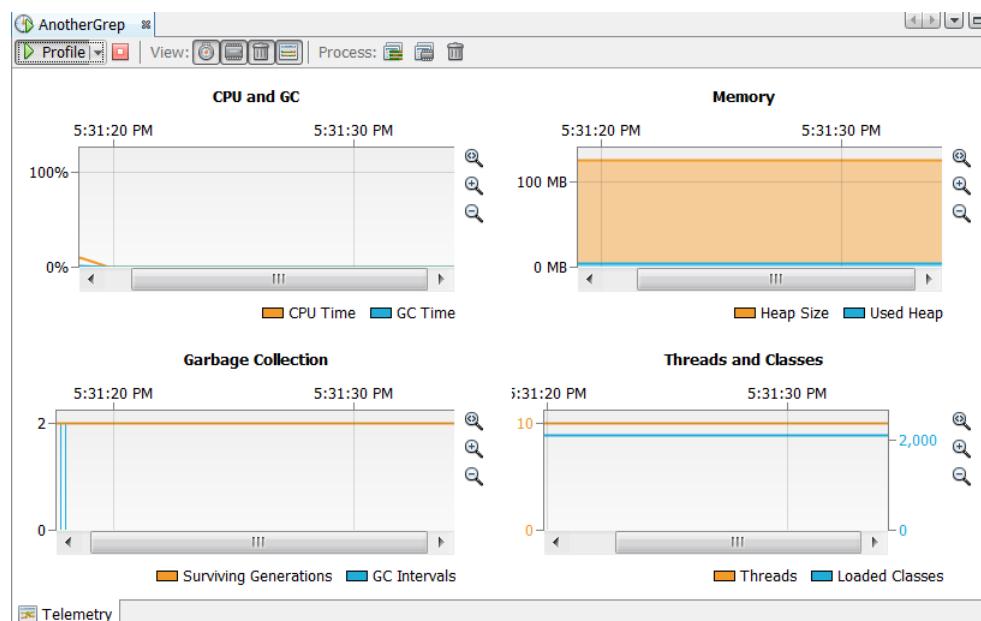
## 9.17 Profiling Telemetry

The telemetry mode provides the following metrics:

- **CPU and GC**—displays the CPU and GC percentage of use at a given time
- **Memory**—displays in MB the heap size and used heap at a given time
- **Surviving Generations**—displays the number of surviving generations at a given time. It also displays indicates the GC intervals
- **Threads and Classes**—displays number of loaded classes and threads at a given time

To start a profiling telemetry session, see [Section 9.9, "Starting a Profiling Session"](#).

[Figure 9–1](#) shows a snapshot of a telemetry session.

**Figure 9–1 Telemetry Session**

When running in the Telemetry mode the profiler monitors the target application with a very low overhead. A Thread Dump or Heap Dump can be taken from the profiled application. The garbage collection in the target VM can be requested using a toolbar button.

Each graph displays the full data for a profiling session and can be zoomed and panned separately by the control buttons and mouse wheel. A graph can be maximized to display more details by hiding the other graphs using the toolbar buttons.

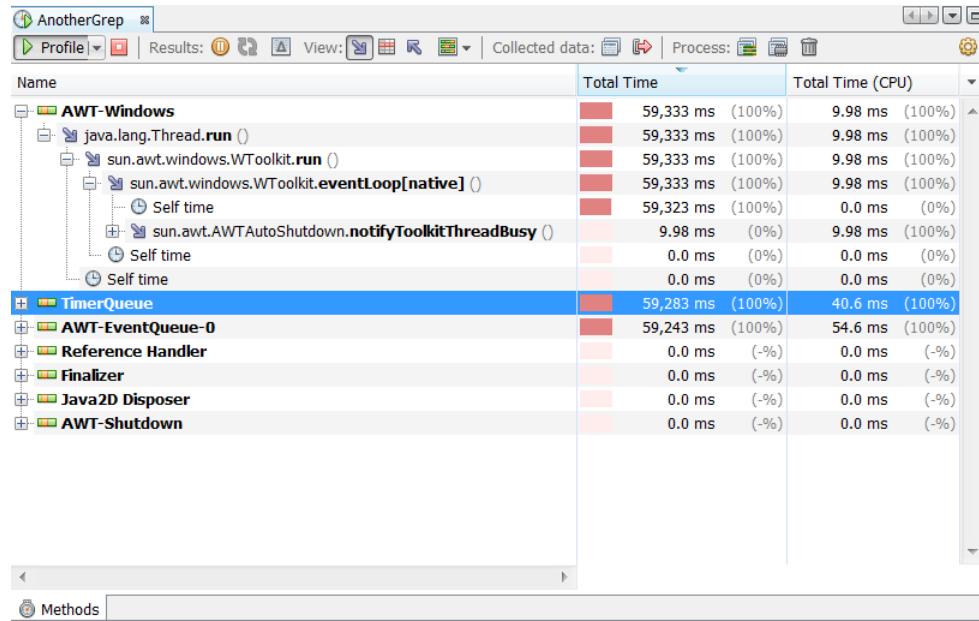
For the description of the toolbar buttons see [Section 9.9.3, "Understanding the Toolbar Icons"](#).

For more information on profiling methods in NetBeans IDE, see <http://wiki.netbeans.org/ProfilerTelemetry>.

## 9.18 Profiling Methods

The methods profiling mode tracks methods call trees, execution times and invocations count if configured, enabling to measure and optimize application performance. Use this profiling mode if experiencing slow responses or optimizing algorithms for speed of execution.

[Figure 9–2](#) shows a snapshot of a methods session.

**Figure 9–2 Methods Session**

For the description of the toolbar buttons see [Section 9.9.3, "Understanding the Toolbar Icons"](#).

To start a profiling methods session, see [Section 9.9, "Starting a Profiling Session"](#).

### 9.18.1 Basic Methods Profiling Mode

The methods mode is designed to provide profiling data from all classes and methods without any initial setup. The methods mode provides metrics for methods and classes. The methods report allows you to view the data by **Forward Calls**, **Hot Spots**, and **Reverse Calls** by clicking on the appropriate icons.

[Table 9–6](#) lists three different views on the methods data collected by the profiler.

**Table 9–6 Methods Views**

Methods View Name	Description
Forward calls	Shows the methods execution tree, from a thread down to single method calls. The Self time node represents execution time of the method, without any outgoing calls.
Hot spots	Shows a flat list of methods without the incoming/outgoing context. If sorted by the Self Time, it can immediately uncover an obvious performance bottleneck.
Reverse calls	Shows a flat list of methods for each thread and a tree of all execution points of these methods, transitively. It helps to discover the right call tree affected by a slowdown.

You may also choose to **Show Delta Values** and **Select Threads**.

- **Show Delta Values.** Switches from absolute values to incremental values. The values displayed prior to switching the view are remembered but the new view displays changes starting at the moment the new selection was made. Clicking this icon again resets the results back to absolute values.

- **Select Threads.** Shows threads available in live results or a saved snapshot and allows you to select specific threads for displaying results. This feature is useful when tracking EDT slowness in desktop applications or analyzing worker threads in server applications. This feature merges results from the selected threads to a single tree.

Additionally, you may select the columns to be displayed; the options are **Total Time**, **Total Time (CPU)**, **Selected**, and **Hits/Invocations** (depending on the session configuration).

### 9.18.1.1 Selecting Threads

By default results for all threads are displayed. By clicking the **Select threads** button it's possible to select just some threads and the methods executed by other threads won't be displayed in the results.

Initially the **Show all threads** option is selected, which means that results of all threads including any new threads started in future are displayed. Deselecting this option clears all selected threads, enabling to quickly select one desired thread. Clicking a thread in the Select thread table also deselects the option, but keeps the other threads selected, which makes it easier to remove just one thread from the results.

When the **Show all threads** option is deselected, the **Merge selected threads** becomes enabled. This option allows to merge methods executed from different threads into single call tree, simplifying analysis of profiling data from multiple worker threads.

### 9.18.1.2 Searching And Filtering Results

To find a method in results, either:

- use the Find stripe which opens by invoking the Find action in results context menu or in the Edit menu
- use the Ctrl - F keyboard shortcut

To filter collected results, either:

- use the Filter stripe which opens by invoking the Filter action in the results context menu
- use the Ctrl - G keyboard shortcut

For more information on profiling methods in NetBeans IDE, see <http://wiki.netbeans.org/ProfilerMethods>.

## 9.18.2 Advanced Methods Profiling Mode

To change the profiling modes and settings in the Settings pane, press the **Settings** switch in the Profiler window toolbar.

**Table 9–7** lists the profiling modes the profiler offers with different profiling techniques and settings.

**Table 9–7 Methods Profiling Modes**

Name	Description
All classes	Uses sampling and collects profiling data from methods of all classes. This mode is selected by default and doesn't require any additional configuration.

**Table 9–7 (Cont.) Methods Profiling Modes**

Name	Description
Project classes	Uses sampling and collects data from methods of projects classes. This mode doesn't require any additional configuration.
Selected classes	Uses bytecode instrumentation and collects data from methods of the defined classes. To use this mode, at least one class has to be selected for profiling.
Selected methods	Uses bytecode instrumentation and collects data from the defined methods. To use this mode, at least one method has to be selected for profiling.

For more information on profiling methods in NetBeans IDE, see  
<http://wiki.netbeans.org/ProfilerMethods>.

### 9.18.3 Selecting Classes And Methods For Profiling

Selected classes or methods instruct the profiler from where to start collecting the profiling data. Once a JVM executes a selected method or the method of a selected class, the profiler starts collecting data of this method and all methods called by this method, transitively.

A class or method for instrumented profiling can be selected via:

- **Select Class** or **Select Method** dialog. Allows to select one or several classes or methods from a project or .class/.jar file. Can be opened directly from the Methods Settings pane.
- **Code editor** context menu. Provides **Profile** | **Profile Class** and **Profile** | **Profile Method** actions.
- **Navigator** context menu. Provides **Profile** | **Profile Class** and **Profile** | **Profile Method** actions.
- **Profiling results**. Live or saved Methods results provide **Profile Class** and **Profile Method** actions in the context menu. The **Selected** column is available to select methods for profiling. The column can be displayed using the right corner button in the view header by clicking the drop-down arrow (or right clicking the header on Mac OS X).

For more information on profiling methods in NetBeans IDE, see  
<http://wiki.netbeans.org/ProfilerMethods>.

### 9.18.4 Configuring Additional Options

Additional options define where to stop collecting the profiling data in order to lower the overhead and keep the profiling data clear and focused.

By default the profiler offers a limited subset of customizable options for an instrumented profiling session.

To enable the expert mode:

1. Choose **Tools** > **Options** > **Java** > **Profiler** > **General** from the main menu.
2. Select the **Enable manual setup for Methods and Objects (expert users)** checkbox.

For more information on the expert mode of profiling methods in NetBeans IDE, see  
<http://wiki.netbeans.org/ProfilerExpertFeatures>.

## 9.19 Profiling Objects

The objects mode provides a list of classes allocated to a project including live instances and bytes allocation.

Figure 9–3 shows a snapshot of an objects session.

**Figure 9–3 Objects Session**

Name	Live Bytes	Live Objects
char[]	1,964,608 B (26.8%)	24,222 (27.2%)
int[]	1,424,968 B (19.5%)	1,222 (1.4%)
byte[]	1,413,336 B (19.3%)	666 (0.7%)
java.lang.String	578,208 B (7.9%)	24,092 (27%)
java.util.HashMap\$Node	455,488 B (6.2%)	14,234 (16%)
java.lang.Class	265,200 B (3.6%)	2,485 (2.8%)
java.util.HashMap\$Node[]	180,576 B (2.5%)	244 (0.3%)
long[]	175,976 B (2.4%)	96 (0.1%)
java.lang.Object[]	113,168 B (1.5%)	1,962 (2.2%)
java.lang.reflect.Method	57,376 B (0.8%)	652 (0.7%)
java.util.Hashtable\$Entry	44,256 B (0.6%)	1,383 (1.6%)
java.lang.reflect.Field	42,120 B (0.6%)	585 (0.7%)
java.util.LinkedHashMap\$Entry	32,880 B (0.4%)	822 (0.9%)
java.lang.Integer	32,192 B (0.4%)	2,012 (2.3%)
java.lang.String[]	25,848 B (0.4%)	638 (0.7%)
java.util.concurrent.ConcurrentHashMap\$Node	22,656 B (0.3%)	708 (0.8%)
java.security.AccessControlContext	16,280 B (0.2%)	407 (0.5%)
java.util.Hashtable\$Entry[]	15,400 B (0.2%)	54 (0.1%)
java.lang.Class[]	14,368 B (0.2%)	683 (0.8%)
java.util.HashMap	13,584 B (0.2%)	283 (0.3%)
Objects	44,256 B (0.6%)	704 (0.8%)

To select the classes to be profiled, click the **Settings** icon in the top-right corner and choose the required option in the drop-down menu:

- **All Classes.** Shows all classes and object that are live on the Virtual Machine heap.
- **Project Classes.** Allows to view only the classes defined in the project.

To start a profiling objects session, see [Section 9.9, "Starting a Profiling Session"](#).

For the description of the toolbar buttons see [Section 9.9.3, "Understanding the Toolbar Icons"](#).

For more information on profiling objects in NetBeans IDE, see <http://wiki.netbeans.org/ProfilerObjects>.

### 9.19.1 Basic Profiling

The Objects mode provides profiling data from all classes without any initial setup. Click the Profile or Attach button in the toolbar to start a profiling session.

#### 9.19.1.1 Results View

The default results view shows a histogram of the classes currently live in the heap memory with instance numbers and sizes. The view provides data columns listed in [Figure 9–8](#).

**Table 9–8 Default Results View**

Column	Description
Name	Shows name of the class with live instances on heap.
Live Bytes	Shows the size of all instances of the class currently live on heap.
Live Objects	Shows the number of all instances of the class currently live on heap.

**Note:** The columns can be displayed or hidden using the right corner button in the view header by clicking the drop-down arrow (or right clicking the header on Mac OS X).

### 9.19.1.2 Searching And Filtering Results

A class can be found in results using the **Find stripe** which opens by invoking the **Find action** in results context menu or IDE Edit menu or using the **Ctrl-F** keyboard shortcut. Values of the Name column are searched for the entered substring, optionally matching case if selected. Once a search has been performed, it can be repeated using the **F3** or **Shift+F3** shortcuts. Next appearance of the currently selected value of Name column can be found using the **Ctrl+F3** shortcut.

Collected results can be filtered using the **Filter stripe** which opens by invoking the **Filter action** in results context menu or using the **Ctrl-G** keyboard shortcut. Values of the Name column are filtered by the entered substring depending on the selected filter mode (**Contains**, **Does Not Contain**, **Regular Expression**), optionally matching case if selected. When collecting the allocation stack traces, the top level class nodes are always displayed, only the allocating methods are filtered.

## 9.19.2 Advanced Profiling

To cover various use cases of objects profiling the profiler offers three profiling modes with different profiling techniques and settings. The modes and settings can be changed in the Settings pane which is displayed by pressing the **Settings** switch in Profiler window toolbar.

### 9.19.2.1 Objects Profiling Modes

[Table 9–9](#) lists objects profiling modes.

**Table 9–9 Objects Profiling Modes**

Name	Description
All classes	Uses sampling and shows histogram of live objects of all classes allocated on the heap, including instance numbers and sizes.
Project classes	Uses sampling and shows histogram of live objects of project classes allocated on the heap, including instance numbers and sizes.
Selected classes	Uses bytecode instrumentation and shows allocated or live instances of the defined classes including allocations stack traces if configured. To use this mode, at least one class has to be selected for profiling.

### 9.19.2.2 Selecting Classes For Profiling

Selected classes instruct the profiler to track instances of these classes only, keeping the profiling overhead low. You can select a class or method for instrumented profiling by either:

- Using the **Select Class** dialog. The dialog allows to select one or several classes from a project or .class/.jar file. It can be open directly from the Objects Settings pane.
- From the code editor. Java code editor provides Profile > Profile Class action in its context menu.
- From Navigator. Java Navigator provides Profile > Profile Class action in its context menu.
- From profiling results. Live or saved Objects results provide Profile Class actions in its context menu. Also a special column Selected is available to select classes for profiling. The column can be displayed using the right corner button in the view header by clicking the drop-down arrow (or right-clicking the header on Mac OS X).

### 9.19.2.3 Configuring Additional Options

Based on the additional settings, the profiler can track either all allocated objects or just the objects currently live on the heap. It can also record allocation stack traces to visualize from where in the source code are the objects being created. The following options are available:

- Track only live objects controls whether the profiler tracks all allocated objects from the beginning of the profiling session or last results reset (deselected) or whether the profiler tracks just the objects currently live in the heap memory (selected). When tracking only live objects, the profiler also provides a special metric Surviving Generations which helps to easily discover certain types of memory leaks.
- Limit allocations depth controls whether the profiler collects allocation stack traces and sets the depth limit for the stacks. If deselected, the profiler collects full allocation stack traces. If selected, the value controls the maximum depth of allocation stacks. Zero value means no allocation stack traces are being collected.

The Objects profiling mode, selected Classes and additional options can be changed at any time during a running profiling session. The changes are applied by clicking the Apply button on the right side of the settings area.

### 9.19.2.4 Objects Views

In case the Track only live objects option is not selected, the following data columns are available:

- **Name** column shows name of the class with allocated instances.
- **Allocated Bytes** column shows the size of all allocated objects from the beginning of the profiling session or last results reset.
- **Allocated Objects** column shows the number of all allocated objects from the beginning of the profiling session or last results reset.

In case the Track only live objects option is selected, the following data columns are available:

- **Name** column shows name of the class with live instances.

- **Live Bytes** column shows the size of all instances of the class currently live on heap.
- **Live Objects** column shows the number of all instances of the class currently live on heap.
- **Allocated Objects** column shows the number of all allocated objects from the beginning of the profiling session or last results reset.
- **Avg. Age** average object age of all instances of the class currently live on heap measured by the number of survived garbage collections.
- **Surviving Generations** column shows the number of different generations measured by the number of survived garbage collections.

---

**Note:** To provide more control of which methods are to be profiled there is a special mode with fully manual setup of profiled classes and instrumentation filter.

---

#### 9.19.2.5 Expert Mode of Objects Profiling

Bytecode instrumentation is a powerful tool to analyze Java applications performance and memory management. However, the profiling overhead and amount of collected data is heavily dependent on the profiler configuration. Misconfigured settings may cause extreme slowdown of the profiled process or OutOfMemoryErrors being thrown by the NetBeans JVM due to too many data.

By default the profiler offers just a limited subset of customizable options for an instrumented profiling session to prevent the above mentioned problems. While this makes the instrumenting profiler a safe choice for most of the users, it doesn't provide the full power of the profiler engine to expert users. That's why there's a special mode Defined classes which allows detailed definition of the classes to be instrumented.

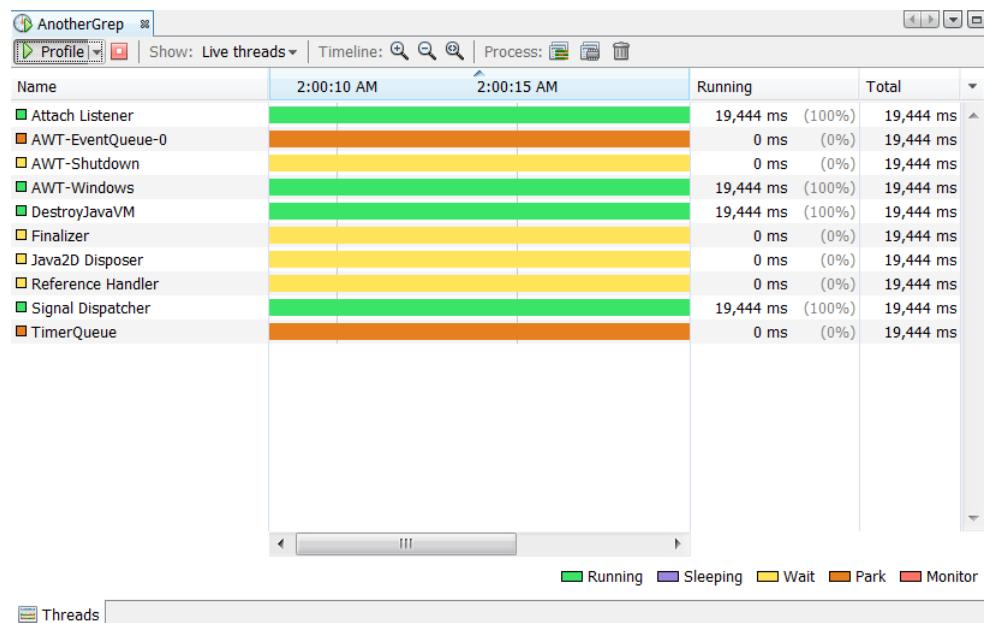
The expert mode needs to be enabled using **Tools > Options > Java > Profiler > General > Enable manual setup for Methods and Objects**. The currently opened profiler window needs to be closed and reopened after changing this option to enable the expert mode. The mode can be set by opening the Settings pane by the rightmost Settings button in the profiler window toolbar and selecting Defined classes.

For more information about defined classes mode of the objects profiling in NetBeans IDE, see <http://wiki.netbeans.org/ProfilerExpertFeatures>.

## 9.20 Profiling Threads

The threads mode allows you to view detailed information about application thread activity.

Figure 9–4 shows a snapshot of a threads session.

**Figure 9–4 Threads Session**

To start a profiling threads session, see [Section 9.9, "Starting a Profiling Session"](#).

For the description of the toolbar buttons see [Section 9.9.3, "Understanding the Toolbar Icons"](#).

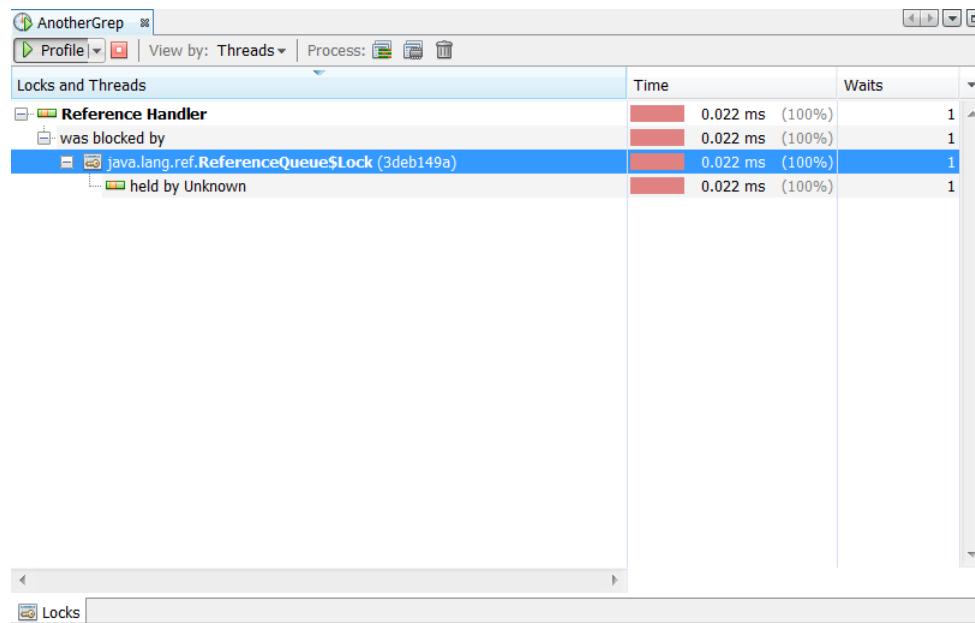
Additionally, you may customize the threads you monitor by accessing the **Live Threads** drop-down list and choosing from the available options: **All Threads**, **Live Threads (Default)**, **Finished Threads**, and **Selected Threads**.

For more information on profiling Threads in NetBeans IDE, see <http://wiki.netbeans.org/ProfilerThreads>.

## 9.21 Profiling Locks

The locks mode allows you to view details about locked threads and the threads that are monitoring and holding locks.

[Figure 9–5](#) shows a snapshot of a locks session.

**Figure 9–5 Locks Session**

The Locks view displays threads and locks of the profiled process and their relation (thread T has been blocked by lock L owned by another thread X) and information about time spent by waiting and number of waits to acquire a lock.

To start a profiling lock session, see [Section 9.9, "Starting a Profiling Session"](#).

For the description of the toolbar buttons see [Section 9.9.3, "Understanding the Toolbar Icons"](#).

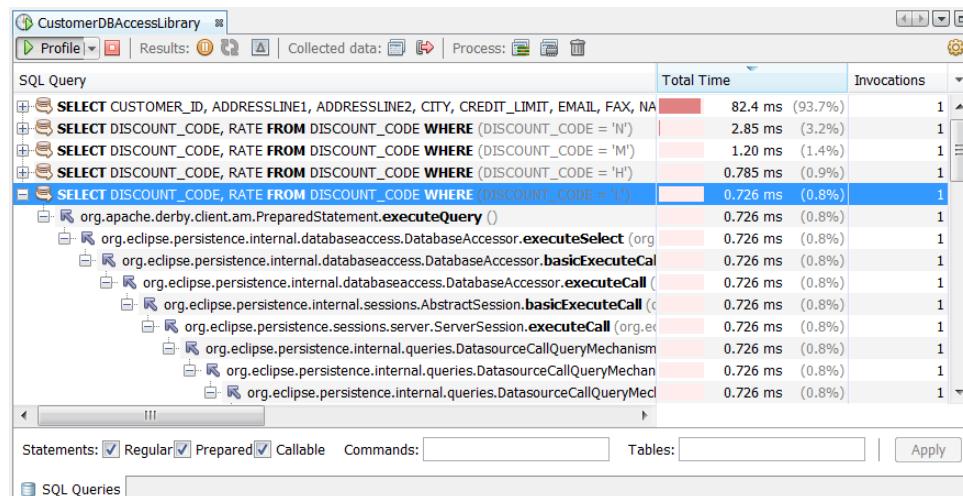
In the session window you can choose **Threads** or **Monitors** in the **Threads** drop-down list. Choose **Threads** to view locked threads. Expand the nodes to view the owners of the locks. Choose **Monitors** to view the threads that are locking other threads.

For more information on profiling locks in NetBeans IDE, see  
<http://wiki.netbeans.org/ProfilerLocks>.

## 9.22 SQL Queries Profiling

SQL queries profiling enables profiling calls from Java processes to databases using a JDBC connection. It allows you to analyze which queries have been invoked, how many times they have been initiated, and how long they took to run. SQL queries profiling helps you to see which SQL query causes the most contention within an application.

[Table 9–6](#) shows a snapshot of a SQL Query session.

**Figure 9–6 SQL Query Session**

The SQL Query view displays a live list of executed SQL queries with their duration and execution counts, including the invocation paths. You can sort the content of each column by clicking a column heading and reorder columns by dragging and dropping column headings.

To start a profiling lock session, see [Section 9.9, "Starting a Profiling Session"](#).

For the description of the toolbar buttons see [Section 9.9.3, "Understanding the Toolbar Icons"](#).

In the **SQL Query** session window you can perform the following actions:

- **View SQL Query.** Displays the text of a query in the SQL Query Viewer window.
- **Copy Row.** Copies the content of the row to the clipboard.
- **Copy SQL Query.** Copies the content of the query to the clipboard.
- **Copy Total Time.** Copies the total time of the query execution to the clipboard.
- **Copy Invocations.** Copies the number of the selected query invocations to the clipboard.
- **Copy Statement Type.** Copies the type of a SQL query to the clipboard.
- **Copy Command Type.** Copies the type of a SQL command to the clipboard.
- **Copy Tables.** Copies the name of a database table to the clipboard.
- **Find.** Searches for the specified query content.

#### To switch between different profiling modes:

1. Click the **Settings** button in the toolbar.
2. Select the required option in the **Profile** drop-down list.
3. If the **Defined queries** option is selected, specify the text that the query must contain.
4. Click **Apply**.

**To filter the collected queries in live results or snapshots:**

1. Specify the filtering options - Statements, Commands, Tables - below the live results area.
2. Click **Apply**.

**To visually filter out profiling results by coloring:**

1. Choose **Tools > Options** from the main menu.
2. In the **Options** dialog box, select **Java > Profiler**.
3. In the **Filters** category, select the **Use defined filters for coloring results** option and click the **Add new filter** button.
4. In the **Add Filter** dialog window, specify the name and value of the new filter.
5. (Optional) Select the **Color** checkbox to define custom color for your filter results.
6. Click **OK**.

**To display a hidden column:**

1. Click the pointing down triangle icon to the right of the table.
2. Select the name of the column to be displayed in the displayed menu.

## 9.23 Additional Functions when Running a Profiling Session

While the profiler session is in progress, additional actions related to the actual profiler mode are available in the toolbar of the profiler window. The following actions are always available:

**Thread dump** — creates a textual dump of all active threads and monitors of the profiled application. It shows what methods have been executed at the point of capturing the dump, thread by thread. This information is useful to view what the application is currently doing. The thread dump also contains information about locks, threads holding the locks, and threads waiting to acquire a lock. This data is essential when debugging deadlocks. To capture a Thread Dump, click the **Thread Dump** icon during the profiling session. To learn more about taking snapshots, see [Appendix 9.14](#).

**Heap dump** — saves an image of the current heap content of the profiled process in .hprof format and optionally opens it in heap browser. For more information, see [Capturing Heap Dump Data](#).

**GC** — requests the JVM of the profiled process to invoke garbage collection. The JVM behavior for garbage collection is not defined in the JVM specification. It should do the garbage collection at some point, but there is no guarantee it will do it immediately or at all.

Additionally, when profiling Methods or Objects, the following actions are available:

**Snapshot** — creates a snapshot of all currently collected profiling data related to methods or objects. The snapshot opens in a separate window and can be saved to the project or to an external file. For more information, see [Taking and Accessing Snapshots of Profiling Data](#).

**Reset collected results** — clears all currently collected profiling data related to methods or objects.

The other actions displayed in the toolbar of the profiler window are specific to the actual profiling mode. If multiple profiling modes are active in a profiling session, the toolbar displays actions available for the currently displayed modes.

# 10

---

## Running and Debugging Java Application Projects

This chapter describes how you can use the running and debugging features in NetBeans to create and perfect your projects.

This chapter contains the following sections:

- [About Running Java Application Projects](#)
- [Working with Project Execution](#)
- [Running an Application](#)
- [Running a Single File](#)
- [Setting the Runtime Classpath](#)
- [Setting the Main Class and Runtime Arguments](#)
- [Setting JVM Arguments](#)
- [Debugging Applications](#)
- [Using the Debugger Windows](#)

### 10.1 About Running Java Application Projects

As you are developing your application, you can run the application in the IDE to test the application's behavior. When you run a project in the IDE, the IDE runs the application from the files in the project's build/classes folder.

Typically, the project that contains the program's main class is set as the main project. You can then run the entire application with the **Run Main Project** command (F6). You can also run any executable class by choosing **Run > Run File > Run my\_class** (Shift+F6). Alternatively, you can run any project that has a main class by right-clicking its project node in the Projects window and choosing **Run Project**.

When you run the project the IDE displays any compilation errors and output in the Output window.

---

**Note:** If Compile on Save is enabled for a project, the **Run Project** command operates on class files that have been created when you have saved those files. The Ant build script is not used. If you have defined custom steps in the build script, those steps are not followed. If you would like the full build process to occur when you use **Run Project**, **Debug Project**, and **Profile Project**, disable Compile on Save. The Compile on Save feature can be toggled from the **Run** category in the **Project Properties** window.

---

### 10.1.1 Running Standard Projects

For standard projects, the IDE uses settings that you specify in project's Project Properties dialog box. You can set the project's main class, runtime arguments, VM arguments, and working directory.

To run the application outside of the IDE, you must first use the **Clean and Build** command so that the project's JAR file is built or updated. For standard projects that have a main class specified, the IDE automatically copies any JAR files on the project's classpath to the dist/lib folder. The IDE also adds each of the JAR files to the Class-Path element in the application JAR's manifest .mf file. This simplifies running the application outside the IDE. For more information, see [Section 8.9, "Preparing a JAR File for Deployment Outside the IDE."](#)

### 10.1.2 Running Free-form Projects

For free-form projects, the IDE uses an existing Ant script to run your class. You can write a target that executes the currently selected file in the IDE and map it to the **Run File** command.

## 10.2 Working with Project Execution

Once you have created and built your project, you must perform the following operations to configure the run process for your project:

- Set your project as the main project in the IDE.
- Set the project's main class.
- Modify the runtime classpath, as needed, to include any special libraries, project dependencies, or specific JAR files.
- Specify any runtime and Java VM arguments the project requires.

When you have the runtime configurations set, you can either run individual project files or run the entire the project. Each of these operations is discussed in more detail in the following sections.

## 10.3 Running an Application

With standard projects, you typically have one project that contains the application main class and several projects containing class libraries. You can run the project or run any individual project that contains an executable class. For information on setting the main class, see [Section 10.6, "Setting the Main Class and Runtime Arguments."](#)

#### To run a project:

1. Select the project that you want to run in the Projects window.

**2. Choose **Run > Run Project (F6)**.**

When running an application, be aware of the following considerations:

- If you have the **Compile on Save** option selected in the Compiling section of the Project Properties window, the project is run directly from the build/classes directory of your project. If **Compile on Save** is not selected, the project is run using the project's build script.
- If you run a project for which you have not specified a main class, the IDE prompts you for the main class. You can change this setting in the Run panel of the project's Project Properties dialog box. Each project can contain only one main class. For information on setting the classpath, see [Section 10.5, "Setting the Runtime Classpath."](#)
- If you are running a project often, you can set a project as the main project by choosing **Run > Set Main Project** from the main menu and selecting the project in the submenu or by right-clicking the project node in the Projects window and choosing **Set as Main Project**.
- You can select multiple projects in the Projects window and run them at once by choosing **Run > Run (number of selected projects) Projects (F6)** from the main IDE's menu.

## 10.4 Running a Single File

You might have a file in your project that you want to test before running the entire project. You can run a single file in the IDE and see any warning or error messages that might occur during the run in the Output window.

**To run a single file:**

1. Select the file in the Source Editor or Projects window.
2. Choose **Run > Run File > Run my\_class**.

The IDE initiates execution of the selected file.

---

**Note:** In free-form projects, this command is disabled by default. To enable this function, you have to write an Ant target for running the file in the IDE and map it to the IDE's **Run Class** command. This command is not available for EJB projects.

---

### 10.4.1 Writing a Target to Run/Debug/Test a Single File

The IDE does not generate targets for the Run File, Debug File, Test File, and Debug Test for File commands, but you can create your own targets and map them to the following predefined actions:

- `run.single` - Run selected file
- `debug.single` - Debug selected file
- `test.single` - Run the JUnit test for selected file
- `debug.test.single` - Debug the JUnit test for selected file

Each of these actions contains a context element that gets a reference to the currently selected files and stores it in a property of your choice. You use this property in your Ant targets to specify which files to process.

### 10.4.1.1 Running the Selected File

Let's demonstrate how this works when you run a class. A typical target for running a project looks something like the following:

```
<target name="run2" depends="... ">
    <java fork="true" classname="MyMainClass" classpath="MyRunClasspath" />
</target>
```

The target runs the file specified by `classname`. To run the currently selected file in the IDE, you need to modify the above target to something like the following:

```
<target name="run-selected-file" depends="compile" description="Run Single File">
    <fail unless="runclass">Must set property 'classname'</fail>
    <java classname="${runclass}">
        <classpath refid="run.classpath" />
    </java>
</target>
```

### 10.4.1.2 Getting a Reference to the Currently Selected File in the IDE

Once you have an Ant target for running the selected file, you have to get a reference to that file in the IDE and store it in a property. For example, the `run-selected-file` target above looks for the currently selected file in the `runclass` property.

You store this reference in the same place where you map the build target (`run-selected-file`) to the IDE action. First we will look at how to do this and then we will explain it in detail:

```
<action name="run.single">
    <target>run-single</target>
    <context>
        <property>runclass</property>
        <folder>${src.dir}</folder>
        <pattern>\.java$</pattern>
        <format>java-name</format>
        <arity>
            <one-file-only/>
        </arity>
    </context>
</action>
```

The `runclass` property is a newly defined property that holds the file that you want to run and is referenced by the `java` task.

Now let's take a look at the following lines to see how it works.

```
<action name="run.single">
    <target>run-selected-file</target>
    <context>
        <property>runclass</property>
    </context>
    <action name="run.single"> maps the Run File command and the F9 shortcut to
    the run-selected-file target.
    <action name="run.selected"> maps the Run Selected File command and the F8
    shortcut to the run-selected-file target.
    <context> sets the context on which the Ant target is executed. In this case, it is
    the name of file that you want to run.
    runclass is the name of the property that holds the context. You can choose any
    unique name for this property. This property must be set by the IDE before the
    target can be run.
    <arity> specifies that runclass can hold only one file. If you want the property to
    be able to hold more than one file (such as for the Compile File target), you can use
    the following, where the comma (,) is the separator between file names:
    <arity>
```

- ```

        <separated-files>, </separated-files>
    </arity>
■ <format>java-name</format> specifies that the IDE should pass the relative file name to the target but delimited by periods (.) and without an extension. Other formatting options include the following:
    - relative-path - specifies that the IDE should pass the relative file name to the target
    - relative-path-noext - Same as relative-path, but the file's extension is removed
    - absolute-path - Absolute file name
    - absolute-path-noext - Same as absolute-path, but the file's extension is removed
■ <folder>${src.dir}</folder> specifies that the file name should be relative to the src.dir directory and that this action is only enabled for the src.dir directory.

```

---

**Note:** The IDE does not define the \${src.dir} property for you. You have to define the property or import the .properties file that the Ant is using in project.xml. See Using Properties in the project.xml File for more information.

---

- <pattern>\.java\$</pattern> is the regular expression which the file names must pass. You use <pattern> to limit which files can be passed to the Ant target. In this case, you want the target be executed only with files that end in .java.

#### 10.4.1.3 Debugging the Selected File

The process is basically the same for writing targets to debug and run a single file. The debug-selected-files target looks something like this:

```

<target name="debug-selected-files" depends="compile" if="netbeans.home"
description="Debug a Single File">
    <fail unless="classname">Must set property 'classname'</fail>
    <nbjpdastart name="${classname}" addressproperty="jpda.address"
transport="dt_socket">
        <classpath refid="run.classpath"/>
        <!-- Optional - If source roots are properly declared in project, should
work without setting source path.
        <sourcepath refid="debug.sourcepath"/> -->
    </nbjpdastart>
    <java classname="${classname}" fork="true">
        <jvmarg value="-Xdebug"/>
        <jvmarg value="-Xnoagent"/>
        <jvmarg value="-Djava.compiler=none"/>
        <jvmarg value="-Xrunjdwp:transport=dt_socket,address=${jpda.address}"/>
        <classpath refid="run.classpath"/>
    </java>
</target>

```

- This is basically the same as the debug target. Instead of passing the program main class to java, you pass the classname property, which is set by the IDE to the currently selected file.

Then you map the debug-selected-files target to the debug.single action:

```

<action name="debug.single">
```

```

<target>debug-selected-files</target>
<context>
  <property>classname</property>
  <folder>${src.dir}</folder>
  <pattern>\.java$</pattern>
  <format>java-name</format>
  <arity>
    <one-file-only/>
  </arity>
</context>
</action>


- <property> now stores the context in the classname property.
- Since java can only take single file, you set <arity> to <one-file-only>.
- Setting <format> to java-name and making it relative to src.dir creates a fully-qualified class name for the currently selected file.

```

---

**Note:** The IDE does not define the  `${src.dir}` property for you. You have to define the property or import the .properties file that the Ant is using in project.xml. See [Section 6.2.4.9.1, "Using Properties in the project.xml File"](#) for more information.

---

## 10.5 Setting the Runtime Classpath

By default, the runtime classpath of each standard project contains the project's compiled classes and everything in the project's compilation classpath. For information on viewing the compilation classpath, see [Section 6.2.3.1, "Managing the Classpath."](#)

If your project uses special libraries dynamically at runtime through an indirect interface or reflection (like JDBC drivers or JAXP implementations), you have to add these libraries to the runtime classpath.

---

**Note:** For standard projects that have a main class is specified, the IDE automatically copies any JAR files on the project's classpath to the `dist/lib` folder. The IDE also adds each of the JAR files to the `Class-Path` element in the application JAR's `manifest.mf` file. This simplifies running the application outside the IDE. For more information, see [Section 8.9, "Preparing a JAR File for Deployment Outside the IDE."](#)

---

You also have to adjust your runtime classpath if the runtime dependencies between your projects do not match the compilation dependencies between the projects. For example, imagine that project A compiles against project B, and project B compiles against project C, but project A does not compile against project C. This means that project A only has project B on its runtime classpath. If project A requires both project B and project C during execution, you have to add project C to project A's runtime classpath.

**To set the runtime classpath:**

1. Right-click the project node in the Projects window and choose **Properties**.
2. In the Project Properties dialog box, select the Libraries node in the Categories pane.

3. Select the Run tab in the dialog's right pane.
4. Add the necessary elements to the project's runtime classpath by clicking the appropriate button. You can add any of the following:
  - **Project.** The build output, source files, and Javadoc files of another IDE project. Adding a project to the classpath makes it dependent on the present project. Whenever you clean or build the project, all of its dependent projects are also cleaned or built.
  - **Library.** A collection of binary files, source files, and Javadoc files.
  - **JAR/Folder.** A JAR file or folder somewhere on your system.
5. (Optional) Click **Move Up** and **Move Down** to alter to the classpath priority.

---

**Note:** In free-form projects, your Ant script handles the classpath for all of your source folders. The classpath settings for free-form projects only tell the IDE what classes to make available for code completion and refactoring. For more information, see [Section 6.2.4.5, "Declaring the Classpath in a Free-Form Project."](#)

---

## 10.6 Setting the Main Class and Runtime Arguments

By default, the IDE specifies neither a main class nor runtime arguments. The runtime classpath of each standard project contains the project's compiled classes and everything in the project's compilation classpath.

Indicate which class in your project is the entry point for the application by setting a main class.

**To set the main class and runtime arguments:**

1. Right-click the project node in the Projects window and choose **Project Properties**.
2. Select the **Run** node in the **Categories** pane of the dialog box.
3. Type the fully qualified name of the main class in the **Main Class** field (for example, `org.myCompany.myLib.MyLibClass`). The main class must exist in the project or in one of the JAR files or libraries on the project's runtime classpath.

---

**Note:** If you use the Browse button to choose the project main class, the file chooser only displays classes in your project source directory. If you want to specify a class in one the libraries on the classpath, you have to type the fully-qualified name of the class in the Main Class field.

---

4. Enter any runtime arguments you require in the **Arguments** field.

The IDE sets the project's main class and stores any newly-added arguments.

---

**Note:** For standard projects that have a main class specified, the IDE automatically copies any JAR files on the project's classpath to the dist/lib folder. The IDE also adds each of the JAR files to the Class-Path element in the application JAR's manifest.mf file. This simplifies running the application outside the IDE. For more information, see [Section 8.9, "Preparing a JAR File for Deployment Outside the IDE."](#)

---

If you use the **Browse** button to choose the project main class, the file chooser only shows classes in your project source directory. If you want to specify a class in one of the libraries on the classpath, you have to type the fully-qualified name of the class in the **Main Class** field.

**To change project runtime options:**

1. Right-click the project node in the **Projects** window and choose **Properties**.
2. In the **Project Properties** dialog box, select the **Libraries** node in the **Categories** pane.
3. Click the **Run** tab in the right pane of the dialog box.

---

**Note:** To access settings for the main class, program arguments, the working directory for program execution and VM options, you have to select the **Run** node.

---

## 10.7 Setting JVM Arguments

JVM arguments and system properties can be set through the project's properties file through the IDE.

**To set JVM arguments:**

1. Right-click the project node in the Projects window and choose **Project Properties**.
2. Select the Run node in the Categories pane of the dialog box.
3. Type a space-separated list of JVM arguments in the VM Options field.

**To set system properties:**

- Specify the system property and its value in the VM Options field:  
`-Dname=value`

## 10.8 Debugging Applications

Debugging is the process of examining your application for errors. The process of debugging is accomplished by setting breakpoints and watches in your code and running it in the debugger. This enables you to execute your code one line at a time and examine the state of your application to discover any problems.

When you start a debugging session the Debugging window opens in the left pane of the IDE. Additional debugger windows also appear automatically at the bottom of your screen.

You can also debug applications that are running on a remote machine by attaching the debugger to the application process.

You can customize the Java debugger by choosing **Tools > Options** and clicking the **Java Debugger** tab. From this tab you can perform the following:

- set custom breakpoint commands
- set step filters to specify language constructs that are filtered while stepping
- create and edit variable formatters
- modify options for the Visual Debugger

### 10.8.1 Debugging Free-form Projects

Similar to commands for compiling and running, debugging commands rely on various information, such as the location of your sources, the location of the compiled classes and other items on the classpath, and name of the project's main class (for more information, see [Section 10.5, "Setting the Runtime Classpath"](#)).

In free-form projects, the IDE does not "know" about any of these things. When you run a command in the IDE (such as Build), the IDE simply calls a target in your build script and lets the script handle the command. Therefore, for debugging to work, you also have to have a build script target for debugging. The IDE provides some custom Ant tasks to work with the debugger and also can generate a basic debug target, which attempts to fill in important details based on other targets in your script.

#### To set up debugging in a free-form project:

- Make sure that your classes are compiled with debugging information included. For example, you might accomplish this in the compile target of your build script by including the argument `debug="true"` in the `<javac>` task.
- Set the output of the free-form project. If the output of a free-form project is on the classpath of another project, map the free-form project's source packages to their outputs. This ensures that you can use the debugger to step into the project's sources when you start a debugging session in a project that has a dependency on the free-form project. To declare the output files, right-click the free-form project node and choose **Properties**. Then click the **Output** category in the Properties window and specify the output file for each source folder.
- Confirm that the target JDK is set in your Ant script and the source level is specified in the Project Properties dialog box. When you step into JDK classes, the IDE searches the platforms registered in the Java Platform Manager for a Java platform with a matching source level. If no matching Java platform is found, the IDE opens the source code for the IDE's default platform.
- Create a target in your build script for debugging and map that target to the IDE's Debug Project command. The IDE can assist you by generating a basic target and mapping, but you might have to modify the target. For more information, see [Section 10.8.1.1, "How to Create a Debug Target for a Free-form Java Project."](#)

For information on mapping an Ant target to a debugging command, see [Section 6.2.4.6, "Mapping an Ant Target to an IDE Command."](#)

#### 10.8.1.1 How to Create a Debug Target for a Free-form Java Project

To run a free-form project in the IDE's debugger, you have to have a special target in your project's build script. That target needs to be mapped to the IDE's Debug Project command.

If you do not have a debug target written for your project, the IDE offers to generate a basic target for you when you first try to debug the project. You can then inspect the target and customize it for the project's specific requirements.

---

**Note:** When the IDE generates a debug target, it looks for information in the target you have mapped to the **Run Project** command to determine such things such as the run classpath and the project's main class. If you have a target mapped to the **Run Project** command, there is a good chance that the generated debug target works without further customization.

---

#### To create a debug target for a free-form project:

1. Set the free-form project as the main project by choosing **Run > Set Main Project** in the main menu and selecting the project.
2. Choose **Debug > Debug Main Project** from the main menu.
3. Click **Generate** in the Debug Project dialog box.

When you click **Generate**, a target named debug-nb is created in a file named ide-targets.xml. The generated ide-targets.xml file is a build script that imports your main build.xml file, so your debug target can take advantage of targets and properties set by or referenced by your main build script.

In addition, a mapping for this target is created in the project.xml file so that the target is called whenever you choose the **Debug Project** command in the IDE. If you write the target from scratch, you must also create this mapping yourself. For more information, see [Section 10.8.1.1.2, "Manually Mapping a Target to a Menu Item."](#)

4. Verify that the generated debug-nb target properly takes into account all of the elements of your project.

In particular, you might have to modify the <classpath> argument in the target if it does not include all of the items in your run classpath.

After the target is created, you can begin debugging the project.

#### To debug the project:

1. Set a breakpoint in your main class by clicking in the left margin of the line where you want to set the breakpoint.

The line with the breakpoint is highlighted in pink.

2. Right-click the project's node again and choose **Debug Project**.

The target should run and start execution of the program. Progress of the running target is shown in the Output window and the status of the debugger is shown in the status bar at the bottom of the Output window.

##### 10.8.1.1.1 A Typical Free-form Project Debug Target

The generated Ant target does the following:

- Starts the debugger with the nbjpdastart task.
- Stores the address at which the debugger listens for the application in the jpda.address property (`addressproperty="jpda.address"`). You do not have to define the jpda.address property in your Ant script or properties file. It is defined by the IDE.

- Establishes the runtime classpath. If the IDE is not able to determine your runtime classpath, placeholders are put in the script, which you must fill in yourself.
- Runs the application in debug mode, passing the `jlda.address` property as the address at which to connect to the debugger. Setting `fork="true"` ensures the process is launched in a separate virtual machine.

---

**Note:** You can add any additional JVM arguments or program arguments in the java task as well.

---

For example, if the IDE is able to guess the runtime classpath, the `debug-nb` target that the IDE generates in `ide-targets.xml` might look similar to [Example 10-1](#):

**Example 10-1 Defining a Free-form Debug Target**

```
<?xml version="1.0" encoding="UTF-8"?>
<project basedir=".." name="YourProjectName">
    <import file="../build.xml"/>
    <!-- TODO: edit the following target according to your needs -->
    <!-- (more info: http://www.netbeans.org/kb/articles/freeform-config.html#debugj2se) -->
    <target name="debug-nb" depends="init,compile">
        <nbjpdastart addressproperty="jlda.address" name="NameOfProject" transport="dt_socket">
            <classpath path="ClasspathSpecifiedInYourRunTarget"/>
        </nbjpdastart>
        <java classname="MainClassSpecifiedInRunTarget"
classpath="ClasspathSpecifiedInYourRunTarget" fork="true">
            <jvmarg value="-Xdebug"/>
            <jvmarg value="-Xrunjdwp:transport=dt_socket,address=${jlda.address}"/>
        </java>
    </target>
</project>
```

If you do not have a run target mapped or the IDE otherwise cannot determine the project's classpath or main class, the generated debug target includes "TODO" placeholders for you to fill in these values as shown in [Example 10-2](#).

**Example 10-2 Generated debug target with "TODO" placeholders**

```
<?xml version="1.0" encoding="UTF-8"?>
<project basedir=".." name="YourProjectName">
    <!-- TODO: edit the following target according to your needs -->
    <!-- (more info: https://netbeans.org/kb/articles/freeform-config.html) -->
    <target name="debug-nb">
        <path id="cp">
            <!-- TODO configure the runtime classpath for your project here: -->
        </path>
        <nbjpdastart addressproperty="jlda.address" name="NameOfProject" transport="dt_socket">
            <classpath refid="cp"/>
        </nbjpdastart>
        <!-- TODO configure the main class for your project here: -->
        <java classname="some.main.Class" fork="true">
            <classpath refid="cp"/>
            <jvmarg value="-Xdebug"/>
            <jvmarg value="-Xnoagent"/>
            <jvmarg value="-Djava.compiler=none"/>
            <jvmarg value="-Xrunjdwp:transport=dt_socket,address=${jlda.address}"/>
        </java>
    </target>
```

```
</project>
```

To specify the runtime classpath, insert `pathelement` elements within the `path` element and point them to the directories that contain the items in your classpath. For example, you can use the `location` attribute of `pathelement` to specify the location of the classpath items relative to your project directory as shown in [Example 10–3](#). The project directory is usually the one that contains the project's `build.xml` file.

**Example 10–3 Specifying location of classpath items relative to a project directory**

```
<path id="cp">
    <pathelement location="libs"/>
    <pathelement location="build"/>
</path>
```

**10.8.1.1.2 Manually Mapping a Target to a Menu Item** When you have the IDE generate a target, the IDE automatically provides the mapping between the target and the IDE command's menu item. However, if you have created the target manually, you must also create the mapping manually.

**To map the Debug Project command to a target in an external Ant script:**

1. Open the project's `project.xml` file and add the following to `ide-actions`:

```
<action name="debug">
    <script>path_to_Ant_script</script>
    <target>target_name</target>
</action>
```

2. Add the command to the project node's context menu, by adding the following line to the `<context-menu>` target:

```
<ide-action name="debug" />
```

The IDE maps the Debug Project action to the specified target in the project's Ant script. For information on how to map a target to a debugging command, see [Section 6.2.4.6, "Mapping an Ant Target to an IDE Command."](#)

**10.8.1.1.3 Troubleshooting** If you have successfully created a debug target and started the debugger, but the debugger does not stop at breakpoints, the IDE is probably lacking debugging information or knowledge of where your sources are. For more information, see [Section 10.8.1, "Debugging Free-form Projects."](#)

**10.8.1.2 How to Create a Debug Target for a Free-form Web Project**

In a free-form project, you must set up your own Ant target to run a project in the debugger. However, you can use the IDE to generate a debug target for you. When you do so, the IDE maps the debug target to the Debug Project command. Alternatively, if you have your own debug target, you must map it to the Debug Project command yourself.

**To generate a debug target:**

1. Set the project as the main project and choose **Debug > Debug Main Project** (Ctrl+F5) from the main menu or right-click the project in the Projects window and choose **Debug**.

If no target is mapped to the Debug Project command, you are prompted to let the IDE generate an IDE-specific debug target in `nbproject/ide-targets.xml`.

2. Click **Generate**.

The IDE does the following:

- Generates a target named debug-nb. The target is generated in the new nbproject/ide-targets.xml file, together with other targets that support it, such as the -load-props, -check-props, -init, and debug-display-browser targets. These targets do the following:
  - Checks whether Ant is running inside the IDE (`if="netbeans.home"`).
  - Starts the debugger with the nbjpdaconnect task.
  - Connects the debugger to the application to be debugged at the specified host (`jpda.host`) and port number (`jpda.address`).
  - Opens the IDE's web browser at the URL specified by the `client.url` property.

**Note:** These targets are not generated in the IDE. Therefore, you need to change the generated code so that it looks as follows:

```
<target name="-load-props">
    <property file="nbproject/project.properties"/>
</target>

<target name="-check-props">
    <fail unless="session.name"/>
    <fail unless="jpda.host"/>
    <fail unless="jpda.address"/>
    <fail unless="jpda.transport"/>
    <fail unless="web.docbase.dir"/>
    <fail unless="debug.sourcepath"/>
    <fail unless="client.url"/>
</target>

<target depends="-load-props, -check-props" name="-init"/>

<target depends="-init" name="debug-nb" description="Debug
Project">
    <njpdaconnect address="${jpda.address}" host="${jpda.host}"
    name="${session.name}" transport="${jpda.transport}">
        <sourcepath>
            <path path="${debug.sourcepath}" />
        </sourcepath>
    </njpdaconnect>
    <antcall target="debug-display-browser" />
</target>

<target name="debug-display-browser">
    <nbrowse url="${client.url}" />
</target>
```

There's no need for you to customize the generated targets. All you have to do is set the properties that the IDE requires to use the targets it generated. For example, you need to tell the IDE where your application's sources are. To do this, you will set properties in the nbproject/debug.properties file that the IDE created for you when it generated the debug-nb target above. Using the -load-props target above, the IDE will load the properties when you run the debug-nb target.

- Generates and defines debug properties. The properties are defined in the new nbproject/debug.properties file. The properties define the following:
    - The sources that are to be debugged.
    - The server to which the application to be debugged is to be deployed.
    - The port number and address to be used.
    - The client URL.
  - Maps the debug-nb target to the **Debug Project** command.
3. In the **Files** window, go to the nbproject/debug.properties file and edit the debug properties as shown in [Example 10–3](#), if necessary.

**Example 10–4 Adding properties to a new debug.properties file**

```

jpda.session.name=MyProject
jpda.host=localhost

# Sun Java System Application Server using shared memory (on Windows)
# jpda.address=localhost4848
# jpda.transport=dt_shmem

# Sun Java System Application Server using a socket
# jpda.address=9009
# jpda.transport=dt_socket

# Tomcat using shared memory (on Windows)
jpda.address=tomcat_shared_memory_id
jpda.transport=dt_shmem

# Tomcat using a socket
#jpda.address=11555
#jpda.transport=dt_socket

src.folders=src
web.docbase.dir=web

# you can change this property to a list of your source folders
debug.sourcepath=${src.folders}:${web.docbase.dir}

# Client URL for Tomcat
client.url=http://localhost:8084/MyProject

# Client URL for Sun Java System Application Server
# client.url=http://localhost:8080

```

Property	Value	Notes
jpda.session.name		The display name given in the Sessions window when you debug the project.
jpda.host		The host that the application to be debugged uses to connect to the debugger, such as localhost.

Property	Value	Notes
jpda.address	The bundled Tomcat Web Server defaults are 11555 for socket connections and tomcat_shared_memory_id for shared memory connections.	To set a different address, right-click the Tomcat node in the Services window and choose Properties. In the Properties sheet, change the Debugging Port property (for socket connections) or Name property (for shared memory connections). Then close the Properties sheet. Now stop and restart the Tomcat Web Server, if you had already started it.
jpda.transport	dt_socket (for socket connections) or shmem (for shared memory connections).	To set a different transport, right-click the Tomcat node in the Services window and choose Properties. In the Properties sheet, change the Debugging Type. Then close the Properties sheet. Now stop and restart the Tomcat Web Server, if you had already started it.
web.docbase.dir src.folders	The location of your web root (web.docbase.dir) and Java source files (src.folders).	Multiple source roots can be included in the sourcopath by means of the ":" delimiter. Note that the Java source folders must be specified as Source Package Folders in the Java Sources panel of the Project Properties dialog box. (Right click the project, choose Properties, then click Java Sources in the Project Properties dialog box.)
client.url		The URL that should be opened in the IDE's default browser, such as <a href="http://localhost:8084/MyProject">http://localhost:8084/MyProject</a> .

If you use the IDE to generate the debug target, as described in the previous section, the target is automatically mapped to the **Debug Project** command. However, if your debug target was not generated by the IDE, you must map it to the **Debug Project** command manually.

#### To map a debug target to the **Debug Project** command:

1. In the Projects window, right-click the project node and choose **Properties**.
2. Click **Build and Run** in the left panel of the Project Properties dialog box.
3. Click **Add**, select your debug target, and type a label, such as "Debug Project."

If you want to map the debug action to a target in a separate Ant script, open the project's `project.xml` file and add the following to `<ide-actions>`:

```
<action name="debug">
  <script>path_to_Ant_script</script>
  <target>name_of_target</target>
</action>
```

To add the command to the project node's contextual menu, add the following to `<context-menu>`:

```
<ide-action name="debug" />
```

#### 10.8.1.2.1 Using the Debug Target

Before you can use your debug target, you need to deploy your application. Therefore, start the server and run deploy the application.

Note that the first time that you run the application per session, the Tomcat Web Server asks you for a username and password. The only acceptable username and password is that of a user with a "manager" role. This is defined in the `conf/tomcat-users.xml` file in the Tomcat Web Server's base directory. To identify the location of this directory, right-click the Tomcat Web Server instance node in the Services window and select Properties. In the Properties dialog box, the Base Directory property points to the Tomcat Web Server's base directory.

Once the application is deployed, stop the server and restart it in debug mode. The way this is done depends on the server:

- **Bundled Tomcat Web Server**  
Expand the Servers node in the Services window, right-click the Bundled Tomcat node, choose Start/Stop Server, and click Start Server (Debug).
- **External Tomcat Web Server**  
Run the `catalina jpda start` command.

Once the server has started in debug mode, choose **Run > Debug Main Project**. The application is deployed and is attached to the debugger. The debugger stops at the first breakpoint, after which you can **Step into** or **Step over** the code.

#### To use a debug target in a free-form web project:

1. Set breakpoints in your source files.
2. Right-click the project node in the Projects window, choose **Properties**, click **Java Sources** in the Project Properties dialog box, and make sure that all the source files you want to debug are listed in the Source Package Folders list.
3. In the Services window, expand the Servers node, right-click the server instance and choose **Start/Stop Server**.
4. Click **Start Server (Debug)**.
5. Make sure that you have a debug target and that it is mapped to the Debug command, as described in the previous sections.
6. Choose **Debug > Debug Main Project** (Ctrl+F5).

For more information debugging features that you can use for web applications in the IDE, see [Section 12.12, "Debugging a Web Application."](#)

**10.8.1.2.2 Troubleshooting the Debug Target** Even though the IDE does its best to generate a complete debug target for you, with properties that are tailored to your specific environment, you should always analyze and fine tune the debug process. Work through the questions below when you encounter problems while using an Ant debug target from NetBeans IDE:

- Has the web application been correctly deployed?

Check that the web application has been deployed:

1. In the Services window, expand the Servers node, start the server (if not started), expand the server's instance node, and expand the Web Applications node.
  2. If you do not see your application's context (/MyProject, for the application in this document), it has not been correctly deployed.
  3. Deploy the application.
- Are you behind a firewall?

Check that your proxy settings are correct. Depending on your proxy type do the following:

- **HTTP Proxy.** Choose Tools > Setup Wizard. In the wizard, select the Use HTTP Proxy Server checkbox. Type the proxy host name in the Proxy Server Name field and the port number in the Port field. Click Finish.
- **SOCKS Proxy.** You must pass the SOCKS proxy host and proxy port parameters to the JVM software when you start the IDE. On Microsoft Windows machines, use the IDE-HOME/etc/netbeans.conf file to pass the parameters. On UNIX and Linux machines, you can write a wrapper shell script. Go to Help > Help Contents for details.

■ Is the server running in debug mode?

Check that the server has been started in debug mode:

1. In the Services window, expand the Servers node and check that the server is running. Note that even if it is running, it may not be running in debug mode.
2. If it is not running, right-click it, choose Start/Stop Server, and click Start Server (Debug). If it is running, but you are not sure that it is running in debug mode, stop the server and restart it in debug mode.

■ Are the server's port and address set correctly?

Check that the `jpda.address` set in `debug.properties` matches the server's settings:

1. Right-click the server's node in the Services window and choose Properties.
2. In the Properties sheet:
  - Check the Debugging Port property (for socket connections). By default, it should be 9009 for the SJS Application Server or 11555 for the Tomcat Web Server.
  - Check the Name property (for shared memory connections). By default, it should be localhost4848 for the SJS Application Server or `tomcat_shared_memory_id` for the Tomcat Web Server.

If you change the server's Debugging Port property or Name property, make sure that it matches the related property in the `debug.properties` file.

3. Close the Properties sheet and stop and restart the server, if you had already started it.

Check that the `jpda.transport` set in `debug.properties` matches the server's settings:

1. Right-click the server's node in the Services window and choose Properties.
2. In the Properties sheet, check the Debugging Type property:
  - `dt_socket` for socket connections
  - `dt_shmem` for shared memory (Windows)

If you change the server's Debugging Type property, make sure that it matches the related property in the `debug.properties` file.

3. Close the Properties sheet and stop and restart the server, if you had already started it.

■ Unable to step through your code?

If you are unable to step from line to line in your code, but only from breakpoint to breakpoint, the IDE has not been able to find your sources. This is because you have not specified your sources correctly.

- **Servlets:** Choose Window > Debugging > Sources. The Sources window displays all the Java source folders that are available for debugging. If you want to debug a source folder that is not available in the Sources window, specify it in the Project Properties dialog box:
  - \* Right-click the project node, choose Properties, click Java Sources.
  - \* Add the source folders to be debugged to the Source Package Folders table or to the Test Package Folders table.

---

**Note:** The target you use for compiling servlets must specify debug="true" when calling the javac task. If a servlet is compiled without debug info, the debugger will not stop on its breakpoints.

---

- **JSP pages:** Make sure that you have defined a context path for the project:
  - \* Right-click the project node, choose Properties, click Web Sources.
  - \* Type the context path. For example, type /MyProject in the Context Path field.

---

**Note:** If you have set your breakpoints before specifying the context path, you must remove and reset the breakpoints after specifying the context path. In other words, the context path must be set first.

---

Also make sure that the sources are correctly specified in the debug.properties file and in the debug-nb target. Note that if your nbproject folder is not housed within the folder that houses your sources folder, you should set the following properties for your src.folder and web.docbase.folders properties:

- src.folder=\${project.dir}/src
- web.docbase.dir=\${project.dir}/web

## 10.8.2 Debugging GUI Projects

You can use the visual debugger to help you locate and debug the code for visual elements in your Java and JavaFX GUI applications.

### 10.8.2.1 GUI Snapshots

After you create a project you can start a debugging session, take a GUI snapshot of the application, and then work with the snapshot to locate source code, add listeners to events and view the event log of GUI components.

#### To take a GUI snapshot:

1. Click the **Debug** button in the toolbar to start a debugging session.

Alternatively, right-click the project node in the **Projects** window and choose **Debug**.

When you start the session, the IDE will launch your application and open the **Debugging** window.

2. Choose **Debug > Take GUI Snapshot** from the main menu.

When you choose **Take GUI Snapshot**, the IDE takes a snapshot of the GUI and opens the snapshot in the main window.

### 10.8.2.2 Working with the Visual Debugger

The GUI snapshot is a visual debugging tool that can help you locate the source code for GUI components. The source code for GUI components can sometimes be difficult to locate and the snapshot provides a way for you to locate the code based on the GUI instead of searching through the code. You can select components in the snapshot and invoke tasks from the context menu to view the source code for the component, show the listeners and set breakpoints on components.

**10.8.2.2.1 Locating the Source Code for Components** to use the GUI snapshot to navigate to the lines in the source code where a component is declared and defined, you select a component in the GUI snapshot and use the context menu to invoke various commands.

---

**Note:** you select a component in the GUI snapshot, you can use the context menu to invoke various commands.

---

#### To locate the declaration and source code for the component:

1. In the GUI snapshot, select a component (for example, a button).

When you select a component in the snapshot, the IDE displays details about the selected component in the **Properties** window. If the **Properties** window is not visible you can choose **Window > Properties** from the main menu to open the window.

The IDE also displays the location of the component in the form hierarchy in the **Navigator** window.

2. Right-click the component in the snapshot and choose **Go to Component Declaration** from the context menu.

When you choose **Go to Component Declaration**, the IDE opens the source file in the editor and moves the cursor to the line in the code where the component is declared.

3. Right-click the component in the snapshot again and choose **Go to Component Source**.

When you choose **Go to Component Source**, the IDE opens the source file in the editor and moves the cursor to the line in the source code for the component.

#### To locate the line in the source code where a component is added to its container:

1. Open the **Options** window.
2. Click the Java Debugger tab in the Java category in the **Options** window.
3. Select Visual Debugging in the list of categories and select **Track locations of component hierarchy changes**. Click **OK**.
4. Stop your debugging session (if one is running).

---

**Note:** After you enable the command in the **Options** window you will need to restart your debugging session and take a new GUI snapshot before you can use the **Go to Hierarchy Addition** command.

---

5. Start a new debugging session and take a GUI snapshot.
6. Right-click a component in the GUI snapshot and choose **Go to Hierarchy Addition**.

The IDE will open the source code in the editor at the line where the component is added.

#### **10.8.2.2.2 Exploring Component Events** you can use the GUI snapshot and the **Events** window to explore component events.

##### **To locate component listeners and the events that are triggered by the components:**

1. Right-click a component in the snapshot and choose **Show Listeners** from the context menu.  
When you choose **Show Listeners**, the IDE opens the **Events** window with the expanded **Custom Listeners** node.
2. Right-click an item below the **Custom Listeners** node and choose **Go to Component Source** in the context menu.

The source code opens in the editor at the line where the listener is defined.

3. Select another component in the snapshot.

Alternatively, you can select a component in the **Navigator** window.

When you select another component, the items in the **Events** window will change automatically to display the listeners for the selected component.

4. In the **Events** window, double-click the **Event Log** node to open the **Select Listener** window.

Alternatively, you can right-click the **Event Log** node and choose **Set Logging Events** from the context menu.

5. Select a required listener from the dialog (for example, `java.awt.event.KeyListener`). Click **OK**.
6. In your application, complete an event (for example, type a character in a text field).

The event is recorded in the events log. (For example, if you expand the **Event Log** node you can see that each keystroke is now logged). New events appear each time that you complete an event. If you expand an individual event, you can see the properties of that event in the log.

If you expand the **Called From** node for an event you can see the stack trace for the event.

#### **10.8.2.3 How to Configure Java Debugger Options**

You can configure the options for debugging Java applications in the Options window.

**To configure Java debugging options:**

1. Open the Options window by choosing **Tools > Options** from the main menu. (If you are running on Mac OS X, choose NetBeans > Preferences.)
2. Select the Java Debugger tab in the Java category of the Options window.
3. Select a category in the Java Debugger tab to configure the global settings for the behavior of the Java debugger.
4. Click **apply** in the Options window to apply the changes.

## 10.9 Using the Debugger Windows

When you start a debugging session the IDE opens some debugging windows by default. In addition to the main Debugging window in the left pane of the IDE, other debugger windows open as tabs below the editor. You can open any debugger window by choosing **Window > Debugging > window-name** (for example, **Window > Debugging > Breakpoints**).

Each debugger window displays a variety of icons to relay information about the object. For example, the Breakpoints window uses a small red square to indicate a breakpoint set on a line. Some windows also include a node expansion control to the left of the icon. Clicking this control expands and collapses the object.

The Debugging window opens in the left pane of the IDE and uses a tree structure to display the threads and calls in the current debugging session. The current thread and call are displayed in bold. You can expand the node for suspended threads to view the call stack.

In the debugger tabs, information is organized into lists. Each list item represents a single object. Each column represents a property of the object. Data displayed in blue underlined text is linked to the source code.

Some elements of lists in the debugger tabs have editable properties, such as the value property of a variable in the Variables window. If you select a property and the property has a white background you can edit the property. A selected property with a gray background cannot be edited.

### 10.9.1 Customizing a Debugger Window

You can rearrange elements of a debugger window or remove columns to display only the information of interest.

**To add or remove a column to a window:**

1. Click  (to the right of the column titles) or right-click in the window and choose **List Options > Change Visible Columns**.
2. Click the appropriate checkbox to turn the display of information on or off.
3. Click **OK**.

**To rearrange the columns in a window:**

- Drag the column header to the right or left to place the column in the new location.

**To set the sort order of a window:**

- Right-click anywhere in the window and choose **List Options > Sort > sort-order**.

For all windows, you can sort the column in ascending or descending order. A triangle is displayed in the column header. The direction in which the triangle is pointing indicates whether the sort order is ascending or descending. Some windows provide additional sort orders.

## 10.9.2 Choosing Current Context in the Debugger

The current context is the portion of your program on which the debugger is currently focusing. When multiple sessions are running, only one session is current. Within the current session, the thread from which the debugger regained control is the default current thread. Inside the current thread, the most recent call is the default current call.

You can make any session, thread, or call current by right-clicking its node in the appropriate debugger window and choosing **Make Current**.

### 10.9.2.1 Debugger Windows and Context

Most debugger windows depend on the current context. When you change the current context, the contents of these windows are updated to reflect the new context.

For example, the Debugging window shows the threads in the current session, while the Call Stack window shows the call stack for the current thread. The Variables window shows the variables that are local to the current call, and the Loaded Classes window shows the classes that have been loaded by the current session. For more information on viewing classes and class instances, see [Section 10.9.6, "Viewing Program Information When Debugging."](#)

The exceptions are the Breakpoints and Watches windows. These windows list all breakpoints and watches set in the IDE. While the set of watches is shared by all sessions, an individual watch expression is evaluated and displayed based on the current context. For information on setting breakpoints, [Section 10.9.4, "Managing Breakpoints."](#)

For information on debugging threads, see [Section 10.9.6.8, "Debugging Threads in the IDE."](#)

For information on examining the call stack for the current thread, see [Section 10.9.6.9, "Using the Call Stack."](#)

### 10.9.2.2 The Source Editor and Context

When a variable is active in the current context, the Source Editor displays the value of the variable when you move the pointer over it. In cases where a program includes different variables with the same name, the Source Editor displays the value based on the current context, and not on the instance of the variable in the source code.

## 10.9.3 Attaching Source Code to a JAR File

When you add a JAR file or folder of compiled classes to a project's classpath, it is often useful to add the source files for those classes so that you can view their contents when working with them. Attaching source code to a JAR file or compiled classes folder lets the IDE know where to find the source code for those classes. You can then step into the source files when debugging and open the source files with the **Go To Source** command.

---

**Note:** For code completion to work properly in the IDE, you must either attach a complete set of source files as a folder or add the available source files as a Zip archive.

---

**To attach source code to a JAR file or compiled classes folder:**

1. Choose **Tools > Libraries** from the main menu.
  2. In the left pane of the Ant Library Manager, select the project library within which the JAR file you want to add the source code to is located.
- Only libraries already registered with the IDE are listed in the Ant Library Manager's Class Libraries list.
3. If the JAR file or classes folder for which you want to add the source code has not already been added to a registered library, create a new empty library using the **New Library** button.
  4. In the **Classpath** tab click **Add JAR/Folder** and specify the location of the JAR file containing the compiled class files.

A class library can contain multiple JAR files as well as their Javadoc documentation and source code.

5. In the Sources tab, click **Add JAR/Folder** to add the folder or archive file containing the source code.
6. Click **OK** to exit the Ant Library Manager.

The IDE adds the selected JAR files and source code to the specified library and automatically registers the source code in every project that has that JAR file on its classpath.

When you create a Java class library for a single JAR file, you can simply add the JAR file to the project's classpath to make the associated Javadoc and source code available. If your Java library contains multiple JAR files, however, you must add the library itself to the classpath. Adding the library to the classpath also makes it easier to share the project with other developers. For information on setting the classpath, see [Section 6.2.3.1, "Managing the Classpath."](#)

You can also associate the sources with a JAR file using the project's Project Properties window. However, doing so creates the association only for that project.

**To associate sources with a JAR file through the Project Properties window:**

1. Open the Project Properties dialog box by right-clicking the project node and choosing **Properties**.
2. Select the Libraries node in the Categories pane.
3. Select the JAR with which you want to associate the sources and click **Edit**.
4. Specify the sources to be associated.

**To attach source code for a Java platform:**

1. Choose **Tools > Java Platforms** from the main menu.
2. Select the platform in the left pane of the dialog box.
3. In the **Sources** tab, add the folders or archive files containing the source code.

For free-form projects, set the target JDK in your Ant script and specify the source level in the Project Properties dialog box (for more information, see [Section 6.8, "Setting the Target JDK"](#)). When you step into JDK classes, the IDE searches the platforms registered in the Java Platform Manager for a Java platform with a matching source level. If no matching Java platform is found, the IDE opens the source code for the IDE's default platform.

## 10.9.4 Managing Breakpoints

A *breakpoint* is a flag in the source code that tells the debugger to stop execution of the program. When your program stops on a breakpoint, you can perform actions like examining the value of variables and single-stepping through your program.

The IDE enables you to set several types of breakpoints using the New Breakpoint dialog. You can also set line breakpoints directly in the Source Editor. Breakpoints can be set for the following types of source elements:

- **Class.** You can break when the class is loaded into the virtual machine, unloaded from the virtual machine, or both.
- **Exception.** You can break whenever a specific exception is caught, whenever a specific exception is not handled in the source code, or whenever any exception is encountered regardless of whether the program handles the error or not.
- **Field.** You can stop execution of your program whenever a field in a specific class is accessed (for example, the method was called with the variable as an argument), modified or both.
- **Method.** Program execution stops every time the method is entered, exited or both.
- **Thread.** You can break program execution whenever a thread starts, stops, or both.

The Source Editor indicates a breakpoint by highlighting the line at which the breakpoint is set in red and placing an annotation in the left margin. The following table describes the debugging annotations:

**Table 10–1 Debugging Annotations**

Annotation	Description
	Breakpoint
	Disabled breakpoint
	Invalid breakpoint
	Multiple breakpoints
	Method or field breakpoint
	Disabled method or field breakpoint
	Invalid method or field breakpoint
	Conditional breakpoint
	Disabled conditional breakpoint
	Invalid conditional breakpoint
	Program counter
	Program counter and one breakpoint
	Program counter and multiple breakpoints

**Table 10–1 (Cont.) Debugging Annotations**

Annotation	Description
	The call site or place in the source code from which the current call on the call stack was made
	Suspended threads
	Thread suspended by hitting a breakpoint

All Java breakpoints are defined globally and, therefore, affect all IDE projects that include the source on which a breakpoint is set. For example, if you set a class breakpoint on `com.me.MyClass` in one project, the IDE stops execution every time it encounters that class during a debugging session for other projects that include the class.

You can view and organize all IDE breakpoints by choosing **Windows > Debugging > Breakpoints** (Alt+Shift+F5). If you open the Breakpoints window when a debugging session is running, it closes automatically when you end the debugging session. If you open the window when no debugging session is running, it stays open until you close it.

By default, each entry contains a short text description of the breakpoint and a checkbox indicating whether the breakpoint is enabled or disabled. You can enable or disable a breakpoint directly in the Breakpoints window by selecting or deselecting the checkbox.

#### 10.9.4.1 How to Set a Java Breakpoint

All Java breakpoints are defined globally and therefore affect all IDE projects that include the source on which a breakpoint is set. For example, if you set a class breakpoint on `com.me.MyClass` in one project, the IDE stops execution every time it encounters that class during a debugging session for other projects that include the class.

##### To set a line, field or method breakpoint in the Source Editor:

- Click in the left margin next to the line in the source code or put the insertion point in the line and choose **Debug > New Breakpoint** from the main menu.

A field, method or line breakpoint is created depending on if the line contains a field declaration, method declaration or other code. The corresponding breakpoint annotation is visible in the left margin next to the line of source code.

The IDE tests the validity of set breakpoints when the debugger session is starting or when a debugger session is already running. If a breakpoint is invalid the IDE uses a broken annotation to indicate the invalid breakpoint and displays an error message in the Debugger Console.

##### To set all other types of breakpoints:

1. In the Source Editor, select the code element on which you want to set a breakpoint.

For example, if you want to set a class breakpoint on the class `BeanCounter`, select the class name in the class declaration.

2. Choose **Debug > New Breakpoint** (Ctrl+Shift+F8).

The New Breakpoint dialog box opens with a suggested breakpoint type and target filled in.

3. If necessary, adjust the suggested breakpoint type in the Breakpoint Type drop-down list.
  4. Enter the package and class name for which you want to set the breakpoint.
  5. Set any additional options you require in the New Breakpoint dialog and click **OK**.
- The IDE creates the new breakpoint for the selected source element.

**To modify an existing breakpoint:**

1. Choose **Window > Debugging > Breakpoints** (Alt+Shift+5) to open the Breakpoints window.
2. Right-click any breakpoint and choose **Properties** to open the Breakpoint Properties dialog box.
3. Adjust any settings or actions you require and click **OK**.

The IDE updates the breakpoint for the selected source element.

**To enable and disable a breakpoint:**

- Right-click the breakpoint in the Breakpoints window and choose **Enable** or **Disable**.

You can modify and enable line, field and method breakpoints by right-clicking the breakpoint icon in the left margin of the Source Editor and choosing from the Breakpoint submenu.

When a debugging session is running, use code completion in the New Breakpoint dialog box.

#### 10.9.4.2 How to Set a Conditional Breakpoint

You can set conditions on a breakpoint so that execution only breaks if the condition is true. Set conditions on any breakpoint except thread breakpoints by selecting the **Conditions** checkbox and entering the condition. For all breakpoints you can specify how often the breakpoint is triggered by selecting the **Break When Hit Count** checkbox and choosing a criteria from the drop-down list and specifying a numerical value.

Class breakpoints and exception breakpoints enable you to set the following conditions:

- For class breakpoints, you can exclude classes triggering the breakpoint by selecting the **Exclude classes** checkbox and specifying the classes to exclude.
- For exception breakpoints, you can filter the classes triggering the breakpoint by selecting the **Filter on Classes Throwing the Exception** checkbox and specifying the names of classes to match or exclude.

**To set a conditions on a breakpoint:**

1. Create a new breakpoint or open an existing breakpoint's customizer by right-clicking its name in the Breakpoints window and choosing **Customize**.
2. Select the **Condition** checkbox and type the condition in the **Condition** field. The condition must follow the Java syntax rules. The condition can include anything that can be on the right side of the equal sign (=). The condition can also include

variables and methods that are within the current context. The following are exceptions:

- Imports are ignored. You must use fully qualified names, such as obj instanceof java.lang.String.
- You cannot access outerclass methods and variables directly. Use this.variableName or this\$1.
- (Optional) Select the **Break When Hit Count** checkbox and choose a criteria from the drop-down list and specify a numerical value.

Conditional line breakpoints have a  icon in the left margin of the Source Editor.

#### 10.9.4.3 How to Organize Breakpoints Into a Group

The Breakpoints window lists all of the breakpoints defined for all of your IDE projects. If you have numerous breakpoints set in the IDE, it is useful to organize these breakpoints into groups. Once your breakpoints are placed into groups, you can enable, disable, and delete them as a single unit.

**To add a breakpoint to a custom group:**

1. Choose **Windows > Debugging > Breakpoints** (Alt+Shift+5) to open the Breakpoints window.
2. Right-click the breakpoint and choose **Move Into Group** and select a custom group from the list.

---

**Note:** To create a custom group, right-click the breakpoint and choose **Move Into Group > New** and type the name of the new group in the dialog box.

---

**To remove a breakpoint from a custom group:**

- In the Breakpoints window, right-click the breakpoint and choose **Move Into Group > Default**.

Alternatively, you can modify how breakpoints are organized from the Breakpoints window by clicking Breakpoint Groups () and selecting a grouping strategy from the context window. If you choose Nested you can specify the specific groups and the order of the groups that are displayed.

#### 10.9.4.4 About Source Maps Support

Web client JavaScript debugger and node.js debugger use generated source maps to allow debugging in the original source files.

To debug the code you wrote but not the code that is executed as compressed or translated, the IDE scans your project files to find any present source map and assure that any breakpoint set in the files which were translated for execution (for example, compressed or transpiled), are submitted to the target files. During stepping through the application, the code positions and variable names are translated according to the existing source maps.

### 10.9.5 Managing Debugging Sessions

Debugging is the process of examining your application for errors. The process of debugging is accomplished by setting breakpoints and watches in your code and

running it in the debugger. This enables you to execute your code one line at a time and examine the state of your application to discover any problems.

When you start a debugging session the Debugging window opens in the left pane of the IDE. Additional debugger windows also appear automatically at the bottom of your screen.

You can also debug applications that are running on a remote machine by attaching the debugger to the application process.

### 10.9.5.1 How to Manage a Local Debugging Session

Local debugging is the process of debugging a program that is running on the same computer as the IDE. The IDE starts the debugger, then runs the application inside the debugger. When you start a debugging session, the IDE automatically opens the debugger windows and prints debugger output to the Output window.

#### 10.9.5.1.1 Debugging the Main Project

Debugging commands in the Debug menu are generally run on the main project, and when debugging a project it is recommended that you set the project as the main project.

If no project is set as the main project the Debug Project command is run on the project that is selected in the Projects window and the commands begin the debugging session in the main class of the selected project. For information on how to set a main project, see [Section 6.4, "Setting the Main Project."](#)

[Table 10–2](#) displays the commands in the Debug menu and the corresponding toolbar icons for starting and stopping the debugger.

**Table 10–2 Basic Debugging Commands and Icons**

Command	Icon	Description
Debug Project (Ctrl+F5)		Starts the debugger and runs the program until it reaches a breakpoint or exception or until the program terminates normally.
Finish Debugger Session (Ctrl+F5)		Stops the debugger.
Continue (F5)		Runs the program until it reaches the next breakpoint or until the program terminates normally.

#### To debug an individual project:

- Right-click the project in the Projects window and choose **Debug**. Alternatively, select the project in the projects window and choose **Debug > Debug Project** in the main menu.

The IDE runs the project in the debugger until execution stops or a breakpoint is reached.

#### To debug an individual file:

- Select any runnable file in the Projects window and choose **Debug > Debug my\_file**.

The IDE runs the file in the debugger until execution stops or a breakpoint is reached.

### 10.9.5.2 How to Manage a Remote Debugging Session

Remote debugging is the process of debugging an application that is running on a different computer. This technique is useful when you are developing an application that runs on a web server or in a different environment than the computer on which you are developing the application.

#### To start a remote debugging session:

1. On the computer where the application is located, start the application in debugging mode.
2. On the computer where the IDE is running, open the projects that contain the source for the application.
3. Choose **Debug > Attach Debugger** to open the Attach dialog box.
4. Select the appropriate Connector from the drop-down list, enter any required process information, and click **OK**.

For details on the transport and connector options, see the JPDA documentation, *Connection and Invocation Details*, for your version of the JDK from the Java Platform Debugger Architecture home page at

<http://docs.oracle.com/javase/7/docs/technotes/guides/jpda/conninv.html>.

The instructions for starting an application in debugging mode and the method for connecting to the VM depends on your VM. See your VM documentation for further details.

---

**Note:** You can view and select from the four most recent Attach configurations by expanding the menu under the **Debug** button in the toolbar.

---

### 10.9.5.3 How to Step Through Your Program

After execution of your program is halted, step through your lines of code using the commands listed in [Table 10–3](#) in the **Debug** menu or the corresponding icons in the toolbar:

**Table 10–3 Debugging Step Commands and Icons**

Command	Icon	Description
<b>Step Over</b> (F8)		Executes one source line. If the source line contains a call, executes the entire routine without stepping through the individual instructions.
<b>Step Over Expression</b> (Shift+F8)		Executes one method call in an expression. If an expression has multiple method calls, you can use <b>Step Over Expression</b> to step through an expression and view the value of each method call in the expression in the Variables window. Each time you use the <b>Step Over Expression</b> command, the debugger advances to the next method call in the expression and the completed method call is underlined. <b>Step Over Expression</b> behaves like <b>Step Over</b> when there are no additional method calls.  For more information, see <a href="#">Section 10.9.6.11, "How to Step Through an Expression."</a>

**Table 10–3 (Cont.) Debugging Step Commands and Icons**

Command	Icon	Description
<b>Step Into</b> (F7)		Executes one method call in a source line. If the line has more than one method call you can choose which method call to step into by using the arrow keys or mouse in the source editor to select the method call. The selected method call to step into is indicated by a box around the method call in the source editor. The most likely method call in the line is selected by default.
<b>Step Into Next Method</b> (Shift+F7)	(no icon for this command)	Executes one source line. If the source line contains a call, the IDE stops just before executing the first statement of the routine. You can also start a debugging session with the Step Into command. Program execution stops on the first line after the main routine before any changes have been made to the state of the program.
<b>Step Out</b> (Ctrl+F7)		Executes one source line. If the source line is part of a routine, executes the remaining lines of the routine and returns control to the caller of the routine. The completed method call is highlighted in the Source Editor.
<b>Run to Cursor</b> (F4)		Runs the program to the cursor location in the Source Editor and pauses the program. The file you have selected in the Source Editor must be called from the main class of the main project.

**To view source code files potentially available to the Debugger:**

- Choose **Window > Debugging > Sources** (Alt+Shift+8) to open the Sources window.  
The Sources window lists the source directories on your project classpath. The current source file is checked by default, meaning that the Debugger uses it during the debugging session. If you want the Debugger to be able to step into source files other than the current one, you can select those files in this window.

**10.9.5.4 How to Fix and Continue in a Debugging Session**

If you find a problem while debugging, you can use the **Apply Code Changes** command to fix your source and then continue debugging with the changed code without restarting your program.

It is not possible to use the **Apply Code Changes** command to do the following:

- Change a modifier of a field, a method, or a class
- Add or remove methods or fields
- Change the hierarchy of classes
- Change classes that have not been loaded into the virtual machine

**To fix your code:**

1. From the main menu, choose **Debug > Apply Code Changes** to recompile and begin repairing your source code.
  - If there are errors during compilation, nothing is changed in your program. Edit your source code as needed, then execute the **Apply Code Changes** command again.

- If there are no errors, the resulting object code is swapped into the currently executing program. However, all calls on the call stack continue running the unfixed code. To use the modified code, you must pop from the call stack any calls that contain modified code. When the calls are reentered, they use the modified code.

---

**Note:** If you modify the currently running method, the IDE displays an alert box. If you click **Pop Call**, the most recent call is removed from the current call stack. If you click **Leave Call**, the program executes with the original version of the code. If there is only one call in the stack, you cannot pop the call and continue. For more information on popping the call stack, see [Section 10.9.6.9, "Using the Call Stack."](#)

---

2. Continue the program to verify that the fixed version of your code works correctly.

The **Apply Code Changes** command does not automatically rebuild JAR files, executable files, or similar files. You must rebuild these files if you want to debug them in a new session.

#### 10.9.5.5 How to Finish a Debugging Session

If necessary, stop the current debugging session using the Shift-F5 shortcut. You can also close a specific debugging session using the Sessions window.

**To finish the current debugging session:**

- Choose **Debug > Finish Debugger Session** (Shift+F5).

**To finish one of several debugging sessions:**

1. Open the Sessions window by choosing **Window > Debugging > Sessions** (Alt+Shift+6).

The information given for each session includes the session name and state. In most cases, the state corresponds to the state of the process associated with the session. One session is always considered the current session, unless no sessions are running. By default, the current session is the session that you most recently started.

2. Right-click the debugging session you want to stop and choose **Finish**.

For information on local debugging sessions, see [Section 10.9.5.1, "How to Manage a Local Debugging Session."](#)

For information on remote debugging sessions, see [Section 10.9.5.2, "How to Manage a Remote Debugging Session."](#)

### 10.9.6 Viewing Program Information When Debugging

In the IDE, local variables are listed in the Variables window, however, it is also possible to evaluate variables directly in the Source Editor. You can view the values returned by each method call in an expression by stepping through an expression.

#### 10.9.6.1 Using the Variables Window

For each variable within the current call, the Variables window displays information including the variable name, type, and value. Choose **Window > Debugging > Variables** to open this window.

The Variables window also displays all of the static fields from the present class and all superclasses for each variable, as well as all of the inherited fields from all superclasses. For information on how to a call the current call, see [Section 10.9.2, "Choosing Current Context in the Debugger."](#)

You can change the value of a local variable directly in the Variables window and then continue running your program with the new value in place.

In some cases, the debugger assigns a pound sign (#) and a number as the variable's value. This number is an unique identifier of the given instance. You can use this identifier to determine if a variable points to the same instance or to a different instance. You cannot edit this value.

#### 10.9.6.2 Using the Loaded Classes Window

The Loaded Classes window displays the classes loaded on the heap and the percentage and number of object instances. You can sort the classes by class name or the number or percentage of instances. You can also use the filter box at the bottom of the window to filter the list by class name. If you open the Loaded Classes window when a debugging session is running, it closes automatically when you end the debugging session. If you open the window when no debugging session is running, it stays open until you close it.

The contents of the Loaded Classes window is dependent on the current context. When you change the current session, the Loaded Classes window is updated to show the classes for that session. Choose **Window > Debugging > Loaded Classes** to open this window.

To view information for a particular class, right-click a class in the Loaded Classes window and select **Show in Instances**. The Instances window displays the number of class instances, fields, and references to the class that occur in the current context.

#### 10.9.6.3 Using the Events Window

The Events window displays the events associated with GUI form elements according to the triggering order and also displays the listeners that are associated with the event. Open the Events window by right-clicking a component in the Visual Debugger and choosing **Show Listeners** in the context menu.

Expand the Custom Listeners node or the Internal SWT/Swing Listeners node to view a list of listeners in the project. Right-click a listener and choose **Go to Component Source** to open the source file at the line where the listener is defined.

#### 10.9.6.4 Evaluating Variables in the Source Editor

You can also evaluate a variable directly in the Source Editor by moving the insertion point over the variable. If the variable is active in the current context, the value of the variable is displayed in a tool tip. In cases where a program includes different variables with the same name, the Source Editor displays the value based on the current context, and not on the instance of the variable in the source code. Structured values can be expanded into a detailed view. Expandable tooltips feature an **Expand** icon (⊕) on the left. Alternatively, you can expand a tooltip by pressing the Space bar.

You can track the changes in the value of a variable during program execution by setting a watch on the variable. When you create a watch, the value of the variable is immediately evaluated and displayed in the Watches window.

### 10.9.6.5 How to Create a Watch

A *watch* enables you to track the changes in the value of a variable or expression during program execution. The Watches window lists all of the watches you have defined for all of your IDE projects.

#### To open the Watches window:

- Choose **Window > Debugging > Watches** (Alt+Shift+2).

#### To create a watch from the Source Editor:

1. Select the variable or expression in the Source Editor, right-click, and choose **New Watch** (Ctrl+Shift+F7).

The New Watch dialog box opens with the variable or expression entered in the text field.

2. Click **OK**.

The Watches window opens with the new watch selected.

#### To create a watch from the Variables window:

1. Choose **Window > Debugging > Variables**.
2. Click the Create New Watch icon ( ) in the toolbar.

You can click in the Value cell to edit the value directly in the Watches window.

If you specify an expression, follow the syntax rules of the debugger that you are using.

When you create a watch, the value of the variable or expression is immediately evaluated and displayed in the Watches window. The value of the watch is based on the current context. When you change the current context, the Watches window is updated to show the value of the watch for that context.

When a debugging session is running, you can use code completion in the New Watch dialog box.

For information on how to view variable values during a debugging session, see [Section 10.9.6, "Viewing Program Information When Debugging."](#)

### 10.9.6.6 How to Create a Fixed Watch

A fixed watch describes the object that is currently assigned to a variable while a normal watch describes the content of the variable. Fixed watches are specific to the Java 2 debugger.

For example, consider the following code:

```
java.awt.Dimension dim=new java.awt.Dimension(10,20);
java.awt.Dimension newDim=dim;
dim=new java.awt.Dimension(20,30);
newDim.height=15
```

With the debugger stopped on the second line, you can create a normal watch on the variable `dim`. If you create a fixed watch on the variable `dim`, the watch describes the object that is currently assigned to the variable, which is `java.awt.Dimension(10, 20)`. If you press F8 to step over the code three times, the value of the normal watch becomes `java.awt.Dimension(20, 30)`. This change occurred on the third line of source code. The value of the fixed watch is `java.awt.Dimension(15, 20)`. The fixed

watch was created on the object with a height of 10, but the fourth line changed the height of this object to 15.

**To create a fixed watch:**

- In the Variables or Watches window, right-click a variable and choose **Create Fixed Watch**.

The fixed watch is added to the Watches window.

#### **10.9.6.7 How to Pin a Watch**

You can set a watch or multiple watches in your code to review a variable value in the Source Editor. The pinned watch updates its value like a watch in the **Watches** window and keeps the last known value even after the debugging session finishes.

**To pin a watch:**

1. Hover a mouse over a variable or a selected expression in the Source Editor area until the IDE displays the tooltip with the value of the variable or your selection.
2. Press the **Enter** key or click the **Pin** icon () in the tooltip.

The IDE adds a watch pinned into the Source Editor and displays a pinned watch window with two icons on the right side - the **Comment** () and **Close** () icons. You can drag the pinned watch window with the mouse.

To display the pinned watches in the **Variables** and **Watches** windows, right-click in either window and choose **Show Pinned Watches** from the context menu.

#### **10.9.6.8 Debugging Threads in the IDE**

All the threads created in the current session are displayed in the Debugging window. Choose **Window > Debugging > Debugging** (Alt+Shift+9) to open the Debugging window. You can also see the list of threads in the current session in the Threads window. Choose **Window > Debugging > Threads** (Alt+Shift+7) to open the Threads window.

The information given for each thread is the thread name, state and if the thread is suspended. One thread is the current thread. By default, the current thread is the thread in the current session from which the debugger gained control. When you select a different current session, the Threads window is updated to show the threads for that session.

##### **10.9.6.8.1 Changing the Current Thread**

Only one thread is the current thread at any given time. By default, the current thread is the thread within the current session from which the debugger gained control. When you switch between threads to debug, the Variables window is automatically updated to reflect the data applicable to that thread.

**To change the current thread:**

- Double-click any thread in the Debugging window to make it the current thread.

##### **10.9.6.8.2 Suspending and Resuming Threads**

You can suspend execution of a thread if you think it is causing problems and then later resume the thread once the problem is solved. The Debugging window enables you to easily see the threads in the debugging session and identify the running and suspended threads. The icon to the left of the thread name indicates whether the thread is suspended or waiting to be resumed.

You can suspend, interrupt and resume application threads by right-clicking a thread in the Debugging window and choosing an action from the context menu.

Alternatively, you can click the **Resume** (▶) and **Suspend** (⏸) buttons in the right side of the Debugging window. Hide the **Suspend** and **Resume** buttons by clicking the **Show suspend/resume table** button (▶) in the Debugging window toolbar.

You can also resume and suspend threads in the Threads window.

#### 10.9.6.8.3 Editor window icons

A thread icon in the left margin of the source editor indicates that there is a suspended thread at that line. The following table describes the icons representing the thread states that appear in the source editor during a debugging session.

**Table 10–4 Thread State Icons**

Icons	Description
	Other suspended threads
	Other threads suspended by hitting a breakpoint

#### To switch a suspended thread to the current thread:

- In the source editor, right-click the suspended thread icon and choose **Set Current Thread To > new\_current\_thread**.

**10.9.6.8.4 Multi-threaded Applications** When debugging a multi-threaded application, a step in a particular thread can be interrupted by a breakpoint encountered in some other thread. When this occurs, the IDE gives you the option to switch threads. The IDE does not automatically switch the context to the new thread.

When a breakpoint in another thread is encountered, you are notified by a panel that appears in the Debugging window. The current thread remains the current thread until you explicitly switch it or the thread stops. Click the arrow in the panel in the Debugging window and choose a thread to switch to that thread at any time. This action enables you to continue debugging the current thread and when it is convenient you can switch to the thread that hit a breakpoint.

You can use the **Current Thread Chooser** (Ctrl+8) to select the thread you want to be the current thread.

**10.9.6.8.5 Viewing Source Code for a Thread** If you suspect the source code of a thread is causing problems, you can examine it in the Source Editor.

#### To view a thread's source:

- Right-click the thread in the Threads window and choose **Go To Source**. If the source of the thread is available, the Source Editor jumps to the current call on the thread's call stack.

#### 10.9.6.9 Using the Call Stack

In a debugging session, you can see the call stack for the current thread in the Debugging window (opened automatically whenever you start a debugging session). If you expand the node for the current thread you can see a list of the sequence of calls made during the execution of the thread.

The information given for each call (marked by a  icon) includes the name of the call, followed by the file name and line number of the call's currently executing statement. If the sources are available, you can right-click the call and choose **Go To Source** to go to the source code of the call.

Alternatively, you can open the Call Stack window by choosing **Window > Debugging > Call Stack** (Alt+Shift+3).

**10.9.6.9.1 Changing the Current Call** The current call (indicated in bold) is the most recent call made by the current thread. When you select a different current thread, the window is updated to show the calls for that thread. The Variables window is also updated to display the values of variables for the current call.

**To browse the call stack, do any of the following:**

- To move one level away from the main routine, choose **Debug > Stack > Make Callee Current** (Ctrl+Alt+up arrow).
- To move one level toward the main routine, choose **Debug > Stack > Make Caller Current** (Ctrl+Alt+down arrow).
- To make a call current, double-click the call in the Call Stack window.

You can capture a textual representation of the call stack by right-clicking a call and choosing **Copy Stack** from the context menu. When you copy the call stack, the text is copied to the clipboard. You can then paste the call stack into a text file.

**10.9.6.9.2 Popping a Call From the Call Stack** You can change the execution of your program so that the next statement to be executed is one of the calls made earlier on the stack. In general, popping, or removing, a call from the call stack does not undo any effects that the call caused. For example, if a call opened a database connection and then that call is removed, the database connection remains open.

---

**Note:** You can only pop a call if the program being debugged is paused.

---

**To pop the most recent call from the call stack:**

- From the main menu, choose **Debug > Stack > Pop Topmost Call**.

The call is removed from the call stack. The program counter is moved to the line before the instruction that made the removed call. If the source code is available, the focus of the Source Editor is set to that line. When you continue program execution, the call is repeated.

**To pop multiple calls from the call stack:**

- In the Debugging window, right-click the call that you want to remain at the top of the call stack and choose **Pop to Here**.

All of the calls above the selected call are removed from the call stack. The program counter is moved to the line before the instruction that made the removed call. If the source code is available, the program counter moves to that line. When you continue program execution, the call is repeated.

**10.9.6.10 How to Evaluate Code**

When you are in a debugging session, you can evaluate any code snippet to view the values of local variables for the current method, class variables, and method calls.

You can view the values returned by each method call in the expression by stepping through an expression.

**To evaluate an expression:**

1. Start a debugging session.
2. Choose **Debug > Evaluate Expression** (Ctrl+F9) from the main menu.
3. Type a code snippet in the Evaluate Code window and click  in the bottom right corner of the window). The result is shown in the Variables view. The result with historical values is shown in the Evaluation Results view.

Click the Create New Watch button () in the Variables view to set a watch for the expression. The watch is added to the Watches view.

If you evaluate a method that changes a global variable, the global variable's value is changed in the debugging session as well.

If you try to evaluate a variable that is outside the scope of the current method, a message displays "variable" is not a known variable in current context. Use the Call Stack window to change the current method context.

If you try to access a global variable before the file that contains the variable is instantiated, a message displays Cannot access instance variable "variable" from static context.

#### 10.9.6.11 How to Step Through an Expression

When you are in a debugging session, use the **Step Over Expression** command to step through an expression and view the values returned by each method call in the expression.

Use the **Step Over Expression** command to achieve a more fine-grained stepping than other debugging steps. **Step Over Expression** enables you to proceed through each method call in an expression and view the input parameters and resulting output values of each method call. Invoke the **Step Over Expression** command just as you would any other step commands. If there are no further method calls, **Step Over Expression** behaves like the **Step Over** command.

Each method call in an expression has some input (parameters) and output values. Each time you use the **Step Over Expression** command in an expression, the debugger resumes VM execution and then stops before executing the next method call. You can inspect the output values for the previous method and the input parameters for the next method in the Variables window. Invoking **Step Over Expression** again resumes the VM execution until the next method call in the expression.

**To step through an expression:**

1. Place a breakpoint in the line containing the expression you want to debug and start the debugging session.

When you start the debugger, the VM stops before executing any of the method calls in the expression.

2. Choose **Debug > Step Over Expression** (Shift+F8) from the main menu.

The debugger steps to the first method call in the expression but does not execute the method. The Variables window displays the input parameters of the method.

3. Invoke **Step Over Expression** again to execute the first method and step to the next method call in the expression.

The Variables window displays the output values of the executed method and any input parameters for the next method in the expression.

In the Source Editor, the executed method call is underlined and the next method call in the expression is highlighted. Mouse over the executed method to display a tooltip and view the output values.

4. Invoke **Step Over Expression** again to step to the next method call in the expression.

The Variables window displays the output values under the Return Values node.

---

# Implementing Java GUIs

This chapter describes how to implement Java GUIs using the IDE's Java GUI tools.

- [About Implementing Java GUIs](#)
- [Working with the GUI Builder](#)
- [Working with Layout Managers](#)
- [Adding a Bean to the Window](#)
- [Working with Database Applications and Beans Binding](#)
- [Deploying GUI Applications](#)
- [Configuring the GUI Builder](#)

## 11.1 About Implementing Java GUIs

In Java applications, the components that comprise a GUI (Graphical User Interface) are stored in containers called forms. The Java language provides a set of user interface components from which GUI forms can be built.

The IDE's GUI Builder assists you in designing and building Java forms by providing a series of tools that simplify the process.

### 11.1.1 The IDE's Java GUI Tools

The IDE provides several tools to simplify the process of building GUIs:

- **GUI Builder.** The primary workspace within which GUI design takes place in the IDE. The GUI Builder enables you to lay out forms by placing components where you want them and by providing visual feedback in the form of guidelines. See [Section 11.2, "Working with the GUI Builder."](#)
- **Navigator window.** Displays a tree hierarchy of all components contained in the currently opened form. Displayed items include visual components and containers, such as buttons, labels, menus, and panels, as well as non-visual components such as timers and data sources.
- **Palette window.** A list containing all the components that can be added to forms. You can customize the window to display its contents as icons only, or as icons with component names.
- **Properties window.** Displays the editable settings for the currently selected component.
- **Connection wizard.** Assists in setting events between components in a form without the need of writing code manually.

- **Manager.** Enables you to add, remove, and organize window components such as Swing components, AWT components, Layouts, and beans.

In addition, the IDE provides support for the Beans Binding specification which provides a way to synchronize the values of different bean properties. This support also simplifies the creation of desktop database applications.

To access binding features, right-click a component, select **Bind** in the context menu, and select a property from the drop-down list. For more information see "[Working with Database Applications and Beans Binding](#)."

For more information about creating Java GUIs, see the *Java GUI Application Learning Trail* at:

<https://netbeans.org/kb/trails/matisse.html>

For more information about handling events in your application, see the *Creating a GUI with JFC/Swing tutorial* at:

<http://docs.oracle.com/javase/tutorial/uiswing/>

## 11.2 Working with the GUI Builder

The GUI Builder is a tool for designing GUIs visually. As you create and modify your GUI, the IDE automatically generates the Java code to implement the interface. GUI forms are indicated by form nodes (□) in the Projects, Files, and Favorites windows.

When you open a GUI form, the IDE displays it in an Editor tab with toggle buttons that enable switching between Source and Design views. The Design view enables you to work with GUI forms visually while the Source view permits the form's source code to be edited directly. Each time you select a form's Design toggle button, the Palette, Navigator, and Properties windows appear automatically.

Components are typically added to a form using the window and arranged in the GUI Builder workspace. As you work, the GUI Builder automatically displays guidelines suggesting preferred alignment and anchoring for the components you add. Use the Navigator window in conjunction with the Properties window to examine a form's component and layout manager properties, manage component event handlers, and define how code is generated.

---

**Note:** Interfaces created with the GroupLayout layout manager must have the Swing Layout Extensions library available if they are run outside the IDE. The Swing Layout Extension Library is included in the Java Platform JDK 6 or higher, so no additional steps are needed if you develop the application with JDK 6 or 7 and deploy in environments that have JRE version 6 or higher. For more information, see [Section 11.6, "Deploying GUI Applications"](#).

---

### To design GUI applications:

These are the basic steps to create and deploy GUI applications:

1. **Create forms.** Forms can be created within existing projects. To simplify handling, a form includes containers, subcontainers, and components. Layout managers control the arrangement of components within a container. You also have the option to create a Multiple Document Interface (MDI) application.
2. **Edit forms.** A component within a form has behavior, appearance, and accessibility properties that can be modified directly or through the property

editor. See [Section 11.2.10, "How to Manage Component Events"](#) and [Section 11.3.1, "How to Set the Layout Manager."](#)

3. **Preview forms.** With the Preview Design capability you can test your form without compiling or running it. See [Section 11.2.14, "How to Preview a Form."](#)
4. **Deploy the GUI application.** Applications are typically distributed as JAR files. Before deployment you must ensure that the JAR contains all necessary libraries. See [Section 11.6, "Deploying GUI Applications."](#)

### 11.2.1 How to Create a New Form

In the IDE you can create JFC/Swing or AWT (Abstract Window Toolkit) forms, pre-built sample application skeletons, or any class that is based on the JavaBeans component architecture using the provided templates.

---

**Note:** The GUI Builder cannot be used to edit GUI forms that were created outside of the IDE.

---

**To create a new GUI form in an existing project:**

1. Choose **File > New File** from the main menu.
2. In the New wizard's **Project** combo box, select the project for which you want to create the form.
3. Expand the Swing GUI Forms or AWT GUI forms node in the Categories pane and select the desired form template. Click **Next**.
4. Enter the GUI form's class name and location. Click **Finish**.

The IDE creates a blank form of the selected type and opens the form in the Source Editor's Design view.

---

**Note:** In order to avoid repaint problems at both design and run time, only AWT components should be used in AWT forms and only JFC/Swing components should be used in JFC/Swing forms.

---

**GUI Form Types in the New File Wizard:**

The following table lists the types of form templates available in the IDE. Each differs in the design time and run time look of the form, as well as in the code generated for the form's class.

**Table 11-1 GUI Form Types**

Form Type	Description
JApplet	Program run by a Java-enabled web browser or other applet viewer.
JDialog	Modal or modeless window for collecting user input.
JFrame	Top-level application window.
JInternalFrame	An internal frame that can be placed on a JDesktopPane component to create an MDI application.
JPanel	Lightweight container for holding parts of an interface. In turn, the container can be used in any other container, such as a JFrame, JPanel, JApplet, or JDialog component.

**Table 11–1 (Cont.) GUI Form Types**

Form Type	Description
Bean Form	<p>The template used to create a new form based on any JavaBeans component. The new form can be visual or nonvisual. In the New wizard go to the Superclass page and on the Form Superclass page specify the bean class. The bean class that you specify when creating the new form must be in the classpath and must be already compiled.</p> <p>A bean is any class that complies with the JavaBeans component architecture. A bean must have a public constructor without parameters. Use any JFC/Swing component as an example of a JavaBeans class. For example, use <code>javax.swing.JButton</code> to create a form to produce a customized button.</p> <p>To use this template to create a plain container for holding beans, specify <code>java.lang.Object</code> as the superclass.</p>
AWT Forms	Visual forms that are based on the AWT. The AWT forms include Applet, Dialog, Frame, and Panel.
Sample Forms	Customized sample forms that include a JFrame-based application with three menus, a JFrame application that can be used as the main window for an MDI application, and a dialog box with <b>OK</b> and <b>Cancel</b> buttons.

## 11.2.2 How to Work with Containers

Java GUIs are forms comprised of top-level containers within which are grouped sub-containers as well as the various components used to provide the desired information and control functionality.

### 11.2.2.1 Controlling View Focus

It is often useful to focus work on single subcontainers rather than the entire form the GUI Builder generally displays. When working with large forms containing complex nested hierarchies of containers, changing the scope of the GUI Builder's focus enables you to concentrate on specific parts of your interface.

#### To change the GUI Builder's focus to a specific container:

In the GUI Builder or Navigator window, right-click the container you want to edit.

- Choose **Design This Container** from the contextual menu.
- The IDE adjusts the workspace display such that the current container fills the work area and hides the form's other components. The form's entire hierarchy remains available in the Navigator window.

#### To return the GUI Builder's display focus to the entire form:

- Right-click the container in the GUI Builder.
- Choose **Design Top Container** from the contextual menu.

The IDE adjusts the work area display such that the entire form is visible. If the **Design Top Container** menu item is dimmed, you are already designing the entire form.

### 11.2.2.2 Reordering Components Within a Container

The component order in a container follows the sequence in which components are added. See [Section 11.2.3, "How to Add a Component to a Form."](#) If the layout manager for a container does not use constraints (FlowLayout, BoxLayout, and GridLayout), the order of components also determines how they are arranged visually.

You can reorder the components in the Navigator window or drag the components in the form itself.

With layout managers that use constraints (BorderLayout, GridBagLayout, CardLayout, AbsoluteLayout, and Null Layout), the order of components in the container does not determine the order in which the components appear. For these containers, the component order can only be changed in the Navigator window. Although GridBagLayout uses constraints to determine how components are arranged, component order determines the layout when the Grid X and Grid Y constraints are not used. See [Section 11.3, "Working with Layout Managers."](#)

### 11.2.3 How to Add a Component to a Form

Once you have created a new form, you can add components to display information and control functionality. In the IDE, you can add components several ways, including:

- Drag and Drop
- Pick and Plop
- The Navigator window. The IDE's Navigator window provides a hierarchical tree view of the form's various components. Each form is represented by a root node (inode) within which all components in the form's class are contained. All other GUI components are organized into one of the following two subnodes:
  - **Form Container node** (inode). Represents the top level of the form's hierarchy and contains the visible components currently displayed in the GUI Builder.
  - **Other Components node** (inode). Contains the form's non-visual components.

#### To add a component from the Palette:

1. To select a component in the window click its icon. Without releasing the mouse button, drag the component to the desired location in the form. For more information, see [Section 11.2.4, "How to Select Components in a Form."](#)
2. If you want to add the component to the Other Components node, drag the component to the white area outside the form in the GUI Editor's workspace.

The IDE adds the component to the selected container in the form.

#### To add multiple components from the Palette:

1. Click a component icon.
2. Shift+Click in the form to place the first component.
3. Click all locations.

If you want to add the component to the Other Components node, click in the white area outside the form in the GUI Editor's workspace.

The IDE adds the components to the selected container in the form.

#### To add a component using the Navigator window:

1. In the Navigator window, right-click the container to which you want to add a component.
2. Choose the desired component from the **Add From** submenu.

The IDE adds the component to the selected container in the form.

**To add a bean from the Projects, Files, or Favorites windows:**

1. In the Files window, locate the bean's class node. The class must be a bean and it must be compiled.

---

**Note:** For a class to be a bean, it must be possible to create an instance of the class using an empty public constructor.

---

2. Right-click the class node and choose **Copy** from the contextual menu.
3. Choose **Paste** from the contextual menu of either the GUI Builder or the Other Components node in the Navigator window.

The IDE adds the bean to the current form. See [Section 11.4, "Adding a Bean to the Window."](#)

---

**Note:** When working with nested containers, it is sometimes difficult to select the container to which you want to add the component. You must first select the desired container before selecting the component you want to add. You then select the component from the window, hold down the Alt key, and click the desired container again to add the component.

---

#### 11.2.4 How to Select Components in a Form

The IDE's GUI Builder and Navigator window always display the same selected component or components. Selecting a component in one window automatically selects it in the other.

**To select a single component:**

Click the component's node in the Navigator window or click the component in the GUI Builder.

**To select multiple components, do one of the following:**

- Hold down the Control key while clicking multiple components in the Form Editor or while clicking multiple nodes in the Navigator window.
- In the GUI Builder, hold down the Shift key while holding down the left mouse button and drag the mouse over each component in the group. A rectangle is painted as the mouse is dragged over the group of components. When the mouse button is released, all the components included in the rectangle are selected.
- In the Navigator window, to select a consecutive group of components, Shift+click the first one, and while holding down the Shift key, click the last component in the group.

**To select next and previous components:**

Press Tab in the GUI Builder to select the next component or press Shift+Tab to select the previous component.

#### 11.2.5 Controlling Selection Depth

When working with nested components (components within a container like a panel for example), clicking in the GUI Builder always selects the deepest component at the

point of the click. You can, however, select nested components on different levels using modifier keys, as described below.

Right-click to display the contextual menu for the selected container or subcomponent. The right-click action does not change the selection depth.

**To select the parent container of a selected component:**

1. Press and hold the Alt key.
2. In the GUI Builder, click the selected component.

If the currently selected component does not have a parent container (i.e. it is the container component for the entire form), Alt+Click selects the deepest component at the point of the click.

**To select a subcomponent of a selected container:**

1. Press and hold both the Alt and Shift keys.
2. Click the subcomponent.

If the currently selected component does not contain any subcomponents, the container component for the entire form is selected.

---

**Note:** It is often easier to select components using the Navigator window than the GUI Builder.

---

### 11.2.6 How to Align Components

Once you have added components to a form you can adjust the alignment.

**To align components:**

1. Select the components you want to align in the GUI Builder's workspace.
2. Click the appropriate align button in the GUI Builder toolbar. Alternately, you can right-click either component and choose **Align > Left in Column** (or **Align > Right in Column**) from the pop-up menu.

The IDE shifts the component positions such that the specified edges are aligned and the component anchoring relationships are updated.

**To align component baselines:**

1. Select the component you want to align in the GUI Builder's workspace.
2. Drag the component to the right or left of the selected component.
3. A horizontal guideline appears indicating that the selected component's baseline is aligned with the baseline of the second component and a vertical guideline suggest the spacing between the two components. Click to position the component.

The IDE snaps the second component into a position aligned with the baseline of the first and displays status lines indicating the component spacing and anchoring relationships.

**To indent components:**

1. Select the component you want to indent in the GUI Builder workspace.
2. Select the component below the component below which you want to indent it.

3. When guidelines appear indicating that the first component's left edge is aligned with that of the JLabel, move it further to the right until secondary indentation guidelines appear.
4. Click to position the component.

The IDE indents the second component below the first component.

For more information of working with components see [Section 11.2.4, "How to Select Components in a Form,"](#) [Section 11.2.7, "How to Size Components,"](#) and [Section 11.2.8, "How to Edit Component Properties."](#)

## 11.2.7 How to Size Components

It is often beneficial to set several related components, such as buttons in modal dialogs, to be the same size so they can be easily recognized as offering similar functionality. You can also adjust the resizing behavior of components to ensure that component relationships are maintained at runtime.

### To set components to the same size:

1. Select all of the components you want to resize in the GUI Builder's workspace.
2. Right-click any one of components, and choose **Set the Same Size > Set Width** (or **Same Size > Set Height**) from the contextual menu.

The IDE sets the selected components to the same size and adds small graphic to indicate the component relationships.

### To set component resizing behavior:

1. Select the components whose auto resizing behavior you want to set.
2. Right-click any one of the components and choose **Auto Resizing > Horizontal** (or **Auto Resizing > Vertical**) from the pop-up menu.

The IDE sets the components auto-resizing behavior to resize horizontally at runtime. The alignment guidelines and anchoring indicators are updated to indicate the new component relationships.

For more information of working with components see [Section 11.2.4, "How to Select Components in a Form,"](#) [Section 11.2.7, "How to Size Components,"](#) and [Section 11.2.8, "How to Edit Component Properties."](#)

## 11.2.8 How to Edit Component Properties

Once you have added a component to a form, you can edit values in the Properties window to modify its behavior and appearance.

### To edit a component's properties:

1. Select the component in the GUI Builder or Navigator window to display its properties in the Properties window.

---

**Note:** If you select multiple components, their common properties are displayed and any modifications apply to all of the selected components.

---

2. Click a button at the top of the Properties window to select an appropriate property.

3. In the Properties window select the property and enter the desired value.
4. (Optional) If the property you want to edit has an ellipsis (...) button, click it to open a special property editor where you can modify both the property and the initialization code generated for it.
5. (Optional) If you are using the property editor, use the **Select Mode** combo box to choose between available custom editors for the property, make the necessary changes, and click **OK**.

The IDE applies the new parameter values to the selected components.

**To return a property to its default value:**

Right-click the property name and choose **Restore Default Value** from the contextual menu. The IDE returns the selected to its default value.

---

**Note:** You can edit text label properties for common components directly in the GUI Builder.

---

For more information of working with components see [Section 11.2.10, "How to Manage Component Events,"](#) [Section 11.3.4, "How to Set Layout Properties,"](#) and [Section 11.2.11, "How to Modify GUI Source Code."](#)

### 11.2.9 How to Set Events with the Connection Wizard

Use the Connection wizard to set events between two components within a form without having to write code manually.

**To set an event using the Connection wizard:**

1. Click the **Connection Mode** button () in the GUI Builder toolbar.
2. In the GUI Builder or Navigator window, select the component that fires the event. [Section 11.2, "Working with the GUI Builder."](#)

---

**Note:** The selected component is highlighted in red when selected.

---

3. Select the component whose state you want to affect with the event.

---

**Note:** The selected component is highlighted in red when selected.

---

4. On the Connection wizard's **Select Source Event** page, expand the event type directory node and select the event that the trigger component fires from the Events list. Accept the default handler method name or type in a new name and click **Next**.
5. On the Specify Target Operation page, specify the operation to be performed on the target component by selecting the appropriate radio button and choosing from the list of operations. Click **Next**.

---

**Note:** If you choose a method call with no parameters, the wizard's **Finish** button becomes available.

---

6. (Optional) If you wish to enter event handler code directly, click the **User Code** radio button and click **Finish**. The insertion point is placed in the new event handler in the Source Editor, where you can write the event handler code yourself. For more information see [Section 11.2.10, "How to Manage Component Events."](#)
7. On the Enter Parameters page, specify the values for each tab's target property or method by selecting the source from which each parameter's value is acquired.  
The source code generated for each of the parameters is displayed in the Generated Parameters Preview field.
8. Click **Finish**.  
The IDE automatically generates the code to connect the form's components.

## 11.2.10 How to Manage Component Events

The Java programming language uses events to enable GUI form behavior. Source objects can trigger events which one or more objects with event listeners react to by means of event handlers.

### 11.2.10.1 Defining Event Handlers

You can define event handlers using a component's property sheet or contextual menu. You can also define an event handler using the Connection wizard.

#### To define an event handler using the property sheet:

1. Select the component in the Navigator window.
2. Click the **Events** button at the top of the Properties window.
3. Click the value of the desired event in the list. Initially, the value for all events is **<none>**. When you click the value field, **<none>** is automatically replaced with the default event name.
4. Click the event's ellipsis (...) button to open the Handlers dialog box.
5. Click the **Add** button to add a new name to the list of handlers. Click **OK**.

The code for the listener and the empty handler method body is generated.

---

**Note:** You still must add the desired code for the new event handler in the Source Editor.

---

#### To define an event handler using the contextual menu:

1. Right-click a form component in the Files window, Project window, or Navigator window.
2. Choose **Events** from the contextual menu and its submenus. Bold menu items in the **Events** submenus indicate event handlers that have already been defined.  
The code for the listener and the empty body of the handler method is generated. The default name is assigned to the event handler.
3. Add your code for the new event handler in the Source Editor.

---

**Note:** If multiple events are of the same type, you can use the same handler for all of them. For example, you could set both `focusGained` and `focusLost` to use the `button1FocusChange` handler since they are both of the type `java.awt.event.FocusEvent`. You can also use the same handler for the same event on multiple components.

---



---

**Note:** You can set the code generation style for how the component event code is generated. Choose **Tools > Options**. Select the Miscellaneous pane and then select the **GUI Builder** tab. Set the **Listener Generation Style** property, one of Anonymous Innerclasses, One Innerclass, or Main Class.

---

#### To add multiple handlers for one event:

1. In the Navigator window, select the component for which you want to add multiple handlers.
2. Click the **Events** button at the top of the Properties window.
3. Select the event in the property sheet and click the ellipsis (...) button to display the Handlers dialog box. Click the **Add** button and fill out the form. Repeat these steps to add additional event handlers.

#### To remove event handlers:

1. In the Navigator window, select the component from which you want to remove the event handlers.
2. Click the **Events** button at the top of the Properties window.
3. Select the event in the property sheet and click the ellipsis (...) button to display the Handlers dialog box. Select the unwanted handler and click the **Remove** button.

Alternately, go to the Properties window and delete the name of the handler you want to remove.

4. In the **Handlers** dialog box, select the unwanted handler and click **Remove**.

When you remove an event handler, the corresponding code block is deleted. If more than one handler uses the same name and same block of code, deleting a single reference to the code does not delete the code itself. You must delete all references to the corresponding code block; a confirmation dialog box is displayed first.

For more information about how to handle events in your program, including information about the Java event model, see the *Creating a GUI with JFC/Swing* tutorial at:

<http://docs.oracle.com/javase/tutorial/uiswing/>

### 11.2.11 How to Modify GUI Source Code

The IDE automatically generates blue guarded blocks of code as you create your GUI form in the GUI Builder. Guarded text generated includes:

- Blocks of components' variable declarations.
- The `initComponents()` method, in which form initialization is performed. This method is called from the form's constructor and though it is not editable

manually, to affect the way it is generated, edit the Code properties in the component's property sheet.

- The header (and trailing brace) of all event handlers.

You can modify the way initialization code is generated and even write custom code to be placed within the initialization code.

#### **11.2.11.1 Modifying Code Generation for Form Components**

You can modify the way initialization code is generated for a component, form, or component property by editing its Code properties in the Properties window. In addition, you can write custom code and specify where it should be placed within the initialization code.

##### **To modify a form component's guarded block:**

1. In the Navigator window, select the component whose initialization code you want to edit.
2. Click the **Code** button at the top of the Properties window to view the Code properties.
3. Select the property you wish to edit and enter the desired value.

The IDE updates the selected component's guarded code block with the new value.

Alternatively, you can customize code generated for a component through the Code Customizer dialog box. To open this dialog box, right-click a component in the Design view of the GUI Builder and choose **Customize Code**.

#### **11.2.11.2 Setting Variable Modifiers for a Java Component**

You can set the Java language modifiers for a Java element using the Variables Modifier Property Editor dialog box.

##### **To modify a component variable:**

1. In the Navigator window, select the component whose variable modifier you want to edit.
2. Click the **Code** button at the top of the Properties window to view the Code properties.
3. Select **Variable Modifiers** and click the ellipsis (...) button.

The Variable Modifiers dialog box opens. You can edit the access modifiers or other modifiers as needed.

#### **11.2.11.3 Modifying Code Generation for a Property**

The IDE enables form component properties to be initialized in more ways than simply setting static values.

The IDE enables you to initialize property values from:

- A static value you define
- A component written to the JavaBeans architecture
- A property of another component on the form
- A call to a method of the form or one of its components. You can choose from a list of methods that return the appropriate data type.

- Code you define, to be included in the generated code

**To modify the initialization code for a component's property:**

Follow these steps to modify the initialization code generated for a component's property:

1. Select the component in the Navigator window.
2. Click the **Properties** button at the top of the Properties window.
3. Select the property for which you would like to modify the initialization code.
4. Click the ellipsis (...) button to bring up the Property Editor dialog box.
5. Select **Form Connection** from the **Select Mode** combo box.
6. In the Property Editor, select the type of initialization code you would like to add (Value, Bean, Property, Method Call, or User Code).
  - If you select Value or User Code, you must add a static value or your custom initialization code in the field provided.
  - If you select Bean, Property, or Method Call, you can select from a list of valid options.

The IDE adds the new code to the selected component's guarded block.

---

**Note:** You can also place custom code before or after a property's initializer. To do this, follow steps 1 through 4 above, and then click the **Advanced** button to bring up the Advanced Initialization Code dialog box. Type your custom pre-initialization code, post-initialization code, or both in the fields provided.

---

#### 11.2.11.4 Modifying GUI Form Code Outside of the IDE

In the IDE each form is comprised of two files:

- A .java file, which contains the form's Java source code.
- A .form file, which stores the information that is used to generate the .java file when you make changes to the form in the GUI Builder. This file does not need to be distributed with your application. If you delete this file, you can no longer use the GUI Builder to change the form.

You can edit the .java files using external editors (not while the form is being edited in the IDE), with the following exceptions:

- Do not modify the content of the `initComponents()` method. The body of this method is always regenerated when the form is opened in the IDE.
- Do not remove or modify any of the special comments the GUI Builder places in the source (`// GEN-...`). They are required for the form to open correctly. These comments are not visible inside the IDE's Source Editor.
- Do not modify the headers or footers of event handlers.

For more information, see [Section 11.2.8, "How to Edit Component Properties,"](#) and [Section 11.7, "Configuring the GUI Builder."](#)

## 11.2.12 How to Create a Multiple Document Interface (MDI) Application

The multiple document interface (MDI) model is similar to a traditional computer windowing system in that it includes a desktop above which additional windows float. In a JFC/Swing MDI application, the individual internal windows are all contained within a single enclosing window (i.e. desktop) which users can position, resize, minimize, and close.

### To create an MDI application:

1. Choose **File > New** to display the New wizard.
2. In the New File wizard's Project combo box, select the project for which you want to create the form. For more information see [Section 11.2.1, "How to Create a New Form."](#)
3. Expand the Java GUI Forms node and select one of the following templates:
  - **JFrame Form** builds an MDI application from scratch.

---

**Note:** You must also add a JDesktopPane component if you choose this template.

---

- **MDI Application** (in Sample Forms) creates a new MDI form with a JDesktopPane and predefined common menu items.
4. Click **Next**.
  5. On the wizard's Name and Location page, enter the form's name in the Class Name combo box, then specify the Location and Package. Click **Finish**. The IDE displays the new file in the Created File field.
  6. (Optional) If you chose the JFrame Form template, select the JDesktopPane node in the window's Swing category and click anywhere in the form.
  7. (Optional) Add JInternalFrame components to the JDesktopPane container by selecting JInternalFrame components from the Swing category in the window and click in the JDesktopPane container. Alternately, you can copy and paste JInternalFrame components to the JDesktopPane container in the Files or Project window. For more information, see [Section 11.2.3, "How to Add a Component to a Form."](#)

The IDE updates the form's layout and displays the new internal frames in the GUI Builder.

---

**Note:** You can add other components directly to the JDesktopPane container, such as a JTable or JSlider component. However, these have standard properties and users can't manipulate them as they might manipulate components in a JInternalFrame container.

---

---

**Note:** You can create separate JInternalFrame forms and add these to the JDesktopPane container programmatically at runtime.

---

### 11.2.13 How to Create Accessible Forms

To ensure that your GUI forms and the components contained within them meet accessibility requirements, adjust their accessibility properties. A GUI is considered accessible when it works with various assistive technologies, such as screen readers.

The following properties can be edited to aid accessibility:

- **Accessible Name.** Sets the name for the component. By default, the name is set to the component's text property value.
- **Accessible Description.** Sets the description for the component.
- **Accessible Parent.** Sets the name of the accessible parent component.

**To edit a form or component's accessibility properties:**

1. In the Navigator window, select the form or component whose accessibility properties you want to modify.
2. In the Properties window, click the **Properties** tab and scroll down to the Accessibility properties.
3. Click the ellipsis (...) button to open the Property Editor and then enter the desired value. Alternately, you can click the current property value to select it and enter a new value.

The IDE stores the updated accessibility information in the selected component.

For additional information, see [Section 11.2.1, "How to Create a New Form"](#) and [Section 11.2.3, "How to Add a Component to a Form."](#)

### 11.2.14 How to Preview a Form

To quickly test how your form is displayed when it is compiled and run, click the **Test Form** button in the GUI Builder toolbar. A dialog box is displayed with the components arranged as they would actually appear on your form.

When you click in a Testing Form dialog box, mouse events are delivered to the actual components and you can see the components "in action." For example, you can move sliders, type into text fields, and buttons look "pressed" when you click them, however, cross-component and event handling code is not executed.

**To test your form without compiling and running it:**

Click the **Preview Design** button located in the GUI Builder toolbar. The IDE displays the form in the Preview window enabling you to test the behavior of your GUI.

---

**Note:** If you update your form, close any open form Preview windows and click the **Preview Design** button again to see your changes.

---

## 11.3 Working with Layout Managers

Layout managers enable you to control the way in which visual components are arranged in GUI forms by determining the size and position of components within containers. This is accomplished by implementing the `LayoutManager` interface.

By default, new forms created with the GUI Builder use the `GroupLayout` layout manager. The IDE has special support for `GroupLayout` called Free Design. Free Design enables you to lay out your form using visual guidelines that automatically

suggest optimal alignment and spacing of components. As you work, the GUI Builder translates your design decisions into a functional UI without requiring you to specify a layout manager. Because Free Design employs a dynamic layout model, whenever you resize the form or switch locales the GUI adjusts to accommodate your changes without changing the relationships between components.

You can combine FreeDesign containers and containers using other layout managers together in the same form.

---

**Note:** GroupLayout was added to version 6 of the Java Platform. If you must deploy your application to version 5 of the Java Platform, you must use the version of GroupLayout that is in the Swing Layout Extensions library. You can set the version of GroupLayout in the property sheet for each form. With the form selected in the Design view, select the form's root node in the Navigator window, and change the Layout Generation Style property.

---

---

**Note:** New containers added to forms created in earlier versions of the IDE do not assume the FreeDesign layout manager in order to ensure code compatibility. However, it can be set manually in the Set Layout submenu.

---

## Other Layout Managers

You can use other layout managers in the IDE, which might be necessary if you are working with forms created in earlier versions of the IDE, or if you want your form to be compatible with Java Platform version 5 or earlier.

You can choose from the following Layout Managers in the IDE:

- **FlowLayout**

FlowLayout ( ) arranges components in a container like words on a page. It fills the top line from left to right until no more components can fit, continuing the same way on each successive line below.

- **BorderLayout**

BorderLayout ( ) arranges components along the edges or the middle of their container. Use BorderLayout to place components in five possible positions: North, South, East, West, and Center corresponding to the container's top, bottom, right and left edges and interior area.

- **GridLayout**

GridLayout ( ) places components in a grid of equally sized cells, adding them to the grid from left to right and top to bottom.

- **GridBagLayout**

GridBagLayout ( ) is a powerful layout manager that provides precise control over all aspects of the layout even when the container is resized, using a complex set of component properties called "constraints." It is particularly useful for multiplatform Java applications as it enables you to create a free-form layout that maintains a consistent appearance across platforms.

GridBagLayout places components in a grid of rows and columns in which grid cells do not all have to be the same size. In addition, components can span multiple rows, columns, or both. For more information about using

GridBagLayout see [Section 11.3.2, "How to Use the GridBag Customizer."](#)

- **CardLayout**

CardLayout ( ) provides a means of managing two or more components occupying the same display area. When using CardLayout each component is like a card in a deck, where all cards are the same size and only the top card is visible at any time. Since the components share the same display space, at design time you must select individual components using the Navigator window.

- **BoxLayout**

BoxLayout ( ) allows multiple components to be arranged either vertically or horizontally, but not both. Components managed by BoxLayout are arranged from left to right or top to bottom in the order they are added to the container. Components in BoxLayout do not wrap to a second row or column when more components are added or even when the container is resized.

- **AbsoluteLayout**

AbsoluteLayout ( ) is a layout manager that is provided with the IDE. AbsoluteLayout enables components to be placed exactly where you want them in the form, move them around in the IDE, and resize them using their selection borders. It is particularly useful for making prototypes since there are no formal limitations and you do not have to enter any property settings. However, it is not recommended for production applications since the fixed locations and sizes of components do not change with the environment.

- **Null Layout**

The Null Layout ( ) is used to design forms without any layout manager at all. Like the AbsoluteLayout, it is useful for making quick prototypes but is not recommended for production applications, as the fixed locations and sizes of components do not change when the environment changes.

For additional information see [Section 11.3.1, "How to Set the Layout Manager,"](#) [Section 11.3.2, "How to Use the GridBag Customizer,"](#) and [Section 11.3.3, "How to Use a Custom Layout Manager."](#)

### 11.3.1 How to Set the Layout Manager

When you create a new container, it is generally created using GroupLayout so that you can take advantage of the IDE's Free Design features. If necessary, you can change the layout of most containers using the window, GUI Builder, or Navigator window.

**To set the layout manager from the GUI Builder:**

1. Right-click the container whose layout you wish to change.
2. In the contextual menu, choose the desired layout from the **Set Layout** submenu.

The IDE applies the specified layout manager to the selected container.

**To set the layout manager from the Navigator window:**

1. Right-click the node for the container whose layout you wish to change.
2. In the contextual menu, choose the desired layout from the **Set Layout** submenu.

The IDE applies the specified layout manager to the selected container.

When you change layouts, the IDE remembers the properties of the discarded layout manager. If you then revert back to the previous layout manager, the form also returns to its prior state.

For more information see [Section 11.3, "Working with Layout Managers,"](#) [Section 11.3.3, "How to Use a Custom Layout Manager,"](#) and [Section 11.3.4, "How to Set Layout Properties."](#)

### 11.3.2 How to Use the GridBag Customizer

The GridBag customizer enables you to visually adjust the placement and constraints of components in a GridBagLayout. It includes a property sheet for GridBag constraints, buttons for adjusting the constraints, and a rough depiction of the layout of the components. The GUI Builder more closely reflects how the components look at runtime.

#### To use the GridBag customizer:

1. Add the components you require to your form and ensure the GridBagLayout is set for it.
2. To open the customizer, right-click the GridBagLayout node in the Navigator window and choose **Customize** from the contextual menu.
3. Drag the components in the right pane to reposition them as desired. As you drag a component, its Grid X and Grid Y properties change to reflect its new position.
4. Once the approximate layout of the components has been established, select a component and adjust its constraints as desired in the left pane. You can either enter the values directly or use the provided buttons to adjust the component's constraints.

---

**Note:** While editing you might need the **Redo**, **Undo**, **Pad**, and **Test Layout** buttons in the toolbar above the right pane.

---

5. Once you are satisfied with the layout, click **Close** to exit the customizer.

The IDE updates the edited components to reflect their new positions.

#### Adjustable constraints:

- **Grid X and Grid Y.** Fine-tune the component's horizontal and vertical position if necessary by setting its X and Y grid positions.
- **Grid Width and Grid Height.** Set Grid Width and Grid Height to specify how many grid positions are allocated for the component in each direction. Specify one of:
  - **An integer value** - the number of cells the component uses, (not the number of pixels)
  - **Remainder** - to make the component the last one in its row or column, using all remaining horizontal or vertical space
  - **Relative** - to specify that the component be the next to last one in its row or column
- **Fill.** The Fill constraint controls whether the component uses all of the allocated vertical or horizontal space (or both). Use the Grid Size buttons to adjust the Fill constraint. Any filled allocated space is marked with green in the right pane.
- **Internal Padding X and Y.** The internal padding settings enable you to increase the horizontal and vertical dimensions of the component. To adjust these enter numerical values for the properties.

- **Anchor.** The Anchor constraint controls the component placement in one of eleven positions within the allocated space (Center, North, North-West, and so on). This setting has no effect if there is no free space remaining for the component.

You can also adjust the Anchor constraint using the Grid Size buttons.

- **Weight X and Weight Y.** Adjust the weight settings to determine how much space a component should be given relative to other components in its row or column when the container window is resized. Generally, weight values range from zero to one. Components with larger weight values get more space allocated in their row or column when the window is resized.

Components with a weight value of zero always retain their preferred size for that dimension. If all the components in a row or column have a weight of zero, any extra space goes to the outer edges of the row or column and the components stay the same size.

- **Insets.** The Insets determine the minimum amount of external space on each of the four sides of the component. Values can be entered manually. As you change the insets, you see the inset area marked by a green background in the right pane.

---

**Note:** It is often helpful to sketch out the way you want your layout to look before you use the GridBag customizer.

---

For more information see [Section 11.3, "Working with Layout Managers,"](#) [Section 11.3.3, "How to Use a Custom Layout Manager,"](#) and [Section 11.3.4, "How to Set Layout Properties."](#)

### 11.3.3 How to Use a Custom Layout Manager

If you are unable to accomplish the result you require with the provided layout managers ([Section 11.3.1, "How to Set the Layout Manager."](#)), use a custom layout manager.

To integrate a custom layout manager into the IDE's GUI Builder it must:

- Implement the `java.awt.LayoutManager` interface.
- Be a JavaBeans component.
- Not use layout parameters (constraints) for individual components.

---

**Note:** If the layout manager does not meet the above requirements, you must create a special support class in order to add it to the GUI Builder.

---

#### To install a custom layout manager from the Files window:

1. Right-click the layout manager's class in the Files window and choose **Tools > Add** to from the contextual menu.
2. In the Select Category dialog box, select one of the categories and click **OK**.

The IDE adds the custom layout manager to the window.

For related information see [Section 11.3, "Working with Layout Managers."](#)

#### To install a custom layout manager that is packed in a JAR file:

1. Choose **Tools > Palette > Swing/AWT Components**.

2. In the Palette Manager, click **Add From JAR**.
3. In the Install Components to Palette wizard's file chooser, navigate to and select the JAR file for the layout manager.
4. Complete the Install Components to Palette wizard and click **Finish**.

The IDE installs the JavaBean and adds the custom layout manager to the window.

#### 11.3.4 How to Set Layout Properties

To modify the appearance of your forms adjust the general layout manager properties as well as properties specific to any components.

In the IDE you can modify:

- General layout properties which effect all components in a container, such as alignment of components and gaps between the components.
- Layout properties specific to a component that is managed by a particular layout manager and which apply to that component alone. These type of properties are also known as constraints.

##### To set general layout manager properties:

1. Select the layout manager's node in the Navigator window.
2. In the Properties window, select the property you want to edit and enter the desired value.

---

**Note:** The properties vary depending on the layout manager and that some layout managers do not have any properties.

---

The IDE updates all effected components' positions.

For more information see [Section 11.3, "Working with Layout Managers,"](#) and [Section 11.3.1, "How to Set the Layout Manager."](#)

##### To set layout properties of components:

1. Select the component in the Navigator window.
2. Click the **Properties** button at the top of the Properties window.
3. Scroll down to Layout properties, select the property you want to edit and enter the desired value.

The IDE automatically updates the component's layout behavior.

For more information see [Section 11.2.2, "How to Work with Containers."](#)

---

**Note:** Not all layout managers use constraints. FlowLayout, GridLayout, and BoxLayout) do not have Layout properties so component layout is determined by component order.

---

### 11.4 Adding a Bean to the Window

To expand the IDE add your own beans to the window for use in visual design. See [Section 11.2.3, "How to Add a Component to a Form."](#) Once the bean is installed, select

it in the window, compile it, and add the bean to your forms. You must compile a bean before it can be added to the window.

If the bean you add does not have an icon, the IDE assigns a default icon. To see the name of each installed bean in the window, position the pointer over the bean to view the tool tip.

---

**Note:** If you alter a JavaBeans component that is used in a form, those changes are not automatically reflected in the GUI Builder. To apply any changes you have made to a component and cause them to appear in the form, recompile the component. Then reload the form by pressing Ctrl+Shift+R in the GUI Builder or the Source Editor.

---

#### To add a bean to the window:

1. Choose Tools > Palette > Swing/AWT Components.
2. If you want to create a new category for the bean, right-click in the Palette window to open the context menu. Choose **Create New Category** and enter the desired name before you add the bean.
3. Click **Add from JAR**, **Add from Library**, or **Add from Project** and complete the wizard to add the bean. You must specify either a Jar, a library or a project that contain the corresponding bean's .class file that you want to add to the Window.

---

**Note:** To add a bean to the Palette window, right-click the bean's node in the Projects window and choose **Tools > Add To Palette**.

---

## 11.5 Working with Database Applications and Beans Binding

The following table outlines the basic process of binding properties between JavaBeans components.

**Table 11-2 Basic Beans Binding Procedure**

Task	Description
Create GUI forms	See <a href="#">Section 11.2.1, "How to Create a New Form."</a>
Add components	See <a href="#">Section 11.2.3, "How to Add a Component to a Form."</a> Add necessary components to the form, including custom beans for which you create bindings.
Bind component properties	<ol style="list-style-type: none"> <li>1. Right-click a component in the Source Editor Design view, or in the Navigator window, and choose a target property from the <b>Bind</b> menu. The Bind dialog box opens.</li> <li>2. In the <b>Binding</b> tab, specify the source for the binding.</li> <li>3. In the <b>Advanced</b> tab, specify any other customizations to the binding, if necessary. For example, some bindings require custom converters, validators, and code for handling unreadable or non-existent values from the source.</li> </ol> <p>For more on binding beans, see <a href="#">Section 11.5.1, "How to Bind Two Bean Properties,"</a> <a href="#">Section 11.5.2, "How to Bind Data to a Swing Components,"</a> and <a href="#">Section 11.5.3, "How to Use Special Binding Properties (Java Desktop Applications)." </a></p>

### 11.5.1 How to Bind Two Bean Properties

Once you have created a new Java form and added components to the form, you can generate code to bind component properties. Binding keeps the values of those properties synchronized.

#### To prepare components for binding:

Before binding component properties, do the following:

- Add components to a form in the GUI Builder.
- Determine which component is the source of the binding and which is the target.

The binding source is where a value for the property first originates. When binding in the GUI Editor, you initiate a binding on the target and then you declare the source in the Bind dialog box.

---

**Note:** Bindings can be two-way (read/write), so that changes in the target are automatically reflected in the source. However, the direction of the initial binding is always from the source to the target. You can set the update behavior of bindings on the Bind dialog box **Advanced** tab.

---

#### To bind two properties:

1. In the GUI Builder, right-click the component that is the target of the binding, select **Bind**, and choose the property that you want to bind from the **Bind** submenu.
2. From the **Binding Source** combo box, select the component that contains the property to which you want to bind.
3. From the **Binding Expression** combo box, select the property to which you want to bind.
4. Optionally, select the **Advanced** tab to further configure the binding. For example, you can change the update strategy, specify a validator or converter, and specify whether the binding source is null or unreadable.
5. Click **OK**.

### 11.5.2 How to Bind Data to a Swing Components

Once you have created a new Java form and added components to the form, you can generate code to bind those components to data. The IDE makes it easy to bind data to Swing JTable and JList components.

Before binding a component to a database, must have completed the following tasks:

- Connect to a database in the IDE.
- Add the component to a form in the GUI Builder.
- Create entity classes that represent the database tables to which you want to bind. Entity classes are special classes that use the Java Persistence API (JPA). Steps on creating the entity classes for binding data to a component are given below.

#### To create entity classes to represent the database to be bound to the JTable:

1. In the Projects window, right-click your project and choose **New > Other**, select the **Persistence** category, and select the **Entity Classes from Database** template.

2. In the Database Tables page of the wizard, select the database connection.
3. Once the Available Tables column is populated, select the tables that you want to use in your application and click **Add** to move them to the Selected Tables column. Click **Next**.
4. In the Entity Classes page of the wizard, make sure the Generate Named Query Annotations for Persistent Fields dialog box is selected.
5. Make any customizations that you want to make to the names of the generated classes and their location.
6. Click **Create Persistence Unit**.
7. In the Create Persistence Unit dialog box, make sure of the following things:
  - The selected Persistence Library is TopLink.
  - The selected Table Generation Strategy is "None."
8. Click **Finish**.

You should see nodes for the entity classes in the Projects window.

#### To bind the data to a **JTable** component:

1. Right-click the component in the GUI Builder and choose **Bind > elements**.
2. Click **Import Data to Form**. From the Import Data to Form dialog box, select the database table to which you want to bind your components. Click **OK**.
3. From the Binding Source combo box, select an item that represents the result list of the entity class. For example, if the entity class is called, Customer.java, the list object would be generated as customerList.
4. Leave the Binding Expression value as null.
5. In the **Display Expression** drop-down list, select the property that represents the database column that contains the values that you want to display in the list.
6. Select the **Advanced** tab to further configure the binding.
7. Click **OK**.

### 11.5.3 How to Use Special Binding Properties (Java Desktop Applications)

Where necessary, the beans binding library provides special synthetic properties for some Swing components that are missing from the components themselves. These properties represent data selections, such as a table's selected row, that are useful to bind to other properties.

**Table 11–3 Beans Binding Library Synthetic Properties**

Component	Property	Description
AbstractButton	selected	The selected state of a button.
JComboBox	selectedItem	The selected item of a JComboBox.
JSlider	value	The value of a JSlider; notifies of all changes.
	value_IGNORE_ADJUSTING	Same as "value" but does not notify of change while the slider is adjusting its value.

**Table 11–3 (Cont.) Beans Binding Library Synthetic Properties**

Component	Property	Description
JList	selectedElement	The selected element of a JList; notifies of all changes. If there is a JListBinding with the JList as the target, the selected element is reported as an element from the binding's source list. Otherwise, the selected element is reported as an object from the list's model. If nothing is selected, the property evaluates to null.
	selectedElements	A list containing the selected elements of a JList; notifies of all changes. If there is a JListBinding with the JList as the target, the selected elements are reported as elements from the binding's source list. Otherwise, the selected elements are reported as objects from the list's model. If nothing is selected, the property evaluates to an empty list.
	selectedElement_IGNORE_ADJUSTING	Same as "selectedElement" but does not notify of change while the list selection is being updated.
	selectedElements_IGNORE_ADJUSTING	Same as "selectedElements" but does not notify of change while the list selection is being updated.
	selectedElement	The selected element of a JTable; notifies of all changes. If there is a JTableBinding with the JTable as the target, the selected element is reported as an element from the binding's source list. Otherwise, the selected element is reported as a map where the keys are composed of the string "column" plus the column index and the values are the model values for that column. Example: {column0=column0value, column1=column1value, ...} If nothing is selected, the property evaluates to null.
JTable	selectedElements	A list containing the selected elements of a JTable; notifies of all changes. If there is a JTableBinding with the JTable as the target, the selected elements are reported as elements from the binding's source list. Otherwise, each selected element is reported as a map where the keys are composed of the string "column" plus the column index and the values are the model values for that column. Example: {column0=column0value, column1=column1value, ...} If nothing is selected, the property evaluates to an empty list.
	selectedElement_IGNORE_ADJUSTING	Same as "selectedElement" but does not notify of change while the table selection is being updated.
	selectedElements_IGNORE_ADJUSTING	Same as "selectedElements" but does not notify of change while the table selection is being updated.
	text	The text property of a JTextField, JTextArea, and JEditorPane; notifies of all changes (including typing).
	text_ON_FOCUS_LOST	The text property of a JTextField, JTextArea, and JEditorPane; notifies of change only when focus is lost on the component.
JTextComponent (including its sub-classes JTextField, JTextArea, and JEditorPane)	text_ON_ACTION_OR_FOCUS_LOST	The text property of a JTextField, JTextArea, and JEditorPane; notifies of change only when the component notifies of actionPerformed or when focus is lost on the component.

For related information, see [Section 11.2.3, "How to Add a Component to a Form,"](#) [Section 11.5.2, "How to Bind Data to a Swing Components,"](#) and [Section 11.5.1, "How to Bind Two Bean Properties."](#)

#### 11.5.4 How to Convert Values Between Source and Target Properties (Java Desktop Applications)

When you bind the values of two properties of two objects, you must sometimes convert the values between different types.

The beans binding library contains converters for some common conversions. For other conversions, you must provide a custom converter.

Below is a list of conversions which do not need converter:

- BigDecimal to String, String to BigDecimal
- BigInteger to String, String to BigInteger
- Boolean to String, String to Boolean
- Byte to String, String to Byte
- Char to String, String to Char
- Double to String, String to Double
- Float to String, String to Float
- Int to String, String to Int
- Long to String, String to BigDecimal
- Short to String, String to Short
- Int to Boolean, Boolean to Int

To write a custom converter, create a class that extends `org.jdesktop.beansbinding.Converter`. Your class needs to override the `convertForward(S value)` and `convertReverse(T value)` methods. `convertForward(S value)` converts a value from the source type to the target type. `convertReverse(T value)` converts a value from the target type to the source type.

#### **To use a custom converter in a binding:**

1. Right-click the converter class in the Projects window and choose **Compile File**.
2. Drag the converter from the Projects window to the Design view of your form. The converter is added to your form as a bean.
3. Right-click the target of your binding and choose **Bind > TargetProperty**.
4. In the Bind Dialog box, select the **Advanced** tab.
5. From the **Converter** drop-down list, choose the converter you have added to the form.
6. Click **OK**.

---

**Note:** You can also add the conversion code directly. Click the ellipsis (...) button, and select **Custom Code** from the **Select Converter Property Using** drop-down list.

---

For related information, see [Section 11.5.5, "How to Validate Target Value Changes in Bindings \(Java Desktop Applications\)"](#), [Section 11.5.2, "How to Bind Data to a Swing Components,"](#) and [Section 11.5.1, "How to Bind Two Bean Properties."](#)

### **11.5.5 How to Validate Target Value Changes in Bindings (Java Desktop Applications)**

When you bind the values of two properties of two objects, you sometimes must validate any changes to the target property before they are written back to the source (such as a database). To validate a target, specify a validator that extends `org.jdesktop.beansbinding.Validator`.

To write a custom validator, create a class that extends `org.jdesktop.beansbinding.Validator`. Your class needs to implement the `validate(T value)` method. For a valid value, `validate(T value)` returns null. For an invalid value, it returns a `Result` object describing the problem for the invalid value.

#### To use a custom validator in a binding:

1. Right-click the validator class in the Projects window and choose **Compile File**.
2. Drag the validator from the Projects window to the Design view of your form.  
The validator is added to your form as a bean.
3. Right-click the target of your binding and choose **Bind > TargetProperty**.
4. In the Bind Dialog box, select the **Advanced** tab.
5. From the **Validator** drop-down list, choose the validator you have added to the form.
6. Click **OK**.

---

**Note:** To add the validation code directly, click the ellipsis (...) button. From the **Select Validator Property Using** drop-down list, select **Custom Code**.

---

For additional information see [Section 11.5.1, "How to Bind Two Bean Properties,"](#) [Section 11.5.2, "How to Bind Data to a Swing Components,"](#) and [Section 11.5.4, "How to Convert Values Between Source and Target Properties \(Java Desktop Applications\)."](#)

## 11.6 Deploying GUI Applications

In order for the applications that you create to work outside of the IDE, you might have to include some extra JAR files when you deploy the application.

The `swing-layout-1.0.3.jar` file might be needed by your deployed applications. This library contains the various layout-related extensions, such as the `GridLayout` layout manager. This library is included in version 6 of the Java Platform, so you do not need to package it with your application if you are deploying it to environments that have version 6 of the JRE.

---

**Note:** As of JDK 6, the Beans Binding library is not part of the Java Platform.

---

You can find these JAR files in the following folders on your system:

- `NetBeans_installation_folder/java/modules/ext/` (for the Beans Binding library)
- `NetBeans_installation_folder/platform/modules/ext/` (for the Swing Layout Extensions library)

### 11.6.1 Preparing a GUI Application for Distribution

To ensure that your GUI application can reference these libraries at runtime, the IDE automatically copies the library JAR files (and any other JAR files on the project's classpath) to the `dist/lib` folder whenever you build the project. The IDE also adds each of the JAR files to the `Class-Path` element in the application JAR's `manifest.mf` file.

---

**Note:** If your application does not make use of the support of one of these libraries, that library is not included in the dist/lib folder.

---

**To prepare your GUI application for distribution outside of the IDE:**

Zip the project's dist folder (including the lib folder) into a ZIP archive.

## 11.6.2 Running a Standalone GUI Application

Once you have distributed an archive of your GUI application, your application can be run outside of the IDE from the command line.

**To run a standalone GUI application from the command line:**

- Navigate to the project's dist folder.
- Type the following:

```
java -jar jar_name.jar
```

For a discussion of GUI Builder capabilities, see [Section 11.2, "Working with the GUI Builder."](#)

## 11.7 Configuring the GUI Builder

You can adjust how the IDE's GUI Builder generates code and how the GUI Builder design looks.

---

**Note:** If you want to edit settings for a particular form, select the form's root node in the Navigator window and click the **Code** button in the Properties window. For additional information see [Section 11.2.11, "How to Modify GUI Source Code."](#)

---

**To configure GUI Builder settings for all projects:**

1. Choose **Tools > Options** from the main window.
2. Click the **Java** category.
3. Click the **GUI Builder** tab to display the editable GUI Builder Settings.
4. Select and edit the property you wish to change.

The IDE applies the new settings to the GUI Builder.

For more on how to adjust a component's appearance and behavior, see [Section 11.2.8, "How to Edit Component Properties,"](#) and [Section 11.3.4, "How to Set Layout Properties."](#)



# 12

---

## Developing Web Applications

This chapter describes how to build a Java EE-based web application and includes details on support in the IDE for JSF 2.1 (Facelets), JSPs, and Servlets.

This chapter contains the following sections:

- [About Developing Web Applications](#)
- [Creating Web Application Projects](#)
- [Working with JSP Files](#)
- [Working with Tag Libraries](#)
- [Working with Applets](#)
- [Working with Servlets](#)
- [Using Filters](#)
- [Using Web Application Listeners](#)
- [Using WebSocket Endpoints](#)
- [Configuring a Web Application](#)
- [Deploying a Web Application](#)
- [Debugging a Web Application](#)
- [Profiling a Web Application](#)

### 12.1 About Developing Web Applications

A web application is an application written for the Internet, including those built with Java technologies such as JavaServer Pages and servlets, as well as those built with non-Java technologies such as CGI and Perl. Web Applications often include frameworks such as JavaServer Faces, Spring, Hibernate, and Struts.

You can create a web application by writing and editing Java source code in the Source Editor. Configure the application yourself. Add the JavaServer Faces, Struts, or Spring Web framework to improve the design of your application.

A web application roughly corresponds to the J2EE term *web application module*. This is a deployable unit that consists of one or more web components, other resources, and web application deployment descriptors, contained in a hierarchy of directories and files in a standard web application format.

The following steps outline the basic process of working with web applications:

1. Create a project.

1. Register the target server for your application.
  2. Create the project using the best template for your programming needs. While creating the project, you can let the IDE include libraries and configuration files for JavaServer Faces, Struts, or Spring.
  3. Configure the classpath for the project. You can add a JAR file, a library, or an IDE project to the classpath.
2. Create web components.
1. Choose **File > New File** or right-click any project and choose **New > Other**.
  2. Under **Categories**, select **Web**. Under **File Type**, select the web component that you would like to create.
  3. Click **Next** and follow the instructions in the wizard.
3. Edit web components.
1. In the **Projects** window or **Files** window, double-click the web component that you would like to edit.
  2. Use the **Source Editor** to edit the web component.
4. (optional) Connect to database.
1. If the database connection is not visible in the right-click the **Databases** node and choose **New Connection**, then add the database connection in the **New Database Connection** dialog box.
  2. If the database node is broken, right-click the node and choose **Connect**, and then connect to the database.
5. Deploy and run application.
- Choose **Run > Run Project** or right-click the project and choose **Run**. The project is compiled, and new and changed files are copied from the project's build directory to the WAR's deployment directory. The deployed application then opens in a browser.
6. Debug web application.
1. Set breakpoints or watches in your code.
  2. Right-click the project node in the **Projects** window and choose **Debug**. Alternatively, set the project as the main project and choose **Debug > Debug Main Project**. You can also attach a running process to the debugger.
3. Step through the program execution.
  4. Choose **Debug > Finish Debugger Session** when you are finished debugging.

## 12.2 Creating Web Application Projects

Web applications have a defined folder structure.

When you create a web project, the IDE does the following:

- Displays the new project in the **Projects** window.
- Opens a page for you to edit

- Puts all the files in the \$HOME/NetBeansProjects/project-name directory on a UNIX® or Mac OS X system or in the \Documents and Settings\username\My Documents\NetBeansProjects\project-name folder on a Microsoft Windows system.

---

**Note:** You can also specify where the files are located.

---

After you create the project, you can view the files in the **Projects** window or the **Files** window.

When you import a web project from an earlier version of the IDE, the project is structured according to Java BluePrints guidelines. A web application must also contain a deployment descriptor file (WEB-INF/web.xml), and can contain one or more web components.

---

**Note:** You can add a framework to a web application project after creating it. To do so, choose **Tools > Ant Libraries** to access the Ant Library Manager.

---

### 12.2.1 How to Create a Web Application Project

In the IDE, you create applications and modules in projects. The web application project is the project that you use to create web applications and web application modules.

**To create a web application project:**

1. Choose **File > New Project**.
2. From the Java Web category, select one of the following project templates:
  - **Web Application.** Creates an empty web application in a standard project.
  - **Web Application with Existing Sources.** Imports an existing web application into a standard project.
  - **Web Free-Form Project.** Imports an existing web application into a free-form project.
3. Follow the steps in the rest of the wizard.

### 12.2.2 Setting Up a Web Project Based on Existing Sources

For web projects developed outside of NetBeans, you use a **Web Application with Existing Sources** template in the **New Project** wizard to make a NetBeans project. In the wizard, you identify the location of the sources and specify a location for the NetBeans project metadata. You then use the **Project Properties** dialog box to configure the project.

**To set up a NetBeans project for an existing web application:**

1. Choose **File > New Project**.
2. Choose **Java Web > Web Application with Existing Sources**. Click **Next**.
3. In the **Name and Location** page of the wizard, follow these steps:
  - In the **Location** field, enter the folder that contains the web application's source root folders and web page folders.

- Type a project name.
  - (Optional) Change the location of the project folder.
4. (Optional) Select the **Use Dedicated Folder for Storing Libraries** checkbox and specify the location for the libraries folder. For more information on this option see [Section 6.6.1, "How to Share a Library."](#).
5. (Optional) Select the **Set as Main Project** checkbox. When you select this option, keyboard shortcuts for commands such as **Clean and Build Main Project** apply to this project.
6. Click **Next** to advance to the **Server and Settings** page of the wizard.
7. (Optional) Add the project to an existing enterprise application.
8. Select a server to which to deploy. If the server that you want does not appear, click **Add to register the server in the IDE**.
9. Set the source level to the Java version on which you want the application to run.
10. (Optional) Adjust the context path. By default, the context path is based on the project name.
11. Click **Next** to advance to the **Existing Sources and Libraries** page of the wizard.
12. Verify all of the fields on the page, such as the values for the Web Pages Folder and Source Package Folders.
13. Click **Finish**.

## 12.3 Working with JSP Files

The JSP Editor shows you the JavaServer™ Pages (JSP) code generated by the IDE for the page you are editing. The JSP code is in XHTML format, and follows standard JSP conventions for the XML representation of a JSP page. The code uses the JavaServer Faces tag libraries to declare components, event handlers, validators, and so on. In addition, there are expressions written in the JavaServer Faces expression language (JSF EL).

When you open a JSP file in the IDE, a Palette is displayed on the right side of the Source Editor. Code snippets for many of the most common JSP tags are provided -- grouped according to their function: Use Bean, Get Bean Property, Set Bean Property, JSTL Choose, JSTL If, JSTL For Each.

To use an item in the Palette, drag it from the Palette into the Source Editor and drop it exactly where you want the tags to appear. A dialog box appears. You can fill in values for the item's standard properties, click **OK**, and then the IDE generates the tags with your specified values.

The IDE provides smart-case code completion for JSP, HTML, and customized tags. When you start writing a tag, the IDE offers code completion in upper case or lower case, depending on the case you are using.

---

**Note:** To enable code completion for a JavaBean class, the JSP file must have a valid *jsp:useBean* directive, the class must be in a package, and the class must be available from the *src* folder or from a library that is included in the compilation classpath. To enable code completion for tags in tag libraries, the JSP file must have a valid *taglib* directive for the library and the library must be included in the compilation classpath.

---

Other examples of additional JSP support are the highlighting of JSP tags, JSP directives, and EL expressions, code templates, code folding, the display of matching JSP tags, JSP delimiters (<>), and EL delimiters ({}), and code navigation features such as hyperlinking for JSP identifiers. For example, you can jump to the tag source file that defines a tag referenced in a JSP file if you simultaneously hold down the Ctrl key and move the mouse over the tag, as illustrated in [Section 12.3.4, "How to Access a Custom Tag from a JSP Page"](#).

### 12.3.1 How to Create a JSP File

Creating JSP files is similar to creating Java files. The Source Editor provides support for JSP, including code completion for the following JSP elements:

- JSP tags
- JavaBeans that are defined with the <jsp:useBean> tag
- Attributes that are defined in tag files
- Java code in scriptlets

**To create a JSP file:**

1. From the Projects window or Files window, right-click the project's node and choose **New > Other** from the pop-up menu.
2. Under **Categories**, select Web. Under **File Types**, select JSP. Click **Next**.
3. Type the name of your JSP. Do not include a filename extension as this is added automatically when the file is created.
4. Click **Browse** to select a folder to house your JSP file. By default, the JSP file is created in the web subfolder.
5. Select the type of JSP file that you want to create:
  - **JSP File (Standard Syntax)**. Standard JSP pages are written using the standard JSP syntax. Standard JSP pages typically use the .jsp extension.
  - **JSP Document (XML Syntax)**. JSP documents are written using the JSP document syntax, which is well formed XML. JSP documents typically have the .jspx extension. Two examples of the advantages that JSP documents have over standard JSP pages are:
    - You can edit, verify, and view them with XML capable tools, such as the IDE's Source Editor.
    - You can transform them using XSLT tools, such as the IDE's XSL Transformation command.
6. (Optional) Click the **Create as a JSP Segment** checkbox. A segment is a file that contains a fragment of JSP text for inclusion by standard JSP pages and JSP documents. JSP segments typically use the .jspx extension.
7. Click **Finish**. The IDE opens the JSP file for editing in the Source Editor.

### 12.3.2 How to Set Character Encoding

A character encoding maps a character set to units of a specific width and defines byte serialization and ordering rules. Many character sets have more than one encoding. For example, Java programs can represent Japanese character sets using the EUC-JP or Shift-JIS encodings, among others. Each encoding has rules for representing and

serializing a character set. Two of the more popular character encodings are the following:

- **The ISO 8859 series.** This series defines 13 character encodings that can represent texts in dozens of languages. Each ISO 8859 character encoding can have up to 256 characters. ISO 8859-1 (Latin-1) comprises the ASCII character set, characters with diacritics (accents, diaereses, cedillas, circumflexes, and so on), and additional symbols.
- **UTF-8 (Unicode Transformation Format, 8-bit form).** A variable-width character encoding that encodes 16-bit Unicode characters as one to four bytes. A byte in UTF-8 is equivalent to 7-bit ASCII if its high-order bit is zero; otherwise, the character comprises a variable number of bytes. UTF-8 is compatible with the majority of existing web content and provides access to the Unicode character set. Current versions of browsers and E-mail clients support UTF-8. In addition, many new web standards specify UTF-8 as their character encoding. For example, UTF-8 is one of the two required encodings for XML documents (the other is UTF-16).

To produce an internationalized web application, you need to encode the following:

- **Request Character Encoding.** The character encoding in which parameters in an incoming request are interpreted. This encoding converts parameters to string objects.
- **Page Character Encoding.** The character encoding in which the JSP file is written. Unless the page character encoding is set correctly, the JSP parser, web container, or web server that reads the page cannot understand the characters before they do anything with them, such as translating the JSP file into a servlet. Page character encoding is used for the rendering of JSP files only if the response character encoding has not been set separately.
- **Response Character Encoding.** The character encoding of the textual response generated by a web component. This lets you control the encoding that the page uses when it is sent to the browser. The web page encoding must be set appropriately so that the characters are rendered correctly for a given locale.

All modern web browsers understand UTF-8, so that is a safe encoding to pick for the response. In the IDE, it is a good encoding at the page level too. This is why UTF-8 is the default page character encoding and also the default response character encoding for JSP files created in the IDE.

The request encoding is the character encoding in which parameters in an incoming request are interpreted. Many browsers do not send a request encoding qualifier with the Content-Type header. In such cases, a web container will use the default encoding--utf-8--to parse request data. For detailed information on request character encoding, see the references at the end of this topic.

Page character encoding is the encoding in which the JSP source file is written. The JSP 2.0 specification distinguishes between two syntaxes when detecting the page character encoding:

- For files in standard JSP syntax, encoding is detected by looking at two primary sources of information: First in the deployment descriptor (the `web.xml` file) for a `page-encoding` element in a `jsp-property-group` whose URL pattern matches the file; then for a `pageEncoding` attribute in the page itself. If neither is present, the `charset` of a JSP file's `contentType` attribute is used, or the ISO 8859-1 character encoding is used as the ultimate fallback.
- For files in JSP document syntax, the encoding is detected as described in the XML specification; this means that UTF-8 is the default and any other encoding must be declared in the XML declaration at the beginning of the file.

**To set the page character encoding for a file in standard JSP syntax:**

1. Create a JSP file that uses standard JSP syntax. Right-click the JSP file and choose **Properties**. Note that the **Encoding** property is set to the encoding for the project. This is the JSP file's page encoding. You cannot change the page encoding in the Properties sheet. The place where you set the page encoding depends on whether you want to set it for an individual JSP file or for a group of JSP files together. The following step guides you through the setting of the page encoding for JSP files.

2. Do one of the following to change the JSP file's page encoding:

- **Set the page character encoding for an individual JSP file.** Double-click the JSP file so that it opens in the Source Editor. The default `pageEncoding` attribute of a JSP file created in the IDE is as follows:

```
<%@page pageEncoding="UTF-8"%>
```

You can change the page encoding in the JSP file and save it. Note that the IDE warns you if you try to save a character set that is not valid for JSP pages.

Alternatively, set the page encoding in the `contentType` attribute of the `page` directive instead. The default `contentType` attribute of a JSP file does not contain a charset value, because the `pageEncoding` attribute handles page encoding by default. However, you can add a charset value as follows:

```
<%@page contentType="text/html; charset=UTF-8"%>
```

- **Set the page character encoding for groups of JSP files.** Expand the Web Pages node, then the WEB-INF node, and then double-click the web.xml file. Click Pages at the top of the editor and then click the JSP Property Groups header to open the JSP Property Groups section. Use the JSP Property Groups section to add, remove, and view a web application's JSP property groups. A JSP property group is a set of properties defined for a group of JSP files within a web application. One of the properties you can set here is the page encoding value for a group of JSP files.

---

**Note:** Only if the other two are absent is the `contentType` attribute of a JSP file's `page` directive used as the page encoding. If none of these are provided, utf-8 is used as the page encoding. A translation-time error results if you define the page encoding with one value in the JSP property group and then give it a different value in an individual JSP file's `pageEncoding` directive.

---

**To set the page character encoding for a file in JSP document syntax:**

1. Create a JSP file that uses XML syntax, see [Section 12.3.1, "How to Create a JSP File."](#) Right-click the JSP document and choose **Properties**. Note that the **Encoding** property is set to the encoding for the project. This is the JSP document's page encoding. You cannot change the page encoding in the Properties sheet. For JSP documents, you can change the page character encoding in one place only:
  - The `encoding` attribute of an XML declaration at the beginning of the file. The XML declaration is also known as the XML prolog.

---

**Note:** The page encoding for JSP documents may also be *described* in an individual JSP document's page directive or in a JSP property group, as long as the values described there match the value derived from the XML prolog. It is a translation-time error to specify different encodings in the XML prolog and in the declarations in a JSP document or JSP property group. If the XML prolog does not specify an encoding, the UTF-8 encoding is derived instead.

---

2. Double-click the JSP file so that it opens in the Source Editor. The default XML declaration of a JSP document created in the IDE includes an encoding attribute as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
```

You can change the encoding in the XML declaration and save the JSP document.

The response character encoding is the encoding of the textual response generated by a web component. The response character encoding must be set appropriately so that the characters are rendered correctly for a given locale. An initial response character encoding for a JSP page is set from the same sources as for page character encoding:

- The charset value of a contentType attribute in a page directive.
- The pageEncoding attribute of a page directive.
- The <page-encoding> tags of a web.xml file's JSP property group.

However, note that the charset value of a contentType attribute is the place where you are recommended to set the response character encoding. The other two places are meant for the page character encoding; these are only used for the response character encoding when no specific response character encoding has been defined.

#### To set the response character encoding for a JSP file:

1. Double-click the JSP file so that it opens in the Source Editor. The default pageEncoding attribute of a JSP file created in the IDE is as follows:

```
<%@page pageEncoding="UTF-8"%>
```

You can change the page encoding in the JSP file and save it. Note that the IDE warns you if you try to save a character set that is not valid for JSP pages. The place where you set the response character encoding depends on whether you want it to be the same as the page character encoding or not.

2. Do one of the following:

- **Set a distinct response character encoding.** Use the contentType attribute of the page directive. The default contentType attribute of a JSP file created in the IDE does not contain a charset value, because the pageEncoding attribute handles page encoding by default. However, you can add a charset value as follows:

```
<%@page contentType="text/html; charset=UTF-8"%>
```

- **Use the page character encoding as the response character encoding.** Do not add a charset value to the page directive's contentType attribute. In the absence of the charset value, the encoding specified for the page by means of the page directive's pageEncoding attribute is also that of the response.

### 12.3.3 How to Edit a JSP File

Editing JSP files is similar to editing Java files. The Source Editor provides support for JSP tags.

To format selected code, right-click in the Source Editor and choose **Format** from the context menu. If no code is selected, the IDE acts as if the whole file has been selected. The IDE indents nested tags only if both the start and end parent tags are in the selected area. Note that this action does not format scriptlets.

### 12.3.4 How to Access a Custom Tag from a JSP Page

The JavaServer Pages technology provides a set of standard action elements for performing actions on information. The `<jsp:getProperty>` element is an example of a commonly used action element. You can extend the set of action elements through custom tags that are defined in tag libraries, such as the JSTL tag library.

This topic shows how to use a tag from a tag library JAR file, and then shows how to use a tag from a tag library folder that contains tag files.

#### To access a custom tag from a tag library JAR file:

1. Ensure that the tag library's JAR file is on the web application's classpath, as described in [Section 6.2.3.1, "Managing the Classpath"](#).
2. Before referencing one of its tags in a JSP file, add a taglib directive with a `uri` and `prefix` attribute to the JSP file: `<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>`.
  - The `uri` attribute shows the location of the tag library and should be unique within the web application. The `uri` can either be the URI specified by the TLD file, or the `uri` can be the location of the TLD file in the `WEB-INF` folder, if the `uri` element is missing in TLD file. When you type the `uri` attribute, press Ctrl-Space after you type the first quotation mark. The IDE shows the list of URIs for available tag libraries.
  - The `prefix` attribute is used to identify tags from the library. The TLD file usually recommends a prefix, but you can use any prefix you want.
3. At any point after the taglib directive, you can use the tag prefix to reference tags from the tag library. For example:

```
<c:if test="${param.sayHello}">
    Hello ${param.name}!
</c:if>
```

#### To access a custom tag from a tag library folder:

1. Ensure that the tag library's JAR file is on the web application's classpath, as described in [Section 6.2.3.1, "Managing the Classpath"](#).
2. Before referencing one of its tags in a JSP source file, add a taglib directive to the file:

```
<%@ taglib tagdir="/WEB-INF/tags/" prefix="a" %>
```

- The `tagdir` attribute shows the location of the tag library within the web application.
- The `prefix` attribute is used to identify tags from the library. The tag file usually recommends a prefix, but you can use any prefix you want.

3. At any point after the taglib directive, you can use the tag prefix to reference tags from the tag library. For example:

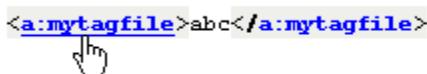
```
<a:mytagfile>abc</a:mytagfile>
```

**To quickly locate a tag's source file:**

1. Hold down the Ctrl key and, at the same time, move your mouse over a tag in the JSP file.

Figure 12–1 shows that the Source Editor displays the tag as a hyperlink and that the cursor changes to a hand symbol.

**Figure 12–1 Tag in Source Editor**



2. Click the hyperlink.

The tag file referenced by the tag opens in the Source Editor.

### 12.3.5 How to Access an Applet from a JSP Page

**To access an applet from a JSP page:**

1. Ensure that the applet is correctly packaged in the web application.
2. Define the applet in the JSP file by using the following applet tag:

```
<applet code="mypkg.MyApplet" archive="JavaAppletLibrary.jar" />
```

- mypkg.MyApplet is the full classname to your applet
- JavaAppletLibrary.jar is the JAR file built from the Java class library project where your applet was created.

### 12.3.6 How to Compile a JSP File

When you compile a JSP file you can detect syntax problems before you execute the file on a server. Compilation also translates the JSP file into a servlet. Therefore, compilation discovers syntax errors that occur at translation time and at compile time. If a JSP file references a tag file, the referenced tag file is compiled with the JSP file.

**To compile a JSP file:**

1. Do one of the following:

- **Compile a single file.** In the Projects window, right-click the JSP file and choose **Compile File** from the context menu.
- **Compile selected files.** In the Projects window, use the Ctrl key and the Shift key to select the files that you want to compile, right-click the selection and choose **Compile File** from the context menu.
- **Compile all JSP files in a project.** In the Projects window, right-click the project node, choose **Properties**, click **Compiling**, and select the **Test compile all JSP files during builds** checkbox. Close the **Project Properties** dialog box, right-click the project node, and choose **Build** from the context menu.

---

**Note:** By default, the IDE does not compile JSP files while building a project. This is because JSP compilation can take a long time because JSP files have to be translated to servlets before they can be compiled.

---

2. In the **Output** window, click an error to jump to the source of the error in the Source Editor.

### Troubleshooting

The following list shows some typical compilation messages and possible solutions:

- **Invalid expression.** Might be caused by an unmatched curly brace ({}). Look for EL syntax highlighting that extends past the EL expression. If the ending brace is missing, the Source Editor automatically highlights the next beginning brace as an error. When you select a brace, the Source Editor highlights the matching brace.
- **Missing equal symbol.** This error can be caused by a missing quote mark (""). Look for code that has the same color as the text or text that has the same color as the code. Look for tag highlighting that extends past the tag's end delimiter.
- **Missing mandatory attribute.** Might be caused by a misspelled attribute. Use the code completion feature to verify the correct spelling.
- **Unterminated tag.** Can be caused by a missing percent sign (%) in the directive's closing delimiter. Look for an end delimiter. that is a different color from the start delimiter.
- **Tag must be empty.** Check for a missing forward slash (/) in the tag's closing delimiter.

Forgetting to delimit EL expressions with curly braces ({{}}) is a common error that cannot be caught through compilation, because the text is valid syntax. To catch this type of error, look for expressions that are not highlighted with the color used for EL expressions.

If you see the following compilation output, there might be an internal cache problem:

```
java.lang.NoClassDefFoundError: javax/servlet/jsp/tagext/SimpleTagSupport
```

You might need to reboot the IDE to be able to compile the JSP file.

### 12.3.7 How to View a JSP File's Servlet

When you execute a JSP file, the server translates the contents of the JSP file into a Java servlet and compiles the servlet. You can view, but not edit, the translated servlet source in the Source Editor. You cannot set breakpoints in the servlet source code. You can set breakpoints in the JSP file only.

#### To view the servlet generated from a JSP file:

1. Run the JSP file, see [Section 12.3.9, "How to Run a JSP File"](#).
2. In the **Projects** window, right-click the JSP file and choose **View Servlet**.

---

**Note:** The View Servlet menu item is enabled only if the target server has successfully executed the JSP file and the target server supports the viewing of translated servlets.

---

The servlet appears in the Source Editor. This servlet is based on the state of the JSP file when the file was last executed. If you have changed the JSP file but have not executed it, the servlet does not reflect the changes. While this file is opened, if you execute the JSP file or if you change the target server in the Run section of the Project Properties dialog box, the file changes to reflect the latest translated servlet on the selected server.

### 12.3.8 How to Pass Request Parameters

You can pass request parameters in URL query string format to JSP pages and servlets from the IDE. Specifying input data in this fashion provides a useful aid in testing for expected JSP and servlet output.

**To specify parameters for a JSP page:**

1. In the **Projects** window or **Files** window, right-click the JSP file and choose **Properties**.
2. In the **Request Parameters** dialog box, type the parameters in the URL query string format.

URL query string is a string appended to the URL for passing parameter values in a GET request. The query string must begin with a question mark (?). The parameters must be in name/value pairs, with the pairs separated by ampersands (&). The names and values must be URL-encoded. For example, white space is encoded as +. The following URL shows an example of a query string:  
`http://www.myapp.com/sayhello?name=Fido+Filbert&type=Dog.`

3. Click **OK**.
4. Click **Close** to exit the **Properties** window.

The IDE saves the parameters and automatically passes them to the JSP or servlet the next time you select the file and choose **Run File** from the context menu.

---

**Note:** The HTTP Monitor enables you to monitor data flow on the server.

---

### 12.3.9 How to Run a JSP File

When you run a JSP file, the IDE displays the file in the IDE's default web browser. If the JSP file that you run from the IDE has not been compiled, or if it has changed since it was last compiled, the IDE automatically compiles it before executing it. If the server or the IDE's default web browser have not been started, the IDE automatically starts it.

**To run a JSP file:**

1. (Optional) Define request parameters to pass to the JSP file.
2. Select the JSP file in the Source Editor or Projects window.
3. Choose **Run > Run File > Run "filename.jsp"** from the main menu. If there are no errors, the server displays the file in the IDE's default web browser.

---

**Note:** If you are using a free-form project, this command is disabled by default. You have to write an Ant target for running the currently selected file in the IDE and hook it up to the Run Class command. For a full guide to configuring free-form projects, see  
<https://netbeans.org/kb/articles/freeform-config.html>

---

### Troubleshooting

If you get "File not found" errors when you execute a JSP file, see [Section 23.2.2, "How to Change the Default Web Browser"](#) for a possible solution.

## 12.4 Working with Tag Libraries

Rather than writing action code directly in your JSP file, you can store common functions in a tag library and implement the functions using simple tags. This practice makes the JSP file more readable and isolates the JSP file from any underlying implementation changes.

Tag libraries can be in one of the following forms:

- Java class tag handlers or JSP tag files (or both) bundled with a TLD (tag library descriptor) into a JAR file.
- A tag library folder containing tag files. The tag files can use standard JSP syntax or JSP document syntax.
- A tag library folder containing tag handlers. The tag handlers must be written in the Java programming language.

The IDE is bundled with the JSTL (JSP Standard Tag Library), which is a set of compatible tag libraries that you can use in your JSP pages. The JSTL supports iteration, control-flow, text inclusion, and formatting features, as well as XML manipulation capabilities. The JSTL also supports an expression language to simplify page development, as well as extensibility mechanisms that allow you to integrate your own custom tags with JSTL tags.

For a complete description of the JSTL, including documentation and tutorials, see <http://jakarta.apache.org/taglibs/doc/standard-doc/GettingStarted.html>.

---

**Note:** When the JSTL tag library exists in the web application's WEB-INF/lib folder and the JSP file has taglib directives with URIs for the parts of the library that you are using, the Source Editor provides code completion for this library.

---

### 12.4.1 How to use Tag Libraries

You use the IDE to add the tag library to the web application's classpath. You use taglib directives in a JSP file to declare the tag libraries that the JSP file uses and to give each library a namespace unique within the JSP file. The tags are used in the same manner as regular XML tags with namespaces. The Source Editor provides code completion and Javadoc for tags and tag attributes of all types of tag libraries mentioned previously.

### 12.4.2 How to Create a Tag Library Descriptor

A tag library consists of:

- A set of tag handlers that implement the tag library's feature set.
- A Tag Library Descriptor (TLD) that describes the tags in the tag library and maps each tag to a tag handler.

You can provide custom tags for JSP files using either tag handlers or JSP tag files. A tag library can contain either type of custom tag, or it can contain both types. When

you bundle a tag library in a JAR file, you must include a TLD (tag library descriptor) file. A JSP container uses the TLD file to associate a URI with a tag library and its tags.

The TLD file contains:

- Documentation on the library as a whole and on its individual tags
- Version information on the JSP container and on the tag library
- Information about each of the actions (tags) defined in the tag library

You use the `tag` element to add a Java class tag handler to a TLD. You use the `tag-file` element to add a tag handler implemented in a tag file to the TLD.

The IDE creates a TLD file when you use the New File wizard to create a tag library. You can then use code completion in the Source Editor to edit the elements of the TLD file. For example, you will need to do this for certain advanced features, including adding validators and event listeners. If you edit the TLD file directly, you can validate it as you would with a normal XML file.

The following steps show how to create a TLD in a web application. After you create the TLD, use code completion in the Source Editor to define the tag library descriptor's properties.

**To create a TLD in a web application:**

1. In the **Projects** window or **Files** window, right-click the project node.
2. From the context menu, choose **New > Other**.
3. Under **Categories**, select Web. Under **File Types**, select Tag Library Descriptor. Click **Next**.
4. Type the name of the TLD file. Do not add the `.tld` extension, unless it is part of the name.
5. Type the folder where your TLD files are housed. By default, it is created in `WEB-INF/tlds`. If you create TLD files outside the `WEB-INF` subfolder, but still within the web folder, you must map them in your `WEB-INF/web.xml` file. You can use the Source Editor to modify the URI.
6. Type the prefix to be used within your JSP file. By default, the prefix is the same as the name of your TLD file. You can use the Source Editor to modify the prefix.
7. Click **Finish**.

The IDE creates a TLD file that conforms to the JSP 2.0 specification in J2EE 1.4 projects or to the JSP 1.2 specification in J2EE 1.3 projects. The IDE supports both types of TLD files. That is, you can edit them using code completion in the Source Editor.

For more information about creating and using tag libraries, see the *JavaServer Pages Specification* available at  
<http://download.oracle.com/otndocs/jcp/jsp-2.1-fr-eval-spec-oth-JSpec/>.

### 12.4.3 How to Edit a Tag Library Descriptor

**You can use the Source Editor to edit a tag library descriptor (TLD) file.**

1. In the **Projects** window or **Files** window, double-click the TLD file's node.
2. The TLD file opens in the Source Editor.
3. In the Source Editor, edit the tag library descriptor by using the following main elements:

- <short-name>. Specifies a simple name that can be used by JSP page authoring tools to suggest a namespace prefix for using the tag library in a JSP page. The default is the .tld filename, without the .tld extension.
- <display-name>. Specifies a short name intended for display by tools. It does not need to be unique.
- <tlib-version>. Specifies the version of the tag library being created. The default is 1.0.
- <uri>. Specifies a public URI that uniquely identifies this version of the tag library. JSP files that use this tag library must use this value for the uri attribute in the taglib directive, unless there exists a jsp-config element in the web.xml file that maps a different URI to the tag library.
- <icon>. Specifies the name of a file containing a small (16 x 16) icon image for use by tools. The file name is a relative path within the tag library.
- <description>. Specifies descriptive information about the tag library.
- <version>. This is an attribute of the <taglib> element. It specifies the JSP version that the tag library requires. The default for J2SEE 1.3 is 1.2 and for J2SEE 1.4 is 2.0.

For general information about tag libraries, see *JavaServer Pages Specification* available at <http://download.oracle.com/otndocs/jcp/jsp-2.1-fr-eval-spec-oth-JSpec/>.

#### 12.4.4 How to Create a Tag File

You can use JSP syntax in tag files to create custom tags. Tag files can have the following extensions:

- .tag - a tag file that is written using the standard JSP syntax
- .tagx - a tag file that is written using JSP document syntax
- .tagf - a tag segment

All tag files that are under WEB-INF/tags are available for use by the JSP files in that web application. Each folder under WEB-INF/tags represents a separate tag library. You use the tagdir attribute in a taglib directive to specify the location of the tag library folder. For example, <%@ taglib prefix="x" tagdir="/WEB-INF/tags/xtremeLib" %>.

Tag files can also be referenced in a TLD file and bundled in a JAR file.

The IDE recognizes tag files and provides wizard support for creating new tag files.

---

**Note:** A tag file is run when the JSP file in which it is defined is deployed. You can set breakpoints in the tag file. Then, when the JSP file is debugged, the tag file is debugged too.

---

For a full guide to creating and using tag files, see <http://wiki.netbeans.org/FaqTaglibrary>.

The following steps show how to create a tag file in a web application. After you create the tag file, use code completion in the Source Editor to specify the tag's attributes, body, and behavior.

##### To create a tag file:

1. In the **Projects** window or **Files** window, right-click the project node.

2. From the context menu, choose **New > Other**.
3. Under **Categories**, select Web. Under **File Types**, select Tag File. Click **Next**.
4. Type the name of the tag file.
5. Type the folder where your tag files are housed. By default, they are created in `WEB-INF/tags`.
6. Select the type of tag file that you want to create:
  - **Tag File (Standard Syntax)**. Tag files written using the standard JSP syntax typically have the `.tag` extension.
  - **Tag File (XML Syntax)**. Tag files written using the XML document syntax typically have the `.tagx` extension. Two examples of the advantages that XML document syntax has over standard JSP syntax are:
    - You can edit, verify, and view files in XML document syntax with XML capable tools, such as the IDE's Source Editor.
    - You can transform files in XML document syntax using XSLT tools, such as the IDE's XSL Transformation command.
7. (Optional) Click the **Create as a Segment of a Tag File** checkbox. A segment is a file that contains a fragment of tag file text for inclusion by tag files. Tag segments typically use the `.tagf` extension.
8. (Optional) Click the **Add Tag File to Tag Library Descriptor** checkbox if you want the IDE to create entries in a TLD file for the tag file. If you click this checkbox, you must have a TLD file available. If not, create one first.

If you selected the **Add Tag File to Tag Library Descriptor** checkbox, do the following:

  1. Click **Browse** to select the tag library descriptor (TLD file) for your tag file. By default, the TLD files are in the `WEB-INF/tlds` folder.
  2. Type the name of the tag. This is the name you use in the JSP file to refer to the tag file.
9. Click **Finish**.

The IDE creates a tag file that conforms to the JSP 2.0 specification.

---

**Note:** A tag file is run when the JSP file in which it is defined is deployed. You can set breakpoints in the tag file. Then, when the JSP file is debugged, the tag file is debugged too.

---

For more information about creating and using tag libraries, see the *JavaServer Pages Specification* available at  
<http://download.oracle.com/otndocs/jcp/jsp-2.1-fr-eval-spec-oth-JSpec/>.

## 12.4.5 How to Edit a Tag File

### To edit a tag file:

1. In the **Projects** or **Files** window, locate your tag file.
2. Double-click the selected file to open it in the Source Editor.

3. Edit your tag file as you would any Java file, including the use of Source Editor features for tag files.
4. Choose **File > Save** to save your file. Unsaved changes are indicated by an asterisk in the file's tab in the Source Editor.

### 12.4.6 How to Create a Tag Handler

You can develop tag handlers for JSP custom actions using Java classes or tag files. This topic discusses how to use the IDE to create tag handlers using Java code. See [Section 12.4.4, "How to Create a Tag File"](#) for information about tag files.

The IDE provides tools for creating a TLD file and for adding tag elements, tag attribute elements, and variable elements to the TLD file. You can create Java code for a tag handler by using the New File wizard and then you can edit the tag handler code in the Source Editor to modify the logic that implements the features of the tag.

---

**Note:** Before you can create a tag handler, you have to create its tag library descriptor (a TLD file). See [Section 12.4.2, "How to Create a Tag Library Descriptor"](#) for information about TLD files.

---

**To create a tag handler:**

1. In the **Projects** window or **Files** window, right-click the project node.
2. From the context menu, choose **New > Other**.
3. Under **Categories**, select Web. Under **File Types**, select Tag Handler. Click **Next**.
4. Type the class name of the tag handler.
5. Type the package name of your tag handler.
6. Select **BodyTagSupport** if you want to write complex tags, such as nested tags or iteration tags. Otherwise, leave the default **SimpleTagSupport** selected.
7. Click **Next**.
8. Define the tag handler's TLD information, see [Section 12.4.7, "How to Define TLD Information for a Tag Handler"](#) for details.

For a full guide to creating and using tag handlers, see <http://wiki.netbeans.org/FaqTaglibrary>.

### 12.4.7 How to Define TLD Information for a Tag Handler

**To define TLD information for a tag handler:**

1. Click the **Add Corresponding Tag to the Tag Library Descriptor** checkbox if you want the IDE to create entries in a TLD file for the tag handler. If you click this checkbox, you must have a TLD file available. If not, create one first.
2. If you clicked the **Add Corresponding Tag to the Tag Library Descriptor** checkbox, do the following:
  1. Click **Browse** to select the tag library descriptor (TLD file) for your tag handler. By default, the TLD files are in the **WEB-INF/tlds** folder.
  2. Type the name of the tag. This is the name you use in the JSP file to refer to the tag handler.
  3. Specify the allowable content of the tag handler:

**empty.** Tags that do not accept a body.

**scriptless (default).** Body content containing custom and standard tags and HTML text.

**tagdependent.** All other types of body content, such as SQL statements passed to the query tag.

3. Click **New** to create attributes for the tag handler. Use the **Add New Attribute** dialog to create an attribute as follows:
  - **Attribute Name.** Specifies the name of the attribute.
  - **Attribute Type.** Specifies the type of the class being referred to. Choose a type from the drop-down menu or type one yourself. The default is `java.lang.String`.
  - **required attribute.** If selected, specifies that the attribute must be given an argument whenever the tag is called. This option is set to False by default.
  - **value evaluated at request time.** If selected, specifies that the value of the attribute can be dynamically calculated at request time. This option is set to True by default. Mutually exclusive with the Value evaluated at JSP translation time attribute.
  - **value evaluated at JSP translation time.** If selected, specifies that the value of the attribute is static and determined at translation time. This option is set to False by default. Mutually exclusive with the Value evaluated request time attribute.

4. Click **Finish**.

## 12.5 Working with Applets

Before you create an applet in the IDE, you must determine the type of project from which you want to run the applet. You can create the applet in one of the following two places:

- Within a Java Application project with Java WebStart enabled and with an applet descriptor specified in the application's JNLP file.
- Within a Java Class Library project that is a library for a Web Application project.

### 12.5.1 How to Create an Applet

#### To create an applet in a web application:

1. Choose **File > New Project**. Under **Categories**, select Java. Under **Projects**, select Java Class Library. Complete the wizard.
2. Right-click the node of the new project in the **Projects** window and select **New > Other**. Under **Categories**, select Swing GUI Forms. Under **File Types**, select JApplet Form. Click **Next**.
3. In the **Class Name** text field, type the name of your applet. In the **Package** text field, enter the package to which the applet will belong.
4. Click **Finish**.

The IDE creates the applet in the specified package. The applet opens in the Source editor.

5. Design the applet with the GUI Builder or type code in the source view.
6. Right click the project node in the **Projects** window and choose **Build** from the context menu.

7. Run the applet, see [Section 12.5.3, "How to Run an Applet"](#).

---

**Note:** When you create the applet, you can also choose from the JApplet or Applet templates in the Java category. These templates do not work with the IDE's GUI Builder.

---

## 12.5.2 How to Create an Applet that is Called from a JNLP File

**To create and run an applet that is called from a JNLP file:**

1. Choose **File > New Project**. Under **Categories**, select Java. Under **Projects**, select Java Application. Complete the wizard.
  2. Right-click the node of the new project in the Projects window and select **New > Other**. Under **Categories**, select Swing GUI Forms. Under **File Types**, select JApplet Form. Click **Next**.
  3. In the **Class Name** text field, type the name of your applet. In the **Package** text field, enter the package to which the applet will belong.
  4. Click **Finish**.
- The IDE creates the applet in the specified package. The applet opens in the Source editor.
5. Design the applet with the GUI Builder or type code in the source view.
  6. Right-click the project's node, choose **Properties**, and select the Web Start panel.
  7. Select the **Enable Web Start** checkbox.
  8. Select the **Applet Descriptor** radio button and make sure that your applet is selected in the **Applet Class** combo box.
  9. Close the **Project Properties** dialog box.
  10. Right-click the project node in the **Projects** window and choose **Build** from the context menu.
  11. Right-click the project node and choose **Run Project**.

## 12.5.3 How to Run an Applet

If you create an applet as part of a Java WebStart-enabled project (see [Section 12.5.2, "How to Create an Applet that is Called from a JNLP File"](#)), you can run the applet by running the project.

If your applet is not part of a WebStart-enabled project, the applet is not run as part of the project.

**To run an applet individually:**

1. Right-click an applet class in the **Projects** window or **Files** window.
2. Select **Run File** from the context menu.

The *myappletclass.html* launcher file, with the applet embedded, is created in the build folder and launched in the Applet Viewer.

To run or debug an applet with the parameters, you need to edit the launcher, copy it from the build folder to the package where the applet class lives in the **src** folder. Make sure that the *myappletclass.html* launcher file has the same name as the applet class. Now edit the *myappletclass.html* launcher file as needed. When you

build the project, the *myappletclass.html* launcher file is copied from the `src` folder to the `build` folder.

---

**Note:** The *myappletclass.html* launcher file in your `build` folder is overwritten each time you run or debug the applet. Therefore, do not modify the *myappletclass.html* launcher file in your `build` folder.

---

3. To exclude the *myappletclass.html* launcher file from the JAR file, right-click the project, choosing Properties, clicking Packaging, and adding an expression to exclude them.

---

**Note:** An HTML file is created by the IDE when you run or debug an applet. When you copy it to your `src` folder for editing, it will automatically be included in the JAR file when you build the project.

---

4. Package the applet in the web application.
5. Define the applet in a JSP file, see [Section 12.3.5, "How to Access an Applet from a JSP Page."](#)
6. Run the JSP file that contains the applet (see [Section 12.3.9, "How to Run a JSP File"](#)) or deploy the web application that contains the JSP file (see [Section 12.11.1, "How to Deploy a Web Application"](#)).

#### Debugging an Applet

Applets run in the virtual machine of the IDE's default web browser. The IDE uses a different virtual machine and therefore applets are not included in a web application's debug session. Therefore, debug the applet launcher file as described in step 2 above.

### 12.5.4 How to Generate an Applet Policy File

By default, applets do not have access to resources on a client's computer, such as threads and disk operations. An applet's permissions are defined in its `applet.policy` file. If you have an applet policy file, you can specify its location in the IDE. Then, when you run the application, the IDE uses the applet policy file that you specified. If you do not specify an applet policy file, the IDE generates one for you when you run the applet. The applet policy file that the IDE generates for you grants all permissions to the applet. You can use the Source Editor to modify the policy file, so that appropriate security checks are done.

#### To specify the location of an existing applet policy file:

1. Right-click the project node and choose **Properties**.
2. In the left pane of the **Project Properties** dialog box, select **Run**.
3. In **VM Options**, specify the location of the applet policy file. For example, use the following setting to specify that the applet policy file is in the project's root folder:  
`-Djava.security.policy=applet.policy`
4. Click **OK**.

#### To use the IDE to generate an applet policy file:

1. Run the applet, see [Section 12.5.3, "How to Run an Applet."](#)

The IDE creates an applet policy file.

2. In the **Files** window, expand the project node and double-click the `applet.policy` file.
3. In the Source Editor, set permissions according to your needs.
4. Right-click the project node and choose **Properties**.
5. In the **Project Properties** dialog box, select **Run**.
6. In **VM Options**, note that the IDE has specified the location of the applet policy file. By default, the applet policy file is in the project's root folder:  
`-Djava.security.policy=applet.policy`
7. Click **OK**.

## 12.6 Working with Servlets

Servlets are server-side programs that give Java technology-enabled servers additional features. Servlets provide web developers with a simple, consistent mechanism for extending the features of a web server and for gaining access to existing business systems. In other words, servlets are programs written in the Java programming language that execute on the server. You can contrast servlets to Java programs written for the client browser, that is, applets.

Servlets are Java classes that can be loaded dynamically into a web server and executed by a web server to extend its feature set. Servlets are useful for tasks that involve processing an HTTP request. Servlets are also helpful for tasks that do not generate an HTTP response at all or that generate a relatively simple HTTP response. Servlets use the `javax.servlet` API.

JSP technology was designed to simplify the process of creating servlets. In fact, the main function of servlets is to generate HTML output in cases where dynamic portions can be encapsulated. Servlets are generated by JSP pages when compiled. In many applications, the response sent to the client is a combination of template data and dynamically-generated data. In this situation, it is often easier to work with JSP pages than to do everything with servlets.

For more information about servlet technology, see the Java Servlet Technology Overview available at  
<http://www.oracle.com/technetwork/java/javaee/servlet/index.html>.

### 12.6.1 How to Create a Servlet Source File

When you create a servlet in the web application's `src` folder and compile it, the IDE includes it in the web application's WAR file. When you create the servlet using the following steps, the IDE compiles the class and its package structure to the `build/web/WEB-INF/classes` folder.

#### To create a servlet source file:

1. In the **Projects** or **Files** window, right-click the project's node and choose **New > Other** from the pop-up menu. The New File wizard appears.
2. Under **Categories**, choose Web. Under **File Types**, choose Servlet. Click **Next**.
3. Specify the fully qualified class name, the location and the servlet's package. Do not include a filename extension to the class name as this is added automatically when the file is created.
4. Click **Next** to specify the server deployment configuration for the servlet.

This page does not enable the **Finish** button until you enter a servlet name and URL mappings that are unique to the deployment descriptor. After you create the servlet, you can edit this information in the deployment descriptor (`web.xml`).

5. Click **Finish**. The IDE opens the file for editing in the Source Editor.

## 12.6.2 How to Edit a Servlet Source File

The IDE provides support for editing Java source files, such as syntax highlighting, Java SE API code completion, automatic formatting, bracket matching, and macros, to name a few of the many features. In addition to the Java support, the IDE provides code completion for the Servlet API. Note that although you can view the servlet that generated from a JSP file by right-clicking the JSP file and choosing **View Servlet** from the pop-up menu, you cannot edit this servlet.

### To edit a servlet source file:

1. Locate your servlet file in the **Projects** window or **Files** window.
2. Double-click the selected file to open it in the Source Editor.
3. Edit your file as you would any Java source file, including the use of code completion and editor abbreviations.
4. Choose **File > Save** to save your file. Unsaved changes are indicated by an asterisk in the file's tab in the Source Editor.

---

**Note:** If you create the servlet outside of the IDE or turn an existing class into a servlet, the IDE may not recognize the file as a servlet. If this is the case, manually register the servlet in the `web.xml` file.

---

## 12.6.3 How to View the Servlet Generated from a JSP File

When you execute a JSP file, the server translates the contents of the JSP file into a Java servlet and compiles the servlet. You can view, but not edit, the translated servlet source in the Source Editor. You cannot set breakpoints in the servlet source code. You can set breakpoints in the JSP file only.

### To view the servlet generated from a JSP file:

1. Run the JSP file, see [Section 12.3.9, "How to Run a JSP File."](#)
2. In the **Projects** window, right-click the JSP file and choose **View Servlet**.

---

**Note:** The **View Servlet** menu item is enabled only if the target server has successfully executed the JSP file and the target server supports the viewing of translated servlets.

---

The servlet appears in the Source Editor. This servlet is based on the state of the JSP file when the file was last executed. If you have changed the JSP file but have not executed it, the servlet does not reflect the changes. While this file is opened, if you execute the JSP file or if you change the target server in the Run section of the Project Properties dialog box, the file changes to reflect the latest translated servlet on the selected server.

## 12.6.4 How to Specify Parameters for a JSP Page

You can pass request parameters in URL query string format to JSP pages and servlets from the IDE. Specifying input data in this fashion provides a useful aid in testing for expected JSP and servlet output.

### To specify parameters for a JSP page:

1. In the **Projects** window or **Files** window, right-click the JSP file and choose **Properties**.
2. In the **Request Parameters** dialog, type the parameters in the URL query string format. See [Section 12.3.8, "How to Pass Request Parameters"](#) for details on URL query string.
3. Click **OK**.
4. Click **Close** to exit the **Properties** window.

The IDE saves the parameters and automatically passes them to the JSP or servlet the next time you select the file and choose **Run File** from the context menu.

---

**Note:** The HTTP Monitor enables you to monitor data flow on the server.

---

## 12.6.5 How to Run a Servlet

When you run a servlet, the IDE sends a URL to the server. The URL is comprised of the server's URL, the Execution URI value, and Request Parameters. The default URI value is specified in the deployment descriptor (the `web.xml` file).

---

**Note:** Your servlet must have an entry defined in the deployment descriptor (the `web.xml` file) in order to be able to run. The entry can be created automatically by the IDE when you create a servlet in the New File wizard. Otherwise, you have to create it by hand in the `web.xml` file.

---

### To run a servlet:

1. (Optional) Define request parameters to pass to the servlet, see [Section 12.6.4, "How to Specify Parameters for a JSP Page."](#)
2. Select the servlet in the Source Editor or Projects window.
3. Choose **Run > Run File > Run "filename.java"** (Shift-F6) from the main menu. If the compiled servlet is newer, the server reloads the servlet.

---

**Note:** If you are using a free-form project, this command is disabled by default. You have to write an Ant target for running the currently selected file in the IDE and hook it up to the Run Class command. See [Section 6.2.4.6, "Mapping an Ant Target to an IDE Command."](#)

---

### Troubleshooting

If you get "file not found" errors when you execute a servlet, see [Section 23.2.2, "How to Change the Default Web Browser"](#) for a possible solution.

## 12.7 Using Filters

A filter is a piece of re-usable code that modifies requests to and responses from a servlet. Filters can perform many functions, including (but not limited to):

- Authentication, that is, blocking requests based on user identity
- Logging and auditing, that is, tracking users of a web application
- Localization, that is, tailoring the request and response to a particular locale
- Data compression, that is, making downloads smaller
- XSLT transformations of XML content, that is, targeting web application responses to more than one type of client

The IDE provides a wizard to help you create both simple and more advanced filter elements.

---

**Note:** Filters are run on each mapped request; therefore, you cannot run or debug them individually.

---

Filters are declared using the filter element in the web application's deployment descriptor (`web.xml`) and are packaged in a WAR file along with the static content and servlets that make up a web application. A filter or collection of filters can be configured for invocation by defining the filter mapping elements in the deployment descriptor.

For more information about filters, see the Java Servlet Technology Overview page available at  
<http://www.oracle.com/technetwork/java/javaee/servlet/index.html>.

### 12.7.1 How to Create a Filter

You create a filter by implementing the `javax.servlet.Filter` interface and providing a public constructor taking no arguments. The IDE provides a wizard to help you create two types of filters:

- A basic filter class
- A filter class that wraps the Request and Response objects prior to passing on the request.

#### To create a Filter:

1. Right-click the package where you want to put the filter and choose **New > Other** from the pop-up menu.
2. Under **Categories**, select Web. Under **File Types**, select Filter. Click **Next**.
3. Specify the fully qualified class name, the location and the servlet's package. Do not include a filename extension to the class name as this is added automatically when the file is created.
4. Select the **Wrap Request and Response Objects** checkbox if you want the filter to wrap the request and response objects. For example, you can extend the capabilities of the request and response by allowing parameters to be set on the request before it is sent to the rest of the filter chain. Or you can keep track of the cookies that are set on the response. Note that some servers do not handle wrappers very well for forward or include requests.

5. Click **Finish** to register the filter in the deployment descriptor. You can also register the filter afterwards, using the Servlet Filters section of the `web.xml` Visual Editor.
6. Click **Next** to define initialization parameters or click **Finish** to accept the default configuration. The IDE opens the filter source code in the Source Editor.

For more information about filters, see the Java Servlet Technology Documentation available at the Java Servlet Technology Overview page at <http://www.oracle.com/technetwork/java/javaee/servlet/index.html>.

### 12.7.2 How to Register a Filter

You can add a filter to a web application deployment descriptor (`web.xml`) in two ways:

- Using the New Filter wizard when you create the filter. On the Configure Filter Deployment page, select the Add information to deployment descriptor (`web.xml`) checkbox, type the filter's internal name, and add the filter's mappings. Optionally, you can add the filter's initialization parameters from the Filter Init Parameters page, which is the next step in the wizard.
- Using the Servlet Filters section of the `web.xml` Visual Editor.

The `filter` element defines a filter in the deployment descriptor. Filter class instances are scoped to one filter per deployment descriptor declaration per Java virtual machine.

For more information about filters, see the *Java Servlet Specification* available at <http://www.oracle.com/technetwork/java/index-jsp-135475.html>.

## 12.8 Using Web Application Listeners

Listeners allow you more control over interactions with the `ServletContext` and `HttpSession` objects and let you efficiently manage the resources used by a web application.

Application event listeners are classes that implement one or more of the servlet event listener interfaces. Application events let listener objects be notified when servlet contexts and sessions are initialized and destroyed, as well as when attributes are added or removed from a context or session.

Use Servlet context listeners to manage resources or state held at a virtual machine level for the web application. The IDE helps you create two types of context event listeners:

- `ServletContextListener`, which notifies you that the servlet context has just been created and is available to service its first request, or that the servlet context is about to be shut down.
- `ServletContextAttributesListener`, which notifies you that attributes on the servlet context have been added, removed, or replaced.

Use HTTP session listeners to manage state or resources associated with a series of requests made to a web application from the same client or users. The IDE helps you create two types of session event listeners:

- `HttpSessionListener`, which notifies you that an `HttpSession` has been created, invalidated, or timed out.

- `HttpSessionAttributesListener`, which notifies you that attributes have been added, removed, or replaced on an `HttpSession`.

Use Servlet request listeners to observe as requests are created and destroyed, and as attributes are added and removed from a request. This is supported by J2EE 1.4 only. The IDE helps you create two types of request event listeners:

- `ServletRequestListener`, which notifies you that the request is coming in or out of a web application's scope.
- `ServletRequestAttributesListener`, which notifies you that request attributes have been added, removed, or replaced.

Listener classes are declared in the deployment descriptor (`web.xml`) using the `listener` element. They are listed by class name in the order in which they are to be invoked.

The IDE provides a wizard to help you create listener classes.

---

**Note:** Listeners are run on each mapped event, therefore you cannot run or debug them individually.

---

For more information about listeners, see the *Java Servlet Specification* available at <http://www.oracle.com/technetwork/java/index-jsp-135475.html>.

### 12.8.1 How to Create a Web Application Listener

The *Java Servlet Specification* supports application level events for controlling interactions with the `ServletContext` and `HttpSession` objects and for managing the resources that a web application uses.

Application event listeners are classes that implement one or more of the servlet event listener interfaces. Application events notify listener objects when servlet contexts and sessions are initialized and destroyed, as well as when attributes are added or removed from a context or session.

The IDE provides a wizard to help you create listener classes. Use the Web Application Listener wizard to create listener classes that manage resources or state held at a virtual machine level for the web application.

You use the `listener` element to declare listeners in the deployment descriptor (`web.xml`). List the `listener` elements in the order in which they are to be invoked.

**To create a listener:**

1. In the **Projects** or **Files** window, right-click the package node and choose **New > Other** from the pop-up menu.
2. Under **Categories**, select **Web**. Under **File Types**, select **Web Application Listener**.
3. Follow the instructions in the New Web Application Listener wizard and click **Finish**. The source code appears in the Source Editor.

For more information about listeners, see the *Java Servlet Specification* available at <http://www.oracle.com/technetwork/java/index-jsp-135475.html>.

### 12.8.2 How to Register a Web Application Listener

You can add a listener to a web application deployment descriptor (`web.xml`) in two ways:

- Using the New Web Application Listener wizard when you create the listener (see [Section 12.8.1, "How to Create a Web Application Listener"](#)). On the Name and Location page, select the Add information to deployment descriptor (web.xml) checkbox.
- Using the Web Application Listeners section of the web.xml Visual Editor.

The listener element defines a listener in the deployment descriptor. Listener class instances are scoped to one listener per deployment descriptor declaration per Java virtual machine.

For more information about listeners, see the *Java Servlet Specification* available at <http://www.oracle.com/technetwork/java/index-jsp-135475.html>.

## 12.9 Using WebSocket Endpoints

The Java API for WebSocket is included as part of the Java EE platform. You can use WebSocket endpoints in a web application to send and receive text and binary messages between two peers over the TCP protocol. To use WebSocket you create an endpoint in the application and deploy the application to the server. After a client establishes a connection to the endpoint URI (the 'handshake') the client can send messages to and receive messages from the endpoint. You can use the WebSocket API in a JavaScript file to initiate the session handshake with the server.

The Java API for WebSocket also provides encoder and decoder interfaces that you can implement to convert custom Java objects to WebSocket messages (for example, JSON) and from WebSocket messages to Java objects.

For more information about using WebSocket endpoints, see the *Java API for WebSocket* chapter in the *Java EE 7 Tutorial* available at <http://docs.oracle.com/javaee/7/tutorial/doc/websocket.htm>.

For an example of using a WebSocket endpoint in an application, see the following tutorial.

[Using the WebSocket API in a Web Application](#)

### 12.9.1 How to Create a WebSocket Endpoint

The IDE provides a wizard to help you create endpoint classes. The WebSocket Endpoint wizard creates an endpoint class that contains a default `onMessage` method that is decorated with the `@OnMessage` annotation. The class is annotated with `@ServerEndpoint` and the value of the endpoint URI. You specify the endpoint URI in the wizard when you create the class.

#### To create a WebSocket endpoint:

1. In the **Projects or Files** window, right-click the package node and choose **New > Other** from the pop-up menu.
2. Under **Categories**, select **Web**. Under **File Types**, select **WebSocket Endpoint**.
3. Specify the name of the class and the package.
4. Specify the endpoint URI. Click **Finish**.

When you click **Finish** the new class opens in the source editor.

## 12.9.2 How to Create a WebSocket Encoder or Decoder

The WebSocket API supports converting different message types with encoders and decoders. The encoder or decoder is a simple Java class that implements the appropriate interface.

### To create a WebSocket encoder or decoder:

1. In the **Projects** or **Files** window, right-click the package node and choose **New > Java Class** from the pop-up menu.
2. Specify the name of the class. Click **Finish**.  
When you click Finish the new class opens in the source editor.
3. Implement the WebSocket encoder or decoder interface.

## 12.10 Configuring a Web Application

You configure your web applications at several different levels.

### Project Contents and Classpath

Basic project settings like the web application's source roots and classpath are set in the application's Project Properties window. You open this window by right-clicking the web application's project node and choosing Properties.

For standard projects, you can add source roots in the Sources pane of the Project Properties dialog box. You can quickly add to the compilation and test classpath by right-clicking the Libraries or Test Libraries node in the Projects window. To further configure the classpath, or to specify which items should be included in deployment, use the Libraries page of the Project Properties dialog box.

For free-form projects, there are no Libraries or Test Libraries nodes. The classpath and source roots are managed completely by your project's Ant script. In the Java Sources and Java Sources Classpath pages of the Project Properties dialog box, you just configure the project to match the settings that already exist in your Ant script. This tells the IDE which classes to offer for debugging, code completion, and so forth.

### Build Settings

Compilation and packaging settings for a project are set in the Build category in the application's Project Properties window. You can use the following panes in the Build category to set build options for the project.

- **Compiling.** You can use the Compiling pane to set compilation options and specify commands that are run when the application is compiled.
- **Packaging.** You can use the Packaging pane to specify the archive that is generated when the application is compiled and any additional libraries, projects and files that you want to include in the archive. Use the Libraries pane to add items to the project classpath.
- **Documenting.** You can use the Documenting pane to specify the options for generating Javadoc documentation for the project.

### Deployment Settings

There are several key steps to configuring deployment settings:

- **Setting the target server.** You can have several instances of your favorite server registered. You can set the target server instance for a web application by doing any of the following:
  - Specifying the server in the New Project wizard when creating the project.
  - Going to a web application's Project Properties window and setting the target server in the Run pane.
  - Adding the web application to an enterprise application. The web application is then deployed to the same server instance as the enterprise application.
- **Adding the web application to a Java EE application.** You can add a web application to an enterprise project by doing any of the following:
  - Specifying the enterprise application in the New Project wizard when creating the project.
  - Right-clicking the J2EE Modules node for any enterprise application project and choosing Add J2EE Module.
- **Configuring deployment descriptors.** The IDE automatically updates your deployment descriptors as you code your web application. Whenever you insert a call to an enterprise bean, for example, the IDE automatically enters the necessary information in your deployment descriptors. You can manually configure your deployment descriptors by expanding the web application's Configuration Files node and double-clicking web.xml, sun-web.xml, or context.xml.

### 12.10.1 How to Set Build Properties

The build properties for the application are grouped under the Build category in the Project Properties window.

#### To set build properties:

1. Right-click the project node in the Projects window and choose Properties.
2. Select a subcategory under the Build category in the left pane of the Project Properties window.
3. Set the build properties.
4. Click OK.

### 12.10.2 How to Edit Deployment Descriptors

Deployment descriptors are XML-based text files whose elements describe how to assemble and deploy a module to a specific environment. The elements also contain behavioral information about components not included directly in code.

For web applications, there are three types of deployment descriptors:

- web.xml. The general web application deployment descriptor that configures deployment settings for components belonging to a web application, such as servlets and JSP files.
- sun-web.xml. The server-specific deployment descriptor that configures deployment settings for the Glassfish application server.
- context.xml. The server-specific deployment descriptor that configures deployment settings for the Tomcat Web Server.

**To edit web.xml:**

1. In the **Projects** window, expand the Configuration Files node for your web application project.
2. Double-click web.xml to open it in the web.xml Visual Editor.

The web.xml Visual Editor opens in a Source Editor tab. The editor contains an Overview section and a section for most of the elements in the web.xml file.

3. Edit the deployment descriptor as necessary.
4. (Optional) Click XML at the top of the web.xml Visual Editor to view and edit the deployment descriptor's XML code.
5. Choose **File > Save** to save your changes.

**To edit sun-web.xml:**

1. In the **Projects** window, expand the Configuration Files node for your web application project.
2. Double-click sun-web.xml to open it in the graphical editor. The graphical editor opens in a Source Editor tab. The editor contains a tree view of the module's contents on the left and property editors for each item on the right.
3. Edit the deployment descriptor as necessary.
4. Choose **File > Save** to save your changes.

---

**Note:** If you enter any XML syntax errors the IDE automatically alerts you. To fully ensure your changes have not cause any errors, you should verify the web application.

---

**To edit context.xml:**

1. In the **Projects** window, expand the Configuration Files node for your web application project.
2. Double-click context.xml to open it in the graphical editor. The graphical editor opens in a Source Editor tab. The editor contains a tree view of the module's contents on the left and property editors for each item on the right.
3. Edit the deployment descriptor as necessary.
4. Choose **File > Save** to save your changes.

---

**Note:** If you enter any XML syntax errors the IDE automatically alerts you. To fully ensure your changes have not cause any errors, you should verify the web application.

---

## 12.11 Deploying a Web Application

The IDE uses an Ant script to run your web applications. If you are using a standard project (for more information on standard projects, see [Section 6.2.1, "Standard Project Templates"](#)), the IDE generates the build script based on the options you enter in the project's Project Properties dialog box. You can set the project's classpath, context path, and web server in the Project Properties dialog box. You can further customize program execution by editing the Ant script and Ant properties for the project.

If you are using a free-form project (for more information on free-form projects, see [Section 6.2.2, "Free-Form Templates"](#)), the IDE uses your existing Ant script to run your project. You can write a target that executes the currently selected project in the IDE and map it to the Deploy command.

You can execute your web application from the IDE using one of the supported servers that implement a web container. The process for running a web application is slightly different depending on which server you use. For more information about specifying the server on which to run the web application, see [Section 12.11.2, "How to Change the Target Server"](#). Some servers do not support web applications in WAR format. In such cases, it might be necessary to extract the WAR file within the server before deploying. For details on whether the server you intend to use supports JSP files and WAR files, see the documentation accompanying your server.

To execute a web application, the web server requires a configuration file. When you create a web application from the IDE, the IDE creates the necessary server configuration for you.

All the resources that the web application uses must be in the server's classpath and their placement must follow the WAR structure:

- Public JSP files and static files, such as HTML files and images, must be in the document base (root folder) or one of its subfolders. JSP files that are under the WEB-INF folder can only be accessed by another JSP or by a servlet.
- Tag library descriptor (TLD) files must be in the WEB-INF folder or a subfolder, such as WEB-INF/tlds or WEB-INF/tags. If a tag library JAR file contains the TLD, you are not required to have a TLD under the WEB-INF folder. Note that tag library descriptors are not required for tag files.
- The classes must be in a Source Packages subfolder.
- JAR files must be in the WEB-INF/lib folder or in the web server's shared library folder. For information about using shared libraries when deploying to other servers, see the documentation accompanying that server. Note that Tomcat does not support .zip files in the WEB-INF/lib folder.
- Tag files must be the WEB-INF/tags folder or one of its subfolders.

### Troubleshooting

If you get "file not found" errors when you execute a web application, see [Section 23.2.2, "How to Change the Default Web Browser"](#) for possible solutions.

If the browser does not reflect changes in a JSP file or HTML file, this might be because of the way the browser is caching pages. Check the browser's settings to ensure that the browser is reloading the page every time you access it.

## 12.11.1 How to Deploy a Web Application

When you run a web application, the IDE automatically builds, deploys, and runs the web application using the project's ant build script and target server.

### To deploy a web application:

1. (Optional) Define parameters to pass to one or more JSP files, servlets, or both (see [Section 12.3.8, "How to Pass Request Parameters"](#)).
2. (Optional) Specify a different welcome file by right-clicking the project node in the **Projects** window, choosing **Properties**, clicking **Run**, and typing it in the **Relative URL** text box.

---

**Note:** The welcome file specified in the **Project Properties** window overrides the welcome file set in the web.xml file. If no welcome file is defined in the **Project Properties** window, the first existing welcome file defined in the web.xml file's welcome-file-list is displayed. If the server does not find a welcome file, the server's default servlet displays the root of the web application. You can change the behavior of the default servlet in the web.xml file.

---

3. (Optional) Specify a different target server or browser.
4. In the **Projects** window, right-click the project's node and choose **Run**. The project is compiled, and new and changed files are copied from the project's build directory to the WAR's deployment directory. The deployed application then opens in a browser.

---

**Note:** The **Compile on Save** feature is enabled by default for web projects. The IDE recompiles and deploys the project application when you save changes to project files in the editor. The **Compile on Save** feature can be toggled from the **Compiling** category in the Project Properties window.

---

The IDE sends the web application's URL to the server. The URL is derived from the server's URL, the web application's context path, and the relative URL or welcome file.

The IDE has a **Deploy on Save** feature for Java web and enterprise applications. When the **Deploy on Save** feature is enabled for a project and a project has been deployed to a server through the IDE, changed files are redeployed to the server immediately. For Deploy on Save to work on GlassFish V4, the Glassfish instance must have the **Directory Deployment Enabled** option selected.

**To enable or disable Deploy on Save for a Java web or enterprise project:**

1. Right-click the project's node and choose **Properties**.
2. Select the **Run** node and set the **Deploy on Save** property.

**To enable directory deployment Glassfish V4:**

1. Choose **Tools > Servers**.
2. Select the server.
3. Select the **Options** tab.
4. Select the **Directory Deployment Enabled** option.

### Troubleshooting

The Output window displays error information and the HTTP monitor displays requests, data states, and the servlet environment.

If you get HTTP 404 or Not Found error messages when you execute a web application, see [Section 23.2.2, "How to Change the Default Web Browser"](#) for possible solutions. Also verify the following:

- The URL is correct.
- The servlet mapping in the web.xml file is correct.

- The application's resources are in the appropriate location.

### 12.11.2 How to Change the Target Server

Each project has a target server. The target server is the server that is used when the project is run. You can set the target server to any server which has been registered in the IDE.

**To change the target server:**

1. Right-click the project node in the **Projects** window and choose **Properties**.
2. Select **Run** in the **Project Properties** dialog.
3. Select the new target server from the **Server** drop-down menu and click **OK**.

## 12.12 Debugging a Web Application

At the end of the development cycle, you need to debug and test your web application. The IDE has a number of features that help you to do so. The following are the most important ones:

- Detecting syntax problems in JSP files during compilation, see [Section 12.3.6, "How to Compile a JSP File."](#)
- Viewing the translated servlet after deployment, see [Section 12.3.7, "How to View a JSP File's Servlet."](#)
- Unit testing, see [Section 9.2, "Testing Java Application Projects with Unit Tests."](#)
- Using the debugger to debug web applications (see [Section 12.12.1, "How to Debug a Web Application"](#)), JSP files (see [Section 12.12.2, "How to Debug a JSP File"](#)), and servlets (see [Section 12.12.3, "How to Debug a Servlet"](#)).
- Using the profiler to profile web application performance (see [Section 12.13.1, "How to Profile a Standalone Web Application"](#)) and check for memory leaks.
- Monitoring by means of the HTTP Monitor. When you run web applications, JSP files, or servlets, the HTTP Monitor can gather data about HTTP requests that the servlet engine processes. For each HTTP request that the engine processes, the monitor records data about the incoming request, the data states maintained on the server, and the servlet context. You can view data, store data for future sessions, and replay and edit previous requests.
- Writing a target to debug and test free-form web projects.

### 12.12.1 How to Debug a Web Application

You can use the debugger in the IDE to debug web applications.

**To debug a web application:**

1. Set breakpoints and watches in the web application's JSP files, servlets, and other source files.
2. If necessary, specify request parameters in the JSP files and servlets, see [Section 12.3.8, "How to Pass Request Parameters."](#)
3. Right-click the project node in the **Projects** window and choose **Debug** from the context menu.

## 12.12.2 How to Debug a JSP File

You can use the debugger in the IDE to debug JSP files. Just as with Java programs, the IDE enables you to set new watches, evaluate a variable by holding the cursor over the variable, and set breakpoints in JSP files, JSP documents, and JSP segments. A tag file is debugged when you debug the JSP file that references it.

### To debug a JSP file:

1. Set breakpoints and watches in the JSP file and, optionally, its tag files.
2. If necessary, specify request parameters in the JSP file, see [Section 12.3.8, "How to Pass Request Parameters."](#)
3. Right-click the JSP file's node and choose **Debug File**. Optionally, choose **Debug > Debug file.jsp** from the main menu.

## 12.12.3 How to Debug a Servlet

You can use the debugger in the IDE to debug servlets.

### To debug a servlet:

1. Set breakpoints and watches in the servlet.
2. If necessary, specify request parameters in the servlet, see [Section 12.3.8, "How to Pass Request Parameters"](#).
3. Right-click the servlet's node and choose **Debug File**. Optionally, choose **Debug > Debug file.java** from the main menu.

## 12.13 Profiling a Web Application

There are two ways to profile a web application project:

- As a stand-alone web application project
- As part of an enterprise application

If your web application is part of an enterprise application, you should always profile it by running the profile command on the enterprise application project. Since the IDE does not know which enterprise applications a web application project belongs to, running the Profile command on a web application project deploys it as a stand-alone application.

If you are deploying your web application or enterprise application to a local installation of the Glassfish application server, you can profile your project by right-clicking the project node in the **Projects** window and choosing **Profile** from the context menu or by setting the project as the main project (see [Section 6.4, "Setting the Main Project"](#)) and choosing **Profile Main Project** from the main menu.

If you are deploying your web application or enterprise application to a remote server, you need to attach the IDE to the remote server and profile the application in Attach mode. The Attach Wizard can help you configure the remote server to accept attachment.

## 12.13.1 How to Profile a Standalone Web Application

### To profile a stand-alone web application:

1. In the **Projects** window, right-click the web application project and choose **Profile**.

- 
2. Select a profiling task and click **Run**.

When you profile a stand-alone web application project, the IDE does the following:

- Compiles the web application if necessary.
- Stops the application server and starts it in profile mode.
- Deploys the web application to the application server.
- Starts a profiling session, attaches the profiler to the server, and opens the Profiler window in the IDE. After the application is deployed to the server, you should invoke its code the same way as you would for debugging or just evaluating application functionality. You can view the profiling data once the code of the application is invoked.

### 12.13.2 How to Profile an Enterprise Application

#### To profile an enterprise application:

1. In the **Projects** window, right-click the enterprise application project and choose **Profile**.
2. Select a profiling task and click **Run**.

---

**Note:** If you are analyzing application performance, you may want to choose the Profile Projects & Subprojects Classes filter if you want to profile the classes in your web application and also any sub-projects, such as any EJB modules. If you do not choose this filter, the classes in the EJB module are not profiled.

You can limit the classes in the enterprise application that are profiled by doing the following:

1. In the **Analyze Performance** pane, choose the Profile Projects & Subprojects Classes filter and then click **Show Filter Value**.
  2. Click to Quickfilter in the dialog box.
  3. In the **Set Quick Filter** dialog, select a filter type and edit the filter values to limit the classes that are profiled and click **OK**.
  4. Make sure that Quick Filter is selected in the **Filter** drop-down list when you run the profiling session.
- 

When you profile an enterprise application, the IDE does the following:

- Compiles the EAR file if necessary.
- Stops the application server and starts it in profile mode.
- Deploys the enterprise application to the application server.
- Starts a profiling session, attaches the profiler to the server, and opens the Profiler window in the IDE. If the application has a designated web application and URL to run, the IDE opens the URL in the external browser.



# 13

---

## Using Web Application Frameworks

This chapter lists various Web application frameworks supported by NetBeans IDE and describes how to use the IDE for Web application development.

This chapter contains the following sections:

- [About Using Web Application Frameworks](#)
- [Working with the JavaServer Faces Framework](#)
- [Working with the Spring Framework](#)
- [Working with the Struts Framework](#)
- [Working with the Hibernate Framework](#)
- [Working with the Grails Framework](#)

### 13.1 About Using Web Application Frameworks

The IDE provides built-in support for various web frameworks, including:

- JavaServer Faces
- Spring Web MVC
- Grails
- Struts
- Hibernate

Framework support in the IDE is generally specific to the framework you are working with. However, support typically consists of the following:

- Project creation with framework template files and libraries added to the project classpath.
- Editor support, including code completion and documentation pop-ups for framework-specific tags.
- Wizards and dialogs for commonly-used framework-specific artifacts.

### 13.2 Working with the JavaServer Faces Framework

JavaServer Faces (JSF) technology is a server-side user interface component framework for building Java technology-based web applications. The main components of JavaServer Faces technology are as follows:

- An API for representing UI components and managing their state; handling events, server-side validation, and data conversion; defining page navigation; supporting internationalization and accessibility; and providing extensibility for all these features.
- Various tag libraries for expressing UI components within web pages and for wiring components to server-side objects.

JavaServer Faces technology provides a well-defined programming model and various tag libraries. These features significantly ease the burden of building and maintaining web applications with server-side UIs. With minimal effort, you can:

- Construct a UI with reusable and extensible components
- Drop components onto a page by adding component tags
- Wire component-generated events to server-side application code
- Bind UI components on a page to server-side data
- Save and restore UI state beyond the life of server requests

For a complete description of the JSF framework, including documentation and tutorials, see:

<http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html>.

The IDE provides functional support for the JavaServer Faces framework by including the following features:

- Project creation support: The JavaServer Faces structural template is included in the Frameworks panel of the New Web Application wizard, enabling you to create new projects with built-in JSF support.
- Editor support: Editor support is provided for Facelets pages and configuration files.
- Palette support: JSF tags are added to the Palette as drag-and-drop components.
- Wizard support: The IDE provides various wizards for generating code templates. You can use wizards to create the following files.
  - JSF Pages. You can use the JSF Pages wizard to create Facelets and JSP pages
  - JSF Managed Beans. You can use the JSF Managed Bean wizard to create a managed bean and register it with the application
  - JSF Pages from Entity Classes. You can use the JSF Pages from Entity Classes wizard to generate a collection of files, classes and front-end pages for CRUD functionality with a back-end data store.
  - Facelets Templates and Facelets Template Clients. You can use the Facelets Template wizard to create a template that determines the layout of client pages. The Facelets Template Client wizard enables you to create client pages that reference a Facelets template.
  - Composite Components. You can use the Composite Component wizard to create composite user interface (UI) components, which can be reused in web pages.

---

**Note:** Because JSF 2.0 (JSR-314) requires Java EE 6, support for JSF 2.0 is available based on your project's Java EE version. IDE continues to provide support for JSF 1.2 (JSR 127). This support is automatically available to you if your project does not rely on Java EE 6.

---

For more information on the JSF Framework, see the following resources:

- JSF 2.x Support in NetBeans IDE at  
<https://netbeans.org/kb/docs/web/jsf20-support.html>
- Generating a JavaServer Faces 2.x CRUD Application from a Database at  
<https://netbeans.org/kb/docs/web/jsf20-crud.html>
- Introduction to JavaServer Faces2.x at  
<https://netbeans.org/kb/docs/web/jsf20-intro.html>
- Official JavaServer Faces Technology documentation at  
<http://www.oracle.com/technetwork/java/javaee/documentation/index-137726.html>
- The Java EE 6 Tutorial, Chapter 4: JavaServer Faces Technology at  
<http://docs.oracle.com/javaee/6/tutorial/doc/bnaph.html>

### 13.2.1 How to Create a New Project with JSF Support

You can create a new JavaServer Faces application using the New Web Application wizard. JSF support is determined based on the Java EE version you specify in the wizard.

Java EE Version	JSF Version
7	2.2
6	2.0
5	1.2

When you create a new web application with JSF support, an empty project template is generated that contains various JSF-specific artifacts. JSF 2.x support is defined by the following:

- Facelets files (.xhtml) are the default page language
- JSF 2.x libraries are added to the project's classpath
- The Faces servlet and servlet mapping are added to the project's deployment descriptor

For JSF 1.2, support constitutes the following:

- JSP files (.jsp) are the default page language
- JSF 1.2 libraries are added to the project's classpath
- The Faces servlet and servlet mapping are added to the project's deployment descriptor
- A Faces configuration file (faces-config.xml) is included in the project

#### To create a new web application with JSF support:

1. Choose **File > New Project**.
2. From the Java Web category, select **Web Application**.
3. Continue with the wizard until you reach the Frameworks panel. Select JavaServer Faces.

After selecting JavaServer Faces, various configuration options become available to you. You can determine how your project has access to JSF libraries. Click the

**Configuration** tab to specify how the Faces servlet will be registered in the project's deployment descriptor, or change the project's default page language.

4. Click **Finish**. The new project is generated and opens in the IDE.

After you create a web application with JSF support, you can begin using JSF-specific wizards for your project.

### 13.2.2 How to Add Support for a JSF Component Suite

You can add support for the PrimeFaces, ICEfaces or RichFaces JSF component suites when you create an application in the New Project wizard or add support to an existing Java web application in the Project Properties window. To add support for the component suite you need to create a class library for the component suite and specify the location of the required JARs. After you create the library you can select the component suite in the Frameworks panel of the New Project wizard or the Project Properties window.

---

**Note:** To create the class library the required JARs for the component suite must exist on your local file system. The component suite JARs are not included with the IDE.

---

Perform the following steps to create a library for the component suite for an existing project or when creating a new project.

**To add support for a JSF component suite:**

1. Download the required JARs for the component suite to your local file system.
2. For an existing project, right-click the web project's node in the Projects window and choose Properties to open the **Frameworks** panel.  
If you are creating a project in the New Project wizard, the Frameworks panel is the fourth panel in the wizard.
3. Select the **JavaServer Faces** framework in the list of frameworks.  
If the JavaServer Faces framework is not available in the list, click **Add** and select JavaServer Faces from the list of available frameworks.
4. Click the **Components** tab in the Frameworks panel.
5. Select the component suite that you want to add and click **More** to open a dialog for you to select the library for the framework  
If the library for the framework exists you can select the library in the drop-down list and click OK.
6. Click **Create Library** and type a name for the library in the Create New Library dialog box. Click OK.
7. Specify the location of the required JARs for the new library. Click **OK** to create the library.
8. Select the new library in the drop-down list. Click **OK**.

The new library for the component suite should automatically be selected in the drop-down list. The dialog box will display a warning message if any of the specified JARs are incorrect or missing.

9. Click **OK** in the Frameworks panel to close the Project Properties window.

### 13.2.3 How to Add JSF Support to an Existing Web Application

If you want to add JSF support to an existing Java web application, you can do so from your project's Properties window.

The "support" available to you is determined by the JSF version your project uses. For Java EE 7 Web, JSF 2.2 is the default JSF version; for Java EE 6, JSF 2.0 is the default JSF version; projects relying on Java EE 5 (and previous versions) use JSF 1.2.

**To add JSF support to an existing web application:**

1. In the **Projects** window, right-click a standard web project's node and choose **Properties**.
2. Click the **Frameworks** category.
3. Click the **Add** button, then from the list of available frameworks, select JavaServer Faces.
4. Click **OK** to confirm your selection and exit the dialog box, then click **OK** to exit the Project Properties window.

---

**Note:** You can also add support for JSF to the application as one of the steps in the wizard when Generating JSF Pages from Entity Classes.

---

### 13.2.4 How to Create a JSF Page

Use the JSF Page wizard to create Facelets and JSP pages for your project.

**To create a new JSF page:**

1. In the Projects window, right-click your project node and choose **New > JSF Page**. The File wizard opens.  
(If JSF Page is not listed, choose **Other**. Then select the JavaServer Faces category and JSF Page file type. Click **Next**.)
2. In the **File Name** field, type in a name for the file.
3. In **Location**, specify the top-level location for the file. ('Web Pages' is the default option, and places the file in the project's web root).
4. In **Folder**, specify a folder within the selected location, if you require. The **Created File** field provides a read-only path to the new location for the file.
5. Under **Options**, indicate the page type you want to create. (Facelets is the default option for Java EE 6 and Java EE 7 projects.) If you select the JSP option, you can also specify whether you want to create a JSP fragment.
6. Click **Finish**. The new file is generated and opens in the editor.

### 13.2.5 How to Edit a JSF page

You can edit JSF pages using the IDE's source editor. From the Projects window, you can double-click any JSF-related file node to open it in the editor.

**Facelets Files**

The IDE's editor provides the following support when editing Facelets pages.

- **Code completion:**

- **JSF and Facelets tags:** Press Ctrl-Space on JSF and Facelets tags to invoke a pop-up listing valid tag entries for completing a tag you've begun typing.
- **JSF namespaces:** If you apply code-completion for a tag whose namespace is not yet declared in the file, the namespace is automatically added to the page's <html> tag.
- **JSF namespace paths:** If you manually declare a namespace, press Ctrl-Space between the quotes of a namespace declaration to invoke a list of possible paths.
- **EL (Expression Language) expressions:** When typing EL expressions in the editor, press Ctrl-Space to call up a list of possible entries for completing the expression. The pop-up list typically includes JSF managed beans and their properties, property bundle messages, and implicit objects.
- **Documentation:** Press Ctrl-Space on JSF and Facelets tags to invoke a documentation pop-up that describes the given tag.
- **Hints and Error Messages:** The editor provides you with warning and error messages while you edit JSF pages. Errors display with a red badge in the left margin of the editor, and the corresponding code is underlined in red. You can hover your mouse over the badge or underlined code to view a tooltip describing the cause of the error. The editor checks whether:
  - a declared library exists
  - the library matched by the tag prefix contains such a component or tag
  - the tag contains all required attributes
  - all entered attributes are defined in the component's interface
  - undeclared components exist on the page
  - taglib declarations exist without usages
- **Syntax highlighting:**
  - JSF and Facelets tags display in blue.
  - Tag attributes display in green.
  - Tag attribute values display in orange.
  - EL expressions display in black, with light-green background.

### Configuration Files

JSF 2.2 and JSF 2.0 do not require the Faces configuration file (`faces-config.xml`), however, if you add this file to your project, the following support is available.

### Configuration Dialogs

Right-click menu support provides several dialogs that enable you to create entries into the configuration file:

- **Navigation Case dialog:** Right-click in the editor and choose Insert > Navigation Case.
- **Navigation Rule dialog:** Right-click in the editor and choose Insert > Navigation Rule.
- **Managed Bean dialog:** Right-click in the editor and choose Insert > Managed Bean.

## Hyperlinking

Hyperlinking is available in JSF configuration files, enabling you to quickly navigate between a source and its reference. To make use of hyperlinking in configuration files, hover the mouse over a JSF class entity while pressing Ctrl. The class declaration displays as a hyperlink. Clicking on the link opens the appropriate Java class in the Source Editor. In Facelets (and JSP) pages, you can press Ctrl while hovering over managed beans, or accessor methods used within JSF tags. Clicking the hyperlink that displays opens the respective class or property in the Source Editor.

### 13.2.6 How to Create a JSF Facelets Template

You can create Facelets Templates for your JSF 2.x application using the Facelets Template wizard. The wizard creates an XHTML template file using `<h:head>` and `<h:body>` tags, and places associated stylesheets in the `resources/css` folder of your application's web root. You can choose from eight unique layout styles, and specify whether the layout is implemented using CSS or an HTML `<table>` element.

After you create a Facelets template, you can use the Facelets Template Client wizard to generate pages that reference the template. The client will be formatted according to the referenced template.

#### To create a new Facelets template:

1. In the Projects window, right-click your project node and choose **New > Facelets Template**. The Facelets Template wizard opens.

(If Facelets Template is not listed, choose **Other**. Then select the JavaServer Faces category and Facelets Template file type. Click **Next**.)

2. In the **File Name** field, type in a name for the template.
3. In **Location**, specify the top-level location for the file. ('Web Pages' is the default option, and places the file in the given project's web root).
4. In **Folder**, specify a folder within the selected location, if you require. The **Created File** field provides a read-only path to the new location for the template file.
5. For **Layout Style**, specify whether you want the layout to be generated using CSS styles or an HTML table.
6. Select the icon that corresponds to the layout you want generated. (A black layout icon indicates that the icon is selected.)
7. Click **Finish**. The new Facelets template is generated and opens in the editor. The wizard generates a `default.css` file, and a `cssLayout.css` or `tableLayout.css` file, depending on your layout selection.

#### To create a new Facelets template client:

1. In the Projects window, right-click your project node and choose **New > Facelets Template Client**. The Facelets Template Client wizard opens.

(If Facelets Template Client is not listed, choose **Other**. Then select the JavaServer Faces category and Facelets Template Client file type. Click **Next**.)

2. In the **File Name** field, type in a name for the client.
3. In **Folder**, specify a folder within the selected location, if you require. The **Created File** field provides a read-only path to the new location for the template file.
4. For **Template**, click **Browse** and locate the Facelets template that you want the client to use.

5. Select the code element that you want as the root tag in the file.
6. Select the sections of the template that you want the IDE to generate in the file. The available sections are determined by the selected template.
7. Click **Finish**. The new Facelets template client file is generated and opens in the editor. The wizard generates placeholder skeleton code in the client file for each of the sections in the template that you selected.

### 13.2.7 How to Create Composite Components

JSF 2.x has simplified the process of creating composite user interface (UI) components, which can be reused in web pages. You can use the IDE's Composite Component wizard to generate a Facelets template for a JSF composite component.

You can access the Composite Component wizard from the JavaServer Faces category in the IDE's File wizard (Ctrl-N). However, a more intuitive way to prompt the wizard is by highlighting the code snippet from a Facelets page in the editor, then choosing Convert to Composite Component from the right-click menu.

**To create a new composite component:**

1. Open a Facelets page contained in your project. In the **Projects** window, double-click any Facelets Page node to open it in the editor.
2. In the editor, highlight the snippet you want to create a component from.
3. Right-click the highlighted snippet and choose **Convert to Composite Component**. The Composite Component wizard opens, containing the selected snippet in its **Implementation Section** panel.
4. In the **File Name** field, type in a name for the composite component.
5. In **Location**, specify the top-level location for the file. ('Web Pages' is the default option, and places the file in the project's web root).
6. In **Folder**, specify a folder that will contain the composite component. The resources/ezcomp folder is provided by default. If the folder does not already exist, the wizard creates it.

The **Created File** field provides a read-only path to the new location for the component.

7. Click **Finish**. The new composite component source file is generated in the specified location, and a new component tag is inserted into the location in the editor where you highlighted the snippet. The namespace for the component is also automatically added to the page's <html> tag.

### 13.2.8 How to Create a JSF Managed Bean

You can create JSF managed beans for your application using the IDE's Managed Bean wizard. In JSF 2.x, any metadata that you specify in the wizard is translated into annotations that are applied to the managed bean once it is generated. If you prefer to register the managed bean with the application using a Faces configuration file (`faces-config.xml`) instead, the wizard provides the option of doing so.

**To create a new JSF managed bean:**

1. In the Projects window, right-click your project node and choose **New > JSF Managed Bean**. The JSF Managed Bean wizard opens.

(If JSF Managed Bean is not listed, choose Other. Then select the JavaServer Faces category and JSF Managed Bean file type. Click Next.)

2. In the **Class Name** field, type in a name for the managed bean.
3. In **Location**, specify the top-level location for the file. ('Source Packages' is the default option, and places the file in the project's `src/java` folder).
4. In **Package**, specify a package name for the new class. If you specify the name of a package that does not exist, it will be created upon completing the wizard. The **Created File** field provides a read-only path to the new location for the file.
5. (Optional) If you want to register the managed bean with the application using a Faces configuration file, select the 'Add data to configuration file' option. Doing so will prevent the wizard from generating annotations in the new class. If a `faces-config.xml` does not yet exist for the application, one will be created upon completing the wizard.
6. (Optional) Specify configuration details:
  - **Name:** Specifies the name which the managed bean can be invoked by. If left blank, the managed bean can be accessed using the name of the managed bean class, but with the first letter in lowercase.
  - **Scope:** Specifies the scope into which a newly created instance of the specified managed bean will be stored. (Default is `request`; choose `none` if you want to refrain from placing bean instances in any scope).
  - **Description:** A textual description of the element, which can be flagged with a language code using the `xml:lang` attribute.
7. Click **Finish**. The new managed bean is generated and opens in the editor.

### 13.2.9 How to Work with JSF Components in the Palette

You can make use of JSF components listed in the Palette (**Window > Palette**). Components in the Palette represent code snippets which you can drag-and-drop into the opened file in the Source Editor. JSF components include:

- **Metadata:** Invokes a dialog to add name-value pairs within JSF metadata tags. For example, if you specify 'myId' and 'myValue' as a name-value pair, the following code snippet is produced as shown in [Example 13–1](#).

#### **Example 13–1 JSF Metadata Tags**

```
<f:metadata>
  <f:viewParam id='myId' value='myValue' />
</f:metadata>
```

The `<f:metadata>` tag requires that you add the JSF core tag library to your page's root element (typically the `<html>` tag):

```
xmlns:f="http://xmlns.jcp.org/jsf/core"
```

- **JSF Form:** Adds the code snippet to the page as shown in [Example 13–2](#).

#### **Example 13–2 JSF Form Code Snippet**

```
<f:view>
  <h:form>
    </h:form>
</f:view>
```

The JSF Form component requires that you add the JSF core and html tag libraries to your page's root element (typically the `<html>` tag) as shown in [Example 13–3](#).

**Example 13–3 JSF Core and HTML Tag Libraries**

```
xmlns:h="http://xmlns.jcp.org/jsf/html"
xmlns:f="http://xmlns.jcp.org/jsf/core"


- JSF Form from Entity: Invokes a dialog enabling you to associate data from an entity class to fields contained in a JSF form.
- JSF Data Table: Adds the code snippet to the page. as shown in Example 13–4.

```

**Example 13–4 JSF Data Table Code Snippet**

```
<f:view>
  <h:form>
    <h:dataTable value="#{}" var="item">
      </h:dataTable>
    </h:form>
</f:view>
```

The JSF Form component requires that you add the JSF core and html tag libraries to your page's root element (typically the `<html>` tag) as shown in [Example 13–5](#).

**Example 13–5 JSF Core and HTML Tag Libraries**

```
xmlns:h="http://xmlns.jcp.org/jsf/html"
xmlns:f="http://xmlns.jcp.org/jsf/core"


- JSF Data Table from Entity: Invokes a dialog enabling you to associate data from an entity class to fields contained in a JSF data table.

```

The five JSF components included in the Palette can equally be accessed by typing 'jsf' in the editor, then pressing Ctrl-Space.

### 13.2.10 How to Create a JSF Form for Entity Data

You can use the Form from Entity dialog box to generate a JSF form that contains fields for all properties contained in an entity class. You must already have a JSF managed bean created to handle any user data associated with the form.

If you use this dialog without having an associated managed bean, you can enter a name for the managed bean in the dialog, and that name will be used in the page regardless of whether it is valid or not. You can then create a managed bean using the IDE's Managed Bean wizard, or if you use the JSF Pages from Entity Classes wizard, managed beans are generated for all selected entity classes.

**To generate a JSF form from an entity class:**

1. Open any Facelets page in the editor.
2. Make sure that you have declared the JSF tag libraries in the page's root element (typically the `<html>` tag), as shown in [Example 13–6](#).

**Example 13–6 JSF Tag Libraries Declared**

```
xmlns:h="http://xmlns.jcp.org/jsf/html"
xmlns:f="http://xmlns.jcp.org/jsf/core"
```

3. Either type 'jsf' in the editor, press Ctrl-Space, then select the JSF Form from Entity option, or, in the Palette, click and drag the JSF Form from Entity component into the desired location in the editor and release the mouse button. The JSF Form from Entity dialog box opens.
4. In the Entity Bean drop-down, select the entity class that you want to generate the form from. The IDE scans the project for all entity classes and makes them accessible from the drop-down list.

5. In the Managed Bean Property field, select a property. The property you choose must return an instance of the entity bean you selected in the Entity Bean drop-down. (This is typically a getter method for the given property.)
6. (Optional) Select the **Generate read only view** option to create a form that contains read-only fields. When this option is selected, the IDE applies `<h:outputText>` tags for form fields, whereas `<h:inputText>` tags are applied when the option is not selected.
7. Click **OK**. The IDE generates code for your Facelets page. For example, a Customer entity class containing a `customerId` property is displayed as shown in [Example 13–7](#).

**Example 13–7 Customer Entity Class**

```
<f:view>
  <h:form>
    <h1><h:outputText value="Create/Edit" /></h1>
    <h:panelGrid columns="2">
      <h:outputLabel value="CustomerId:" for="customerId" />
      <h:inputText id="customerId" value="#{customerController.selected.customerId}" title="CustomerId" required="true" requiredMessage="The CustomerId field is required." />
      ...
      [ Other fields added here. ]
      ...
    </h:panelGrid>
  </h:form>
</f:view>
```

8. Optionally, remove any of the entries for columns that you do not want to display in your Facelets page.

You can customize the template that specifies the generated code for the JSF Form from Entity dialog. To do so, click the Customize Template link in the dialog, and in the template file make any changes to better suit your work patterns. Save the file. The next time you use the dialog, the changes you made to the template will appear in the generated code.

### 13.2.11 How to Generate a JSF Data Table from an Entity Class

You can use the Data Table from Entity dialog to generate a JSF data table that contains columns for all properties contained in an entity class. In order to make use of this facility, you must already have a JSF managed bean created to handle any back-end data associated with the entity class.

If you use this dialog without having an associated managed bean, you can enter a name for the managed bean in the dialog, and that name will be used in the page regardless of whether it is valid or not. You can then create a managed bean using the IDE's Managed Bean wizard, or if you use the JSF Pages from Entity Classes wizard, managed beans are generated for all selected entity classes.

**To generate a JSF data table from an entity class:**

1. Open any Facelets page in the editor.
2. Make sure that you have declared the JSF tag libraries in the page's root element (typically the `<html>` tag), as shown in [Example 13–8](#).

**Example 13–8 JSF Tag Libraries Declared**

```
xmlns:h="http://xmlns.jcp.org/jsf/html"
xmlns:f="http://xmlns.jcp.org/jsf/core"
```

3. Either type 'jsf' in the editor, press Ctrl-Space, then select the JSF Data Table from Entity option, or, in the Palette, click and drag the JSF Data Table from Entity component into the desired location in the editor and release the mouse button. The JSF Table from Entity dialog box opens.
4. In the **Entity Bean** drop-down list, select the entity class that you want to generate the data table from. The IDE scans the project for all entity classes and makes them accessible from the drop-down list.
5. In the **Managed Bean Property** field, select a property. The property you choose must return a List of objects that correspond to the entity bean you selected in the Entity Bean drop-down.
6. Select a **Template Style** in the drop-down list.
7. Click **OK**. The IDE generates code for your Facelets page. For example, a managed bean that contains a property that returns a List of Product objects would display similarly to what is shown in [Example 13–9](#).

**Example 13–9 Generated Code for a Managed Bean**

```
<f:view>
  <h:form>
    <h1><h:outputText value="List"/></h1>
    <h:dataTable value="#{productController.productItems}" var="item">
      <h:column>
        <f:facet name="header">
          <h:outputText value="ProductId"/>
        </f:facet>
        <h:outputText value="#{item.productId}"/>
      </h:column>
      ...
      [ Other columns added here. ]
      ...
    </h:dataTable>
  </h:form>
</f:view>
```

8. (Optional) Remove any of the entries for columns that you do not want to display in your Facelets page.

You can customize the template that specifies the generated code for the JSF Form from Entity dialog. To do so, click the Customize Template link in the dialog, and in the template file make any changes to better suit your work patterns. Save the file. The next time you use the dialog, the changes you made to the template will appear in the generated code.

### 13.2.12 How to Generate JSF Pages from Entity Classes

If you have already generated entity classes from a database in your project, you can use the JSF Pages from Entity Classes wizard to generate a collection of files, classes and front-end pages that enable you to interact with the database. The purpose of the wizard is to provide template code that facilitates CRUD functionality with a back-end data store.

**To generate JSF pages from an entity class:**

1. Choose **File > New** from the main menu.

2. From the JavaServer Faces category, select JSF Pages from Entity Class and click **Next**. The wizard displays all of the entity classes in the project.
3. Specify the entity classes for which you want to generate JSF pages. To do so, click **Add** (or **Add All**) to move entity classes to the Selected Entity Classes list.
4. Click **Next**.
5. In Step 3: Generate JSF Pages and Classes, specify a name for the following fields:
  - **Session Bean Package**. This package contains the session facades for the entity classes.
  - **JSF Classes Package**. This package contains JSF session-scoped, managed beans.
  - **JSF Pages Folder**. This folder contains the Facelets pages generated for the entity classes.
  - **Localization Bundle Name**. The properties bundle contains all the text used in the Facelets pages as key-value pairs.
6. Select the template that you want the IDE to use to generate the JSF pages. You can choose to use either standard JSF templates or PrimeFaces templates.
7. Click **Finish**.

### 13.2.13 How to Generate a Validation Constraint

You can use a validation constraint to ensure that the values of fields, methods, and types are valid and within a specified constraint. You apply a validation constraint by placing annotations on a field, method, or class, for example on a managed bean or persistent entity class.

You can use the validation constraints that are defined as part of the Bean Validation API or create custom constraints by specifying the constraints for the field or property and the annotation that is associated with the custom constraints. You can specify the constraints in the validation constraint class or create a validator class for the constraint.

**To generate a validation constraint:**

1. Choose **File > New** from the main menu.
2. From the Bean Validation category, select Validation Constraint and click **Next**.
3. Specify the class name and package for the class.
4. (Optional) Select **Generate Validator Class**

The wizard can create a constraint validator class for you if you select the Generate Validator Class option. You can edit the generated validator class to create a custom validator for the constraint. The constraint class is automatically marked with the `@Constraint` annotation and the name of the validator class.

5. Click **Finish**.

## 13.3 Working with the Spring Framework

The IDE provides built-in support for the Spring Framework. The Spring Framework libraries (versions 3.2 and 2.5) are packaged with the IDE and are available in the Ant Library Manager. They can also be automatically added to the project classpath during

project creation. Support for Spring XML bean configuration files is also provided. The configuration files are managed by the following functional components:

- **New Spring XML Configuration File wizard.** XML bean configuration files can be easily created. The wizard provides a list of standard Spring namespaces to choose from when creating the configuration file.
- **Spring Framework panel.** This panel, located in the Project Properties window, enables you to specify all the Spring configuration files in a project, as well as organize them into groups.

The following functionality is provided for Spring bean configuration files:

- **Code completion.** Invoked in Spring XML configuration files for Java classes as well as bean references.
- **Navigation.** Hyperlinking of Java classes and properties mentioned in Spring bean definitions, as well as hyperlinking to other Spring bean references.
- **Refactoring.** Renaming of references to Java classes in Spring XML configuration files.
- **Go to Spring Bean dialog.** The Go to Spring Bean dialog enables you to quickly locate Spring beans contained in projects opened in the IDE.

Spring Web MVC Support integrates the IDE's Spring Framework 3.2 core support for Java applications, and adds functional support for developing MVC-structured web applications by including the following features:

- Spring Web MVC structural template listed as an option with the New Web Application wizard.
- Spring Framework libraries automatically added to the project path.
- Configuration settings are provided, such as naming and mapping the default dispatcher servlet.
- Automatic registration of the JSTL library.
- Controller wizards for common Spring Web MVC artifacts, such as `AbstractController` and `SimpleFormController` are available for quick code template generation.

For more information on the Spring Framework, see the following resources:

- Introduction to Spring Web MVC at  
<https://netbeans.org/kb/docs/web/quickstart-webapps-spring.html>
- Official Spring Framework documentation at  
<http://www.springsource.org/documentation>

The IDE provides Spring support for various Java-based applications, such as Java SE, EJB, and web applications. You can add Spring support to a project by either adding the Spring Framework 3.2 library to a project's classpath, or by creating a new Spring configuration file using the Spring XML Configuration File wizard. Along with creating the configuration file, this wizard will also offer to add the Spring library to the classpath if it has not already been added.

#### To add the Spring Framework library to a project classpath:

1. In the Projects window, right-click the Libraries node and choose **Add Library**.
2. In the Add Libraries dialog, select the Spring Framework 3.2.0 library and click **Add Library**.

The Spring Framework libraries are added to the project classpath, and they become visible from the project's Libraries node.

### 13.3.1 How to Create a New Project with Spring MVC Framework Support

The Spring framework libraries and supporting files are bundled with the IDE.

**To create a new web application with Spring Web MVC support:**

1. Choose **File > New Project**. The New Project wizard displays.
2. Under Categories select Web, then under Projects, select Web Application. Click **Next**.
3. Continue through the wizard to Step 4, the Frameworks panel, and select Spring Web MVC 3.2.
4. Under Spring Web MVC Configuration, specify the dispatcher name and mapping you want to use in the application. Click **Finish**.

---

**Note:** The JSTL library version 1.1, is bundled by default with the web application. To exclude JSTL, in the Frameworks panel click on the Libraries tab and deselect the Include JSTL option.

---



---

**Note:** Dispatcher mappings are based on the Java Servlet Specification Section SRV.11.2. In the web application deployment descriptor, the following syntax is used to define standard mappings:

- A string beginning with a '/' character and ending with a '/\*' postfix is used for path mapping.
  - A string beginning with a '\*' '.' prefix is used as an extension mapping.
  - A string containing only the '/' character indicates the default servlet of the application. In this case the servlet path is the requested URI minus the context path and the path info is null.
  - All other strings are used for exact matches only.
- 

### 13.3.2 How to Add Spring MVC Framework Support to an Existing Application

The Spring framework libraries and supporting files are bundled with the IDE.

**To add Spring Web MVC support to an existing web application:**

1. In the **Projects** window, right-click a web application project node and choose **Properties**.
2. Under **Categories**, click Frameworks. Then under Used Frameworks click the **Add** button.
3. In the **Add a Framework** dialog that displays, select Spring Web MVC and click **OK**.
4. Optionally specify any configuration settings, such as the default dispatcher name and mapping. You can also specify the version of Spring libraries to add to your classpath (4.0.1 or 3.2.7). Click **OK** to exit the **Project Properties** dialog.

Spring framework libraries and artifacts are added to the selected project, such as `applicationContext.xml` and `dispatcher-servlet`, where `{dispatcher}` is the user-specified dispatcher name.

### 13.3.3 How to Create a Spring Configuration File

The IDE provides support for managing Spring XML configuration files. You can use the New Spring XML Configuration File wizard to create a configuration file.

#### To create a new Spring configuration file:

1. Right-click on a project node in the Projects window and choose **New > Other**.
2. This step depends on the type of your project. For a web project, select the Spring category under **Categories**. For a Java SE or EJB project select the Other category. Then under **File Types**, select Spring XML Configuration File. Click **Next**.
3. In the **Name and Location** panel, specify a name and location for the file. To specify a location, you can type in a path relative to your project in the **Folder** text field. Otherwise, click **Browse** to navigate to a specific location. If you leave the text field blank, the file will be created in the top-most directory.
4. If you have already added configuration file groups to your project, you can optionally add the new configuration file to any of those groups under **Add File to Configuration File Group(s)**.
5. Add namespaces to the configuration file. The namespaces displayed are the standard set provided by the Spring Framework library. Click **Finish**.

The IDE generates the new configuration file and opens it in the Source Editor.

After the file is generated based on your settings, you can manage it and other configuration files by accessing the Spring Framework category in the **Project Properties** window (Choose **Properties** from a project's right-click menu).

### 13.3.4 How to Organize Spring Configuration Files

The IDE provides support for organizing Spring configuration files. This support allows you to specify all your configuration files and organize them into groups, so that they can refer to each other in features such as code completion and hyperlinking. The Spring Configuration Group panel, located in the Project Properties window, enables you to organize Spring configuration files.

The Spring Framework category contains two tabs: Configuration Files and Configuration File Groups. The Configuration Files tab lists all the Spring configuration files in a project, and the Configuration File Groups lists all configuration file groups.

#### 13.3.4.1 Accessing the Spring Configuration Group Panel

##### To access the Spring Configuration Group panel:

1. Make sure the Spring support has been enabled, for example by adding the Spring Framework library to the project.
2. Right-click on the project node in the **Projects** window and choose **Properties**.
3. Under **Categories**, select Spring Framework.

### 13.3.4.2 Specifying Spring Configuration Files

Select the Configuration Files tab. You can use the **Add File** and **Remove File** buttons to add and remove files from the list, or you can use the **Detect Files** button to let the Spring support try to autodetect the configuration files.

---

**Note:** Files created using the New Spring XML Configuration File wizard are added to this list automatically by the wizard.

---



---

**Note:** If you do not mention a configuration file here, it will be ignored by the Spring features that process all Spring configuration files in the project, such as refactoring. Features which work in single files, such as code completion, will continue to work. It is good practice to specify all your configuration file in this list.

---

### 13.3.4.3 Organizing Spring Configuration Files into Groups

Groups are useful for setting up parent/child relationships between configuration files. For example, in a web application it is typical to have a `applicationContext.xml` file containing business logic service beans, and a `spring-servlet.xml` file containing Spring Web MVC beans such as controllers. The beans in `spring-servlet.xml` are often initialized with service beans from `applicationContext.xml`, and this fact needs to be reflected in features such as code completion and hyperlinking. In order to set up such a relationship, you would create a group containing these two files.

Select the Configuration File Groups tab. You can use the **Add Group**, and **Remove Group** button to add and remove groups. After adding a group, you can add/remove files to/from it using the **Add Files** and **Remove Files** buttons.

## 13.4 Working with the Struts Framework

The IDE provides built-in support for developing MVC-structured web applications with Struts 1.3.10. Functional support includes the following features:

- The Struts structural template is listed as an option with the New Web Application wizard.
- Struts Framework libraries are automatically added to the project classpath.
- Configuration files are provided with default settings; the Struts action servlet is registered in the project's deployment descriptor, and the `struts-config.xml` is generated and registered with the deployment descriptor.
- Wizards for common Struts artifacts, such as Action classes and ActionForm beans.
- Automatic registration of the JSTL library.

For more information on the Struts Framework, see the following resources:

- Introduction to the Struts Web Framework (NetBeans tutorial) at  
<https://netbeans.org/kb/docs/web/quickstart-webapps-struts.html>
- Official Struts Framework documentation at  
<http://struts.apache.org/primer.html>

### 13.4.1 How to Create a New Project with Struts Framework Support

The Struts tag libraries and supporting files, such as the struts-config.xml file, are bundled with the IDE.

#### To create a new application with Struts support:

1. Choose File > New Project.
2. From the Java Web category, select Web Application.
3. Continue with the wizard until you reach the **Frameworks** panel. Select Struts and click **Finish**. The new project is generated and opens in the IDE.

After you add Struts support, you can use Struts-specific wizards for your project, including the New Struts Action Wizard and the New Struts ActionForm Bean Wizard. Right-click the struts-config.xml file in the Projects window to access Struts-specific wizards.

### 13.4.2 How to Add Struts Framework Support to an Existing Application

The Struts tag libraries and supporting files, such as the struts-config.xml file, are bundled with the IDE.

#### To add Struts support to an existing web application:

1. In the Projects window, right-click a standard web application project's node and choose **Properties**.
2. Click **Frameworks** and select **Struts**.
3. Click **OK** to confirm your selection and exit the **Project Properties** dialog box.

After you add Struts support, you can use Struts-specific wizards for your project, including the New Struts Action Wizard and the New Struts ActionForm Bean Wizard.

## 13.5 Working with the Hibernate Framework

Hibernate is a framework that enables your applications to interact with databases using Object Relational Mapping (ORM). The IDE provides Hibernate support for various Java-based applications, such as Java SE, EJB, and web applications.

The IDE has wizards for generating the following files in your application:

- Hibernate configuration file
- Hibernate reverse engineering file
- Hibernate mapping files
- Hibernate utility class
- POJOs and Hibernate mapping files from a database

The IDE also offers support for editing the following Hibernate files and queries:

- Hibernate configuration files
- Hibernate mapping files
- Hibernate Query Language (HQL) queries

For more about Hibernate features and using Hibernate in your application, see the following site at <http://www.hibernate.org/>.

The following steps outline the basic process of working with Hibernate:

1. Create a project and open a database connection in the Services window.
2. Configure the project:
  1. Create a Hibernate configuration file.
  2. Edit the configuration file.
  3. Create the Hibernate utility class.
3. Create Classes by doing either of the following:
  - Create individual POJOs and mapping files.
  - Use the POJOs and Hibernate mapping files from database wizard.
4. Code Application:
  1. Create HQL queries.
  2. Edit source files.

### 13.5.1 How to Create a New Project with Hibernate Support

The Hibernate libraries and wizards to generate supporting files are bundled with the IDE.

**To create a new web application with Hibernate support:**

1. Choose **File > New Project**. The New Project wizard displays.
2. Under **Categories** select Java Web, then under **Projects**, select Web Application. Click **Next**.
3. Continue through the wizard to Step 4, the **Frameworks** panel, and select Hibernate 4.2.6.
4. Under the **Hibernate 4.2.6 Configuration** section of the panel, choose the database connection for the application from the drop-down list. Click **Finish**.

When you click Finish the IDE creates the Hibernate configuration file `hibernate.cfg.xml` in a source package with the name `<default package>`. The wizard also adds the Hibernate library to the project classpath. After you create a web application with Hibernate support, you can use wizards to generate Hibernate mapping files, a helper file and POJOs from a database.

### 13.5.2 How to Add Hibernate Support to an Existing Application

The IDE provides Hibernate support for various Java-based applications, such as Java SE, EJB, and web applications. You can add Hibernate support to a project by either adding the Hibernate library to a project's classpath, or by creating a new Hibernate configuration file using the Hibernate Configuration File wizard. Along with creating the configuration file, this wizard will automatically add the Hibernate library to the classpath if it has not already been added.

---

**Note:** If you are creating a web application project with the New Project wizard you can add Hibernate support to the web application by selecting Hibernate in the Frameworks page of the wizard.

---

**To add the Hibernate library to a project classpath:**

1. In the **Projects** window, right-click the **Libraries** node and choose **Add Library**.

2. In the **Add Libraries** dialog, select the Hibernate library and click **Add Library**.

The Hibernate support libraries are added to the project classpath, and they become visible from the project's Libraries node.

### 13.5.3 How to Create the Hibernate Configuration File

To use Hibernate your application must have a Hibernate configuration file that contains the database connection details enabling the application to connect to a database. The file also contains information about the location of Hibernate mapping files. You can edit this file if you need to change the database connection details. You can have multiple Hibernate configuration files, but the default name for the configuration file used by Hibernate is `hibernate.cfg.xml`.

A Hibernate configuration file is created automatically if you create a web application and you select the Hibernate framework in the New Project wizard.

You can also create a Hibernate configuration file using the New File wizard if you want to use Hibernate in your application, for example in a Java SE application.

**To create `hibernate.cfg.xml` using the New File wizard:**

1. Open the New File wizard.
2. Select Hibernate from the Categories list and Hibernate Configuration Wizard from the File Types list. Click **Next**.
3. Type `hibernate.cfg` for the name and specify a location. Click **Next**.
4. Select a Database Connection from the drop down list. Click **Finish**.

When you click **Finish**, the file `hibernate.cfg.xml` is created in the location you specified and the file opens in the Source Editor. If you did not specify a location the file is created in a source package with the name `<default package>`. The wizard also adds the Hibernate library to the classpath if it has not already been added.

### 13.5.4 How to Edit the Hibernate Configuration File

The `hibernate.cfg.xml` editor enables you edit the configuration properties for an application that uses the Hibernate framework. You can edit the `hibernate.cfg.xml` file in a visual editor or edit the XML directly. In the visual editor you can use the Add, Edit and Remove buttons to open dialog boxes to modify the properties. In the XML editor you can use the IDE's code completion to assist you.

You open the editor by expanding the project's Source Packages node in the Projects window and double-clicking the `hibernate.cfg.xml` file located under the `<default package>` node.

Many of the properties in the configuration file are supplied and modified by other wizards in the IDE. For example, if you create Hibernate mapping files using the Hibernate Mapping File wizard, `hibernate.cfg.xml` is automatically updated with information about the mapping files.

The `hibernate.cfg.xml` visual editor contains the following sections for editing the configuration settings:

- **JDBC Properties.** This section enables you to set the JDBC connection information for your datasource, including the JDBC driver and the location of your datasource.
- **Datasource Properties.** This section enables you to set the datasource properties, including the username and password for accessing the datasource.

- **Mappings.** This section enables you to specify the Hibernate mapping files (.hbm.xml) used in the application.
- **Cache.** This section enables you to add Class and Collection Cache properties.
- **Events.** This section enables you to specify the listener classes for handling Hibernate session events.
- **Optional Properties.** This section enables you to configure optional Hibernate properties such as JDBC connection properties, Hibernate SQL dialects and transaction properties.
- **Security.** This section enables you to configure security roles.

You can also edit any of the properties in the XML editor and use code completion to set the properties and values.

### 13.5.5 How to Create Hibernate Mapping Files

Hibernate mapping files are XML files that contain object relational mapping (ORM) data that Hibernate uses to determine how classes are mapped to table columns and primary keys.

You can use the following wizards to generate Hibernate mapping files:

- **Hibernate Mapping Wizard.** This wizard generates a single mapping file for a specified class. After you create the mapping file you need to edit the mapping file in the XML editor to map the fields in the class to the corresponding columns in the table. You can use the IDE's code completion feature to help you edit the mapping file.
- **Hibernate Mapping Files and POJOs from Database.** If you use this wizard you can set the option for the IDE to automatically generate Hibernate mapping files for the tables you select in the wizard. The wizard automatically modifies the Hibernate configuration file to specify the Hibernate mapping files.

#### To create a Hibernate mapping file for a class:

1. Right-click the package node containing the class to be mapped and choose **New > Other** to open the New File wizard.
2. Select **Hibernate** from the **Categories** list and **Hibernate Mapping Wizard** from the **File Types** list. Click **Next**.
3. Type <CLASS\_NAME>.hbm for the file name, where <CLASS\_NAME> is the name of the class that you want to map. Click **Next**.
4. Specify the Class that you want to map to a table.  
You can click the button to locate the class.
5. Choose the corresponding database table from the drop down list. Click **Finish**.

When you click **Finish**, the mapping file is created in the source package you specified and the file opens in the Source Editor. The Hibernate configuration file is modified to specify the mapping file.

### 13.5.6 How to Generate Hibernate Mapping Files and POJOs from a Database

In addition to writing Hibernate mapping files and POJOs (Plain Old Java Objects) from scratch, you can also generate the mapping files and POJOs from a connected database with the Hibernate Mapping Files and POJOs from Database wizard.

**To generate Hibernate mapping files and POJOs from a database:**

1. Right-click the project node in the Projects window and choose **New > Other** to open the New File wizard.
2. In the New File wizard, select Hibernate Mapping Files and POJOs from Database from the Hibernate category.
3. Select the Configuration File (`hibernate.cfg.xml`) and Reverse Engineering File (`hibernate.reveng`) from the dropdown lists.

---

**Note:** The configuration file contains the database connection details. If a connection to the database is not open you may be prompted to supply a username and password to connect to the database. The database must be running.

---

4. Specify any **General Settings** options for the generated files.
5. Select Domain Code to generate the POJOs.
6. Select Hibernate XML Mappings to generate the mapping files.
7. Select an existing package from the Package drop down list or type the name of a new package.
8. Click **Finish**.

When you click **Finish** the IDE generates the following files:

- POJOs based on the selected specified tables (if you selected Domain Code in the wizard)
- Hibernate mapping files based on the specified tables (if you selected Hibernate XML Mappings in the wizard)
- A `hibernate.reveng` reverse engineering file in the specified location

### 13.5.7 How to Create a Hibernate Reverse Engineering File

To use the Hibernate Mapping Files and POJOs from a Database wizard, you need to first create a Hibernate reverse engineering file (`hibernate.reveng.xml`). The IDE provides a wizard for generating the file based on the data contained in the Hibernate configuration file and the tables you are using in your application.

The reverse engineering file is an XML file that can be used to modify the default settings used when generating Hibernate files from the metadata of the database specified in `hibernate.cfg.xml`. The file can be used to explicitly specify the database schema that is used, to filter out tables that should not be used and how JDBC types are mapped to Hibernate types.

**To create `hibernate.reveng.xml` using the New File wizard:**

1. Open the New File wizard.
2. Select Hibernate from the Categories list and Hibernate Reverse Engineering Wizard from the File Types list. Click **Next**.
3. Type `hibernate.reveng` for the name and specify the Folder where you want to save the file. Click **Next**.
4. Select the Configuration File from the drop down list.

---

**Note:** The configuration file contains the database connection details. If a connection to the database is not open you may be prompted to supply a username and password to connect to the database. The database must be running.

---

5. Select any tables in the **Available Tables** pane and click the Add to move the tables to the Selected Tables pane. Any tables related to the tables you select are automatically added to the list in the **Selected Tables** pane.
6. Click **Finish**.

When you click **Finish**, the file `hibernate.reveng.xml` is created in the location you specified and the file opens in the Source Editor.

### 13.5.8 How to Create a Hibernate Utility Helper File

To use Hibernate you need to create a helper class that handles startup and that accesses Hibernate's SessionFactory to obtain a Session object so that you can load and store objects. The helper class calls `configure()` and loads the `hibernate.cfg.xml` configuration file. The helper class then builds the SessionFactory to obtain the Session object.

**To create the `HibernateUtil.java` helper class:**

1. In the Projects window, right-click the Source Packages node and choose **New > Other** to open the New File wizard.
2. Select **Hibernate** from the **Categories** list and `HibernateUtil.java` from the **File Types** list. Click **Next**.
3. Type `HibernateUtil` for the class name. Click **Finish**.

When you click **Finish**, the file `HibernateUtil.java` opens in the Source Editor. You do not need to edit the file.

### 13.5.9 How to Create and Execute a HQL Statement or Script

You can use the Hibernate Query Language (HQL) Editor to write, edit and execute HQL queries within the IDE. After you create and test the HQL query in the editor you can copy the query to the source code and mapping files. You can right-click in the query editor window to select, copy and paste queries.

The syntax of the HQL query language is very similar to the syntax used in SQL queries but HQL is fully object-oriented and can use objects and properties to represent SQL queries. The syntax of HQL queries is not dependent upon the database the queries are run against. The HQL queries are converted at runtime to the SQL query appropriate for the database. In addition to supporting standard clauses and aggregate functions, HQL also supports polymorphic queries and subqueries.

For more about Hibernate Query Language, see Hibernate documentation at <http://docs.jboss.org/hibernate/orm/4.2/manual/en-US/html/ch16.html>.

**To create and execute a HQL statement or script:**

1. Expand the `<default packages>` node under the Source Packages node in the Projects window.
2. Right-click the `hibernate.cfg.xml` configuration file and choose **Run HQL Query** to open the HQL Editor.

3. Enter the HQL query in the top pane of the editor window.
4. Click **Run HQL** ( in the toolbar. Alternatively, you can right-click in the query editor window and choose **Run HQL** from the popup menu.

When you execute an HQL query, you see the result of the query in the bottom pane of the HQL editor window. The HQL query is executed against the database specified in the project's Hibernate configuration file (`hibernate.cfg.xml`) that is selected in the editor toolbar. You can click the SQL button above the bottom pane to view the SQL query that corresponds to your HQL query.

## 13.6 Working with the Grails Framework

Grails aims to bring the "coding by convention" paradigm to Groovy. It's an open-source web application framework that leverages the Groovy language and complements Java Web development.

The IDE's support for Grails encompasses the following features:

- Registration for Grails distribution in the Options window (**Tools > Options > Groovy**).
- Grails Application project template listed in the New Project wizard.
- Grails libraries automatically added to the project path.
- Grails Plugins Manager for installing Grails plugins into the current project.
- Integration of common Grails command line functionality settings.

Grails ships with a lot of command line functionality out of the box. The IDE provides menu items and other tools that invoke the related scripts:

- **grails compile.** Right-click the application and choose **Compile**.
- **grails create-app.** Create a new Grails application.
- **grails create-controller.** Right-click the Controllers node and choose **New > Grails Controller**.
- **grails create-domain-class.** Right-click the Domain Classes node and choose **New > Grails Domain Class**.
- **grails create-script.** Right-click the Scripts node and choose **New > Grails Gant Script or Groovy Script**.
- **grails create-views.** Right-click a Grails domain class node and choose **Create View**.
- **grails generate-all.** Right-click a Grails domain class node and choose **Generate All**.
- **grails run-app.** Right-click the application and choose **Run**.
- **grails shell.** Right-click the application and choose **Grails Shell**.
- **grails stats.** Right-click the application and choose **Statistics**.
- **grails upgrade.** Right-click the application and choose **Upgrade**.
- **grails war.** Right-click the application and choose **Create War File**.

### 13.6.1 How to Create a New Project with Grails Framework Support

The Grails framework libraries and supporting files are bundled with the IDE.

**To create a new web application with Grails support:**

1. Choose **Tools > Options > Miscellaneous** and set your Grails installation directory in the Grails tab.
2. Choose **File > New Project**. The New Project wizard displays.
3. Under Categories select Groovy, then under Projects, select Grails Application. Click **Next**.
4. Set a project name and location. Click **Finish**.

The IDE creates a new Grails application and displays its structure in the Projects window.

### 13.6.2 How to Install a Grails Plugin into a Grails Application

You can use the IDE to install plugins into a Grails application. The plugins can be obtained from the online Grails plugin repository, as well as locally from disk.

**To install a plugin into a Grails application:**

1. Right-click a Grails application and choose **Plugins**.
2. Select one of the following:
  - Installed. Shows the plugins currently installed in the application. Select an installed plugin and click **Uninstall** to uninstall it.
  - New plugins. Click **Reload Plugins** to let the Plugins Manager access the online Grails repository and display the results in the **New Plugins** tab. Select a plugin and click **Install** to install the plugin. Alternatively, browse to a Grails plugin locally using the **Browse** button.
3. Click **Close**.

If you installed a new Grails plugin, you can view it by opening the **Files** window (Ctrl-2) and then expanding the "plugins" folder.



# 14

---

## Developing Enterprise Applications

This chapter describes how to use NetBeans IDE support for Java EE to develop Web and server side applications.

This chapter contains the following sections:

- [About Developing Enterprise Applications](#)
- [Adding Modules to the Project](#)
- [Adding Resources to the Project](#)
- [Editing Deployment Descriptors](#)
- [Building Enterprise Applications](#)
- [Verifying Enterprise Applications](#)
- [Deploying Enterprise Applications](#)
- [Packaging Enterprise Applications](#)
- [About Docker Platform Support](#)

### 14.1 About Developing Enterprise Applications

An enterprise application is a collection of web application and EJB modules that are configured to work together when deployed to a Java EE application server. The enterprise application contains information about how the modules work together.

The enterprise application also contains information about how the modules work with the application server to which the enterprise application is deployed. For example, if any entity beans use container-managed persistence, the enterprise application tells the application server what transaction services are needed.

An enterprise application has no source files of its own. It only contains deployment descriptors and other configuration files. At compile time, the archive files (JAR files and WAR files) for each of the enterprise application's modules are built and assembled into one Enterprise Archive (EAR) file. This file is deployed to the application server.

The following steps outline the basic process of working with enterprise applications:

1. Create the application and module projects:
  - a. Register an application server.
  - b. Create the project for the enterprise application.
2. Code the modules:

- a. Create web components in the web application modules.
  - b. Create new session beans and message-driven beans.
  - c. Generate entity classes or entity beans from existing database tables or use the Database Explorer to create database tables.
  - d. Call enterprise beans from web application modules and other enterprise beans.
  - e. Use server resources like databases and JMS messages.
3. Configure the application as follows:
  1. Edit the enterprise application's deployment descriptors.
  2. Select libraries and other external resources to bundle with the EAR file.
  3. Set runtime options such as the application's default web page and target application server.
4. Build the project by doing either of the following:
  - Set the project as the main project and choose **Run > Build Main Project** (F11) from the main menu.
  - Right-click the project node in the Projects window and choose **Build**.
5. Deploy and run the application by doing either of the following:
  - Right-click the enterprise application project and choose **Run**. Module projects are compiled, new and changed files are copied from module projects' build directories to the EAR's deployment directory. The EAR "client" is triggered (for example, a web application opens in the browser).
  - Right-click any of the application's module projects and choose **Run** to deploy the individual module without deploying other modules in the application.

## 14.2 Adding Modules to the Project

EJB or web application modules can be deployed either by themselves or as a part of an enterprise application. Likewise, Java application clients can be either regular Java applications, or can be full Java EE application client modules. Adding an EJB module, web module, or application client module to an enterprise application lets you further configure the way those modules interact with each other.

When you add a module to an enterprise application, the IDE does the following:

- Lists the module under the enterprise application's Java EE Modules node.
- Adds the module project to the enterprise application project's list of required projects. This means that the module project is cleaned and built whenever you clean and build the enterprise application project.
- Adds a module reference to the enterprise application's general deployment descriptors.
- Packages the module's build output (JAR file or WAR file) into the enterprise application's EAR file when building the project.

## 14.2.1 How to Add a Module to an Existing Project

**To add an existing module to an enterprise application:**

1. Open the module project and the enterprise application project.
2. In the Projects window, right-click the enterprise application node for the Java EE application and choose **Add Java EE Module**.
3. In the dialog box, select the module and click **OK**.

---

**Note:** You can only add standard projects to an enterprise application. You cannot add free-form projects to an enterprise application because the build and deploy process is not controlled by the IDE.

---

## 14.3 Adding Resources to the Project

When you package an enterprise application as an EAR file, the default package includes only those files that are part of the enterprise application. To add a file that is external to the enterprise application to the EAR package, use the Project Properties dialog box. External resources might include JAR files, libraries, or other projects that reside outside the enterprise application.

### 14.3.1 How to Add an External Resource to an EAR File

**To add external resources to an EAR file:**

1. In the Projects window, right-click the project node and choose **Properties**.
2. In the left panel, click **Packaging**.
3. In the **Additional content to include** section, click **Add JAR/Folder**, **Add Library**, or **Add Project**.
4. (Optional) Exclude content from the project using the **Add Filter** button.

## 14.4 Editing Deployment Descriptors

An enterprise application generally includes two types of deployment descriptors. In Java EE enterprise applications, the descriptor files are optional and you can use annotations to specify most deployment settings and resources. J2EE 1.4 enterprise applications will generally require the following descriptor files.

- A general J2EE deployment descriptor that configures deployment settings on any J2EE-compliant implementation. It describes the enterprise application, which components it uses, how the components relate to each other, and which resources they use. The general enterprise application deployment descriptor is called `application.xml`.
- A server-specific deployment descriptor that configures deployment settings for the application server to which you are deploying. If you are deploying your application to the Sun Java System Application Server or GlassFish, the IDE provides a graphical deployment descriptor editor and automatically updates the deployment descriptor as you edit your files. For all other application servers, you have to write the server-specific deployment descriptors yourself.

Entries in these files are automatically generated when you add a module to the enterprise application. Both deployment descriptors are XML files that are stored in the enterprise application's META-INF folder. You can edit both files in the Source Editor. The `glassfish-application.xml` file can also be edited in a visual editor, as described below.

#### 14.4.1 How to Edit an Enterprise Application's Deployment Descriptors

**To edit an enterprise application's deployment descriptors:**

1. In the Projects window, locate the project and expand its Configuration Files node. Alternatively, in the Files window, expand the project's `src/conf` subfolder.
2. Double-click the node to open the file. The `glassfish-application.xml` file opens in the `glassfish-application.xml` Visual Editor. To open it in the Source Editor, right-click it in the Projects window and choose **Edit**.
3. Edit the elements. You can use code completion features in the Source Editor to ensure that the elements are syntactically correct.
4. (Optional) Right-click in the Source Editor and choose **Validate** from the contextual menu to verify that the document follows the XML schema for web deployment descriptors.

---

**Note:** If you have a Java EE 5 or Java EE 6 enterprise application that is deployed to a Java EE container, you can specify most deployment settings using annotations. However, you have the option of using XML descriptors as an alternative to annotations or to supplement or override some annotations.

---

### 14.5 Building Enterprise Applications

You can build an enterprise application project from the main menu or an individual project from its popup menu.

#### 14.5.1 How to Build an Enterprise Application Project and Sub-Projects

**To build an enterprise application project and its modules:**

1. Select the enterprise application project in the Projects window.
2. Choose **Run > Build Project** (F11) from the main menu.

Alternatively, you can right-click the project node in the Projects window and choose **Build**.

---

**Note:** If the enterprise application project is set as the main project, you can choose **Run > Build Main Project** (F11) from the main menu. You can set a project as the main project by right-clicking a project in the Projects window and choosing **Set as Main Project** or by choosing **Run > Set Main Project** from the main menu and selecting the project in the sub-menu.

---

**To build an individual enterprise application project and its modules:**

Right-click the project's node in the Projects window and choose **Build**.

When you build an EJB module as part of an enterprise application, the IDE does the following:

- Compiles the classes for each of the enterprise application's modules to the module projects' build/ear-module folder.
- Places any JAR files, libraries, or directories that are on the classpath of the module projects and are scheduled for inclusion in deployment in the enterprise project's build folder.
- Builds each module's JAR file or WAR file to the module project's dist folder and copies it to the enterprise application project's build folder. The module JAR files and WAR files do not contain any of the module's classpath elements.
- Puts a Class-path entry in each of the modules' manifest file for all included JAR files and puts a . entry if there is at least one directory on the project's classpath scheduled for inclusion in deployment. This mechanism ensures that the JAR files and directories can be used from the modules even if they reside in the EAR project and not in the modules themselves.
- Builds the enterprise application EAR file to the enterprise application project's dist folder.

## 14.6 Verifying Enterprise Applications

Before you deploy an enterprise application or any stand-alone web or EJB module to the application server, you should verify it, to ensure it is properly implemented to the Java EE specification. When the target server for the project is the Sun Java System Application Server or GlassFish application server, you can use the verifier tool to validate both Java EE and application server-specific deployment descriptors against their corresponding DTD files and display errors and warnings if a module or application is not compliant. You can verify deployment descriptors in EAR, WAR, RAR, and JAR files.

### 14.6.1 How to Verify a Project's Deployment Descriptors

The verifier tool is not simply an XML syntax verifier. Rules and interdependencies between various elements in the deployment descriptors are verified. Where needed, classes in your module are examined to ensure that what is referred to in the deployment descriptors actually exists and will work when deployed.

#### To verify a project deployment descriptors:

1. Right-click any web application module, EJB module, or enterprise application project in the Projects window and choose **Verify**.

The Verifier window opens and displays all the results.
2. Filter the results according to your needs:
  - Click **Failures Only** to display only failures.
  - Click **Failures and Warnings Only** to display only failures and warnings.
3. Make appropriate changes, if any, and verify the project again.

---

**Note:** Not all failures to comply with the Java EE specification will cause the application to fail. However, modules that fail to comply with the specification may not be portable between application servers. The verifier tool helps you ensure your code is not locked into running only on a specific application server.

---

## 14.7 Deploying Enterprise Applications

Unless your EJB modules and web application modules are designed to be deployed as stand-alone modules, you should always deploy them by deploying the enterprise application that contains them. If you run the Run or Deploy commands on an individual module that is part of an enterprise application, the IDE only deploys that module itself. Any changes you have made in the other modules in the project are not propagated to the server.

### 14.7.1 How to Run an Enterprise Application

#### To run an enterprise application:

In the Projects window, right-click the enterprise application project and choose **Run**.

1. Sub-projects are compiled.
2. New and changed files are copied from a sub-project's build directory to the EAR's deployment directory.
3. The target application server and domain start.
4. The EAR's deployment directory is used to redeploy the application.
5. The "client" for the EAR is triggered (i.e., a web application opens in the IDE's default browser).

### 14.7.2 How to Deploy an Enterprise Application

#### To deploy an enterprise application:

In the Projects window, right-click the enterprise application project and choose **Deploy**.

1. Sub-projects are compiled.
2. The target application server and domain start.
3. If the application has already been deployed, any active sessions and processes are terminated, and the application is undeployed from the server.
4. The EAR project's deployment directory is cleared (all content is deleted).
5. All the files in the build directories for each of the sub-projects get copied into the EAR's deployment directory.
6. The EAR's deployment directory is used to redeploy the application to the target application server and domain.

### 14.7.3 How to Undeploy an Enterprise Application

**To undeploy an enterprise application (GlassFish):**

1. In the Services window, expand the **Servers > GlassFish** node.

---

**Note:** The server must be running and registered with the IDE.

---

2. Expand the Applications node and locate the enterprise application you want to undeploy.
3. Right-click the enterprise application node and choose **Undeploy**.

## 14.8 Packaging Enterprise Applications

Packaging lets you define the JARs, folders, libraries, and projects that are packaged in the EAR file when you build your enterprise application. You can set up a filter to exclude files from the EAR file.

Right-click any standard enterprise application project and choose **Properties** to specify packaging options for the project.

### 14.8.1 How to Redeploy a Project to a Different Server

If you are deploying to a different application server, you have to configure the server-specific deployment descriptors and the server resources.

Each project has a target server. The target server is the server that is used when the project is run. You can set the target server to any server which has been registered in the IDE.

**To change the target server:**

1. Right-click the project node in the Projects window and choose **Properties**.
2. Select **Run** in the **Project Properties** dialog box.
3. Select the new target server from the **Server** drop-down menu and click **OK**.

## 14.9 About Docker Platform Support

NetBeans IDE provides Docker platform support for Java developers. It includes registering a Docker instance, pulling and pushing Docker images, starting, stopping, and removing a container as well as editing and building a Docker file.

You must install Docker and start the Docker machine before registering a Docker instance with the IDE. Check out one of the links below related to the operating system on your machine:

- <https://docs.docker.com/linux/>
- <https://docs.docker.com/mac/>
- <https://docs.docker.com/windows/>

### 14.9.1 How to Register a Docker Instance

Before you can start packaging your applications into a Docker archive, you must register a Docker server with the IDE.

**To register a Docker server with the IDE:**

1. Open the **Services** window and right-click the **Docker** node.
2. Choose **Add Docker** from the context menu.  
The **Add Docker Instance** wizard starts.
3. In the **Add Docker Instance** dialog box, type a name for the Docker server instance, its URL, and the certificate path to the folder containing the certificates and key required to access the TLS secured Docker instance.
4. (Optional) Click **Test Connection** to check the connection.
5. Click **Finish** after you see the *Connection successful* message in the dialog box. The IDE adds a node for the Docker server instance under the **Docker** node in the **Services** window.

### 14.9.2 How to Manage Docker Images

A Docker image is a filesystem and parameters to use at runtime. For more details, see [https://docs.docker.com/mac/step\\_two/](https://docs.docker.com/mac/step_two/).

To download a particular image from a Docker registry and use it, you need to pull it from a running Docker server.

**To pull an image:**

1. Right-click the name of your Docker server instance under the **Docker** node and choose **Pull** from the context menu.
2. In the **Search Image** dialog box type the name of the image you are going to pull and click **Pull**.

The **Output** window displays the progress of pulling the image by the IDE.

To share your image, you need to push it to your custom/private repository on Docker Hub.

---

**Note:** You need to get a personal Docker Hub account and store images under a private repository. For details see <https://docs.docker.com/docker-hub/accounts/>.

---

**To push an image:**

- Right-click the name of your Docker server instance under the **Docker** node and choose **Push** from the context menu.

**To delete an image:**

- Right-click the name of your Docker server instance under the **Docker** node and choose **Remove** from the context menu.

### 14.9.3 How to Manage a Container

A Docker container is a running instance of a Docker image. For more details, see [https://docs.docker.com/mac/step\\_two/](https://docs.docker.com/mac/step_two/).

To manage a container, right-click the container name in the **Containers** folder and choose the required command in the context menu.

You can perform the following actions on a container:

- **Start.** Start the selected container.
- **Stop.** Stop the selected container.
- **Pause.** Suspend all processes in a container.
- **Unpause.** Resume all suspended processes in a container.
- **Remove.** Delete the instance which is no longer needed.
- **Show log.** Display a container log.
- **Attach.** Attach to the selected running container.

#### 14.9.4 How to Run a Docker Container

**To run a Docker container:**

1. Right-click the image name in the **Images** folder under the name of your Docker server instance and choose **Run** from the context menu.
2. In the **Container Properties** panel of the **Run** dialog box fill in the **Name**, **Command**, and **User** fields as needed. Click **Next**.
3. (Optional) In the **Port Bindings** panel of the **Run** dialog box, click the **Add Exposed** button to map exposed ports of the container to a Docker host.
4. Click **Finish** to start your container.

#### 14.9.5 How to Create a Dockerfile

You need a Dockerfile to build new images. For more details see  
<https://docs.docker.com/engine/reference/builder/>.

**To create a Dockerfile:**

1. Right-click your project name in the **Projects** window and choose **New > Dockerfile** from the context menu.
2. In the **Name and Location** of the **New Dockerfile** dialog box specify the details of your Dockerfile and click **Finish**.

The IDE creates a new Dockerfile that you can see in the **Files** window in your project files folder.

#### 14.9.6 How to Edit a Dockerfile

A Dockerfile is a text file that contains all the commands used for building Docker images automatically. For more details see  
[https://docs.docker.com/engine/userguide/eng-image/dockerfile\\_best-practices/](https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/).

To edit a Dockerfile, right-click the Dockerfile in the **Files** window and choose **Open** from the context menu. The IDE displays the Dockerfile content in the Source Editor area with syntax highlighting, auto-completion and syntax verification.

#### 14.9.7 How to Build an Image from a Dockerfile

To build a new image you need a Dockerfile that contains the instructions for building an image.

**To build an image from a Dockerfile:**

1. Right-click the image name in the **Images** folder under the name of your Docker server instance and choose **Build** from the context menu.  
Alternatively, you can right-click a Dockerfile in the **Files** window to get preconfigured values for the Dockerfile in the following step.
2. In the **Build Context** panel of the **Build Image** dialog box, specify your build context by clicking the **Browse** button and providing the path to an existing image to be used.
3. Click **Open** to return to the **Build Context** panel.
4. Specify the folder serving as a base folder for the build and, optionally, a tag in the **Repository** and the **Tag** fields respectively. Click **Next**.

---

**Note:** For more information, see the Docker documentation at  
<https://docs.docker.com/engine/reference/commandline/build/>.

---

5. In the **Build Options** panel of the **Build Image** dialog box, specify the Dockerfile location and details if necessary and click **Finish**.

A new image appears in the **images** folder when ready.

# 15

## Developing Application Clients

This chapter describes how to create standalone applications for use in Java EE enterprise environments.

This chapter contains the following sections:

- [About Developing Application Clients](#)
- [Creating Application Clients](#)

### 15.1 About Developing Application Clients

A Java EE application client is a stand-alone application that is configured to work as part of a Java EE enterprise application. Any Java application can access remote EJB methods and web services, so you do not necessarily have to create a Java EE application client to access your enterprise application. The advantage of accessing enterprise applications from enterprise application clients is that application clients have access to services and functionality running on the enterprise application server, while regular Java applications do not. For example, you can configure security roles and permissions for an enterprise application client.

### 15.2 Creating Application Clients

The following table summarizes the steps for creating an application client.

**Table 15–1 Tasks for Creating Application Clients**

Task	Action
Create an application client.	<ol style="list-style-type: none"><li>1. Choose <b>File &gt; New Project</b> (Ctrl+Shift+N).</li><li>2. Select the right template from the Enterprise Java Beans category.</li></ol>
Build an application client.	Right-click the application client project node and choose <b>Build</b> .
Verify an application client.	Right-click the application client project node and choose <b>Verify</b> .
Run an application client.	<ol style="list-style-type: none"><li>1. Start the application server.</li><li>2. Deploy the enterprise application by right-clicking its project node and choosing <b>Deploy</b>.</li><li>3. Right-click the application client project node and choose <b>Run</b>.</li></ol>
Generate a reference to an enterprise bean.	<ol style="list-style-type: none"><li>1. Open any Java class file in the application client project.</li><li>2. Right-click inside the class file and choose <b>Insert Code &gt; Call Enterprise Bean</b>.</li></ol>

**Table 15–1 (Cont.) Tasks for Creating Application Clients**

Task	Action
Edit an application client's deployment descriptors.	<ol style="list-style-type: none"> <li>1. Expand the Configuration Files node for the application client project.</li> <li>2. Double-click the deployment descriptor.</li> </ol>

### 15.2.1 How to create an enterprise application client

Follow these steps to use the IDE to create an enterprise application client.

1. Choose **File > New Project** (Ctrl+Shift+N). For more information, see [Section 6.2.1, "Standard Project Templates."](#)

---

**Note:** Before you can create an enterprise application project, you must register an application server.

---

2. From the Java EE category, select one of the following project templates:
  - **Enterprise Application Client.** Creates an empty enterprise application client in a standard project. See [Section 14.1, "About Developing Enterprise Applications."](#)
  - **Enterprise Application Client with Existing Sources.** Imports an existing enterprise application client into a standard project. The project's folder structure must adhere to the and the project must contain at least the `application-client.xml` deployment descriptor. For more information, see:  
<http://www.oracle.com/technetwork/java/index.html>
3. Follow the steps in the rest of the wizard.

---

**Note:** If you want to turn a regular Java project into a Java EE application client, you have to add at least the `application-client.xml` deployment descriptor to the `src/conf` folder, then close the project and create it again using the Enterprise Application Client with Existing Sources template as described above.

---

For related information see [Section 16.4, "Calling an Enterprise Bean."](#)

### 15.2.2 How to edit the deployment descriptors of an enterprise application client

Deployment descriptors are XML-based text files whose elements describe how to assemble and deploy a module to a specific environment. The elements also contain behavioral information about components that is not included directly in code.

An application client generally has the following deployment descriptors:

- **`application-client.xml`.** The general Java EE deployment descriptor that configures deployment settings on any Java EE compliant implementation.
- The server-specific deployment descriptor that configures deployment settings for the application server to which you are deploying. For application clients deployed to the GlassFish Server, the file is named `glassfish-application-client.xml`. The IDE provides a graphical deployment descriptor editor for `glassfish-application-client.xml` and automatically

updates the deployment descriptor as you edit your files. For other application servers, you have to write the server-specific deployment descriptors yourself.

The deployment descriptors for an application client are located in the project's `src/conf` folder. You can access the deployment descriptors from the project's Configuration Files node. For more information see [Section 14.4.1, "How to Edit an Enterprise Application's Deployment Descriptors."](#)

**To edit `application-client.xml`:**

1. In the Projects window, expand the Configuration Files node for your application client project.
2. Double-click `application-client.xml` to open it in the Source Editor. You have to edit the XML code by hand using the IDE's XML code completion and validation. There is no graphical editor for `application-client.xml`.

**To edit `glassfish-application-client.xml`:**

1. Double-click `glassfish-application-client.xml` to open it in the graphical editor. The graphical editor opens in a Source Editor tab.
2. Edit the deployment descriptor as necessary.
3. Click XML to edit the XML source for `glassfish-application-client.xml`. The Source Editor provides code completion and validation for all XML files.

**To edit the server-specific deployment descriptor for any other application server:**

- Double-click the deployment descriptor node. The IDE opens the XML source of the deployment descriptor in the Source Editor.

---

**Note:** If you enter any XML syntax errors the IDE automatically alerts you. To fully ensure your changes have not caused any errors, you should verify the application client.

---

### 15.2.3 How to add a module to an enterprise application

EJB or web application modules can be deployed either by themselves or as a part of an enterprise application. Likewise, Java application clients can be either regular Java applications, or can be full Java EE application client modules. Adding an EJB module, web module, or application client module to an enterprise application lets you further configure the way those modules interact with each other.

When you add a module to an enterprise application, the IDE does the following:

- Lists the module under the enterprise application's Java EE Modules node.
- Adds the module project to the enterprise application project's list of required projects. This means that the module project is cleaned and built whenever you clean and build the enterprise application project.
- Adds a module reference to the enterprise application's general deployment descriptors.
- Packages the module's build output (JAR file or WAR file) into the enterprise application's EAR file when building the project.

**To add an existing module to an enterprise application:**

1. Open the module project and the enterprise application project.

2. In the Projects window, right-click the enterprise application node for the Java EE application and choose **Add Java EE Module**.
3. In the dialog box, select the module and click **OK**.

You can only add standard projects to an enterprise application. You cannot add free-form projects to an enterprise application because the build and deploy process is not controlled by the IDE.

# 16

---

## Developing with Enterprise Beans

This chapter describes how to use the IDE to implement Enterprise JavaBeans (EJB) technology in applications.

This chapter contains the following sections:

- [About Developing with Enterprise Beans](#)
- [Creating an EJB Module Project](#)
- [Creating an Enterprise Bean](#)
- [Calling an Enterprise Bean](#)
- [Building and Deploying an EJB Module](#)
- [Developing EJB 2.1 Entity Beans](#)

### 16.1 About Developing with Enterprise Beans

Written in the Java programming language, an enterprise bean is a server-side component that encapsulates the business logic of an application. The business logic is the code that fulfills the purpose of the application. In an inventory control application, for example, the enterprise beans might implement the business logic in methods called `checkInventoryLevel` and `orderProduct`. By invoking these methods, remote clients can access the inventory services the application provides. See [Section 16.3, "Creating an Enterprise Bean."](#)

Enterprise beans provide the following advantages when developing large, distributed applications:

- Because the EJB container provides system-level services to enterprise beans, the bean developer can concentrate on solving business problems. The EJB container, and not the bean developer, is responsible for system-level services such as transaction management and security authorization.
- Because the beans, and not the clients, contain the application's business logic, the client developer can focus on the presentation of the client. The client developer does not have to code the routines that implement business rules or access databases. See [Section 16.4, "Calling an Enterprise Bean."](#)
- Because enterprise beans are portable components, the application assembler can build new applications from existing beans. These applications can run on any compliant Java EE-compliant server provided that they use the standard APIs.

You can develop the following types of enterprise beans:

- **Session Beans.** Perform tasks for a client and implement Web services. See [Section 16.3.1, "How to Create Enterprise Beans."](#)

- **Entity Beans.** In J2EE 1.4 applications, entity beans are used to represent a business entity object that exists in persistent storage. In Java EE applications, entities are used instead of entity beans. For more about persistent entity objects, see [Section 17.3.1, "How to Create an Entity Class."](#)
- **Message-Driven Beans.** Act as listeners for the Java Message Service API, processing messages asynchronously. See [Section 16.3.4, "How to Send JMS Messages."](#)

For additional information see [Section 16.2.1, "How to Create an EJB Module Project,"](#) and [Section 16.1, "About Developing with Enterprise Beans."](#)

## 16.2 Creating an EJB Module Project

The following steps summarize enterprise development tasks.

### 1. Create the EJB module project.

- Register a Java EE application server. You cannot deploy EJB modules to the Tomcat server.
- Create the project for the EJB module.
- Add the EJB module to an enterprise application.

### 2. Code the enterprise beans.

- Create the enterprise beans.
- Access the database.

For Java EE 5 applications, use Java Persistence.

If you want to use a database schema, make sure the database schema file exists in your project's conf folder.

- Call enterprise beans from web application modules and other enterprise beans.
- Add business methods to your enterprise beans.
- Add select methods and finder methods to your entity beans.
- Use server resources like databases and JMS messages.

### 3. Configure the application.

- Edit the EJB module's deployment descriptors.
- Configure container-managed relationships between enterprise beans.

### 4. Build the project.

- Set the project as the main project and choose **Run > Build Main Project (F11)** from the main menu or right-click the project node in the Projects window and choose **Build**.

### 5. Deploy the module.

- a. For stand-alone EJB modules, right-click the EJB module's project node and choose **Run**.
- b. For EJB modules that are part of an enterprise application, right-click the enterprise application and choose **Run**.

### 16.2.1 How to Create an EJB Module Project

The IDE provides wizards to help you create EJB module projects for EJB 3.1 enterprise beans (EJB 2.1 is a mandatory part of the EJB 3.1 specification). The wizard can also help you create an EJB module if you already have existing source code.

#### To create an EJB module project:

1. Choose **File > New Project** (Ctrl+Shift+N) from the main window.
2. From the Java EE category, select one of the following project templates:
  - **Enterprise Application.** Lets you create an EJB module project as part of an empty enterprise application. For further information see [Section 15.2.1, "How to create an enterprise application client."](#)
  - **Enterprise Application with Existing Sources.** Imports an existing enterprise application.
  - **EJB Module.** Creates an empty EJB module in a standard EJB module project. See [Section 16.2, "Creating an EJB Module Project."](#)
  - **EJB Module with Existing Sources.** Imports an existing EJB module into a standard project.
3. Follow the steps in the rest of the wizard.

For related information see [Section 6.2.3, "Creating Standard Projects,"](#) and [Section 12.2.1, "How to Create a Web Application Project."](#)

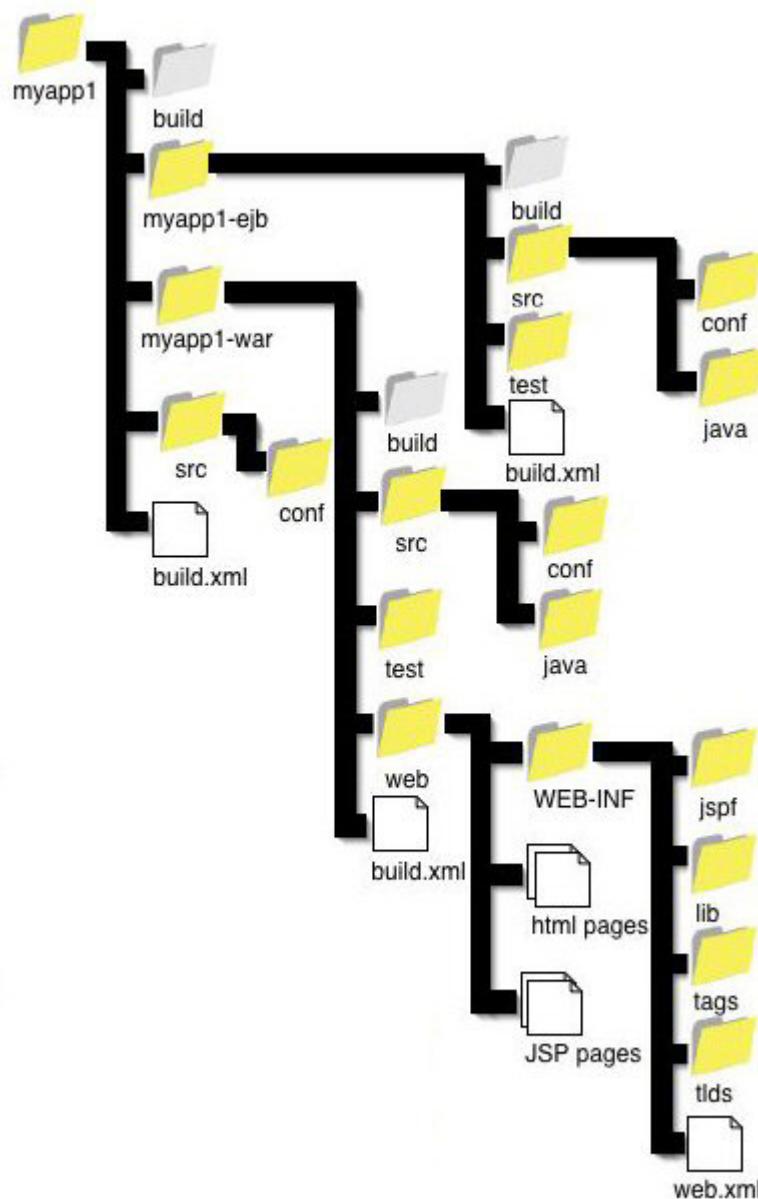
---

**Note:** For standard EJB module projects, the IDE does not support J2EE 1.3 EJB modules and automatically updates the version number of your EJB module's ejb-jar.xml to 1.4. The J2EE 1.4 specification is fully backward-compatible, so upgrading the specification level does not change the behavior of your code.

When you import an individual EJB module into a standard project, the module can have any folder structure. When importing an entire application using the Enterprise Application with Existing Sources template, the project's folder structure must adhere to the application structure shown in [Figure 16-1](#).

---

Figure 16–1 J2EE Blueprint Conventions for Application Structures



## 16.3 Creating an Enterprise Bean

The IDE provides templates for creating session beans, message-driven beans, entity beans, and legacy CMP entity beans.

### 16.3.1 How to Create Enterprise Beans

Enterprise Java Beans can be added to EJB module projects.

#### To create enterprise beans:

1. In the Projects window, select any existing EJB module project. See [Section 16.2.1, "How to Create an EJB Module Project."](#)

2. Choose **File > New** from the main menu. The New File wizard opens.
3. From the Enterprise JavaBeans category, select one of the following EJB templates:
  - Session Bean
  - Timer Session Bean
  - Message-Driven Bean
  - Session Beans for Entity Classes
4. Follow the directions in the rest of the wizard.

When you create an enterprise bean, the IDE does the following:

- **Java EE projects:**
  - Adds a node for the enterprise bean under the project's Enterprise Beans node. This node can contain nodes for bean methods and fields.
  - Creates the bean class and the interfaces under the Source Packages node
  - Opens the bean class in the Source Editor
- **J2EE 1.4 projects:**
  - Adds a node for the enterprise bean under the project's Enterprise Beans node. This node can contain nodes for bean methods and fields.
  - Updates the deployment descriptors for the EJB module and the Enterprise Application module that contains the EJB module. If your target server is the GlassFish server, the IDE also configures the server-specific deployment descriptors. For all other application servers, you have to write the server-specific deployment descriptors yourself.
  - Opens the bean class in the Source Editor

For more information on enterprise projects and beans, see [Section 12.2, "Creating Web Application Projects,"](#) and [Section 14.5.1, "How to Build an Enterprise Application Project and Sub-Projects."](#)

### 16.3.2 How to Generate Session Beans for Entity Classes

The IDE provides a wizard to generate session beans for entity classes. In the wizard choose the entity classes that require session beans. The wizard creates a session facade as well as the selected interfaces for the selected entity class.

#### To generate session beans from entity classes:

1. Right-click the module project node in the Projects window and choose **New > Other**.
2. In the New File wizard, go to the Persistence category and select **Session Beans for Entity Classes**. See [Section 17.1, "About Developing with Java Persistence."](#)
3. Select any entity classes from the list in the left pane and click the **Add** button. Any referenced entity classes are automatically added to the list in the right pane. The IDE generates session beans for each entity class listed in the right pane.

Deselect **Include Referenced Classes** if you do not want session beans created for referenced classes.

4. Click **Next**.
5. Select the location where you want to save the session beans.

6. Select an existing package from the **Package** drop-down menu or type the name of a new package.
7. Select the session facade interfaces that you want the wizard to generate.
8. Click **Finish**.

When you click **Finish**, the IDE creates session beans for each of the entity classes you specified in the wizard.

### 16.3.3 How to Define a Business Method for an Enterprise Bean

A business method is a method inside an enterprise bean that performs some function and can be called from outside the bean. Business methods are written in the bean class and exposed through local or remote interfaces. If a business method is meant to be called by beans and clients that are running in the same JVM (on the same server), they are registered in the bean's local interface. If the method is meant to be called from a remote client, it is registered in the bean's remote interface.

A session bean's business methods typically define tasks that the bean performs for clients. They are not necessarily associated with data in a database.

In EJB 2.1 programming, an entity bean represents a persistent business object whose data is stored in a database. The bean adds behavior specific to that data. Therefore, an entity bean's business methods are typically accessor and mutator methods that enable you to read and update information in a database that represents a business concept. Entity bean business methods are almost always local methods, since remote clients should only access an entity bean through a facade session bean.

#### Business Methods in the IDE

When you create an EJB 2.1 enterprise bean (J2EE 1.4), the IDE creates a special `BeanInterfaceBusiness` interface in which it registers business methods. The bean interface extends the `Business` interface and the bean class implements both the bean interface and the `RemoteBusiness` interface.

The advantage of this approach is that it lets you separate the business logic from implementation logic, and that it lets you check at compile-time that your bean implements the given interfaces. If you want to register business methods directly in the bean's interfaces, delete the `Business` interface files. Then edit the bean interface to not extend the `Business` interface and edit the bean class to not implement the `Business` interface.

When working with EJB 3.0 or EJB 3.1 enterprise beans, there is no need for a separate `Business` interface because the enterprise beans are regular Java objects and therefore have only the remote and local interfaces.

#### To define a business method for an enterprise bean:

1. Open the bean class in the Source Editor.
2. Right-click anywhere in the Source Editor and choose **Insert Code > Add Business Method**.
3. In the Add Business Method dialog box, define the method signature:
  - a. Name the method.
  - b. Specify the return type.
  - c. Specify the method parameters in the Parameters tab.
  - d. Specify the exceptions that the method throws in the Exceptions tab.

4. Specify whether to add the business method to the local interface, remote interface, or both.
5. Click **OK** to create the method.
6. The IDE adds the method signature to the interfaces and the method body to the bean class.
7. Edit your new business method.

You can also just code a regular method and then click the IDE hint in the left margin and choose an interface from the popup menu.

---

**Note:** When the IDE adds a business method to an enterprise bean, it does not save the bean class or the bean interfaces. Only when you compile or save the class is the new method saved to disk.

---

#### 16.3.4 How to Send JMS Messages

A Java Message Service (JMS) message is an object that communicates information between JMS clients. The message is sent from one client to a second client that listens for messages through a JMS destination on an application server. In the IDE, you can automatically generate code to send a JMS message to any message-driven bean in an open project.

**To send a JMS message:**

1. (Optional) Create a service locator class to handle getting the reference to the database. See [Section 16.3.5, "How to Use a Service Locator."](#)
2. Open the Java file from which you want to send the JMS message. The file must be in an EJB module project or a standard web application project. You cannot generate lookup code in a free-form web application project. See [Section 16.3.4, "How to Send JMS Messages,"](#) and [Section 23.3.9, "How to Set up a JMS Resource."](#)
3. In the Source Editor, right-click in the file and choose **Insert Code > Send JMS Message**.
4. Select the message-driven bean that is the destination for the message. You can switch between open projects using the **Project** drop-down menu.
5. Under Service Locator Strategy, specify whether to generate inline lookup code or use an existing service locator. Click **Browse** to search for the service locator class name.
6. Click **OK**. The IDE creates two methods:
  - **createJMSMessageForNewMessageDestination**. The method that creates the message that is sent. You should implement your business logic in this method.
  - **sendJMSMessageToNewMessageDestination**. The method that handles the message you created in the method above. You should not modify this message.

The IDE also registers the resource reference and message destination reference in your module's general deployment descriptor.

---

**Note:** If your project's target server is the GlassFish/Sun Java System Application Server, the IDE automatically configures a destination and connection factory for the message-driven bean when you create the bean. If you are deploying to a different application server, you have to configure the server-specific deployment descriptors and the server resources yourself.

---

### 16.3.5 How to Use a Service Locator

A service locator is a class that provides a single, reusable way to obtain a reference to the following:

- An enterprise bean's local and/or remote home interfaces
- A JMS connection factory and topic destination
- An e-mail session
- A JDBC data source
- The URL, name, or boolean value for an environment entry

You can create a regular service locator or one that caches the obtained reference for further use. Generally you use a caching service regulator in web applications and a regular service locator in the business tier. Enterprise beans especially need non-caching service locators. Because each enterprise bean has a unique JNDI name space, a resource reference with the same name can be declared in multiple enterprise beans, potentially with different types.

#### To create a service locator:

1. Choose **File > New**.
2. From the Enterprise category, select **Service Locator** or **Caching Service Locator**.

#### To use a service locator:

Run any of the following commands:

- Call Enterprise Bean, see [Section 16.4.1, "How to call an Enterprise Bean."](#)
- Use Database. For more information, see [Section 24.1, "About Working and Connecting with Databases."](#)
- Send JMS Message, see [Section 16.3.4, "How to Send JMS Messages."](#)
- Send E-Mail, see [Section 16.3.7, "How to Send an Email from a Java File."](#)

In the Service Locator Strategy section, select **Use Existing Class** and specify the class name of the service locator. Use the **Browse** button to search for the class.

---

**Note:** The service locator template is fully functional. You do not need to modify the code, although you can customize its caching behavior.

You can only create a service locator in a web application project or EJB module project.

---

For more information, see [Section 16.3.6, "How to Access a Connection Pool from a Java Class."](#)

### 16.3.6 How to Access a Connection Pool from a Java Class

Applications that are deployed to servers often need to access databases that reside on remote machines. You therefore have to set up a JDBC connection pool on the server that points to the database. You then create a data source that connects to the connection pool and use the data source's JNDI name to acquire a connection to the database. See [Section 23.3.7, "How to Set up a Connection Pool."](#)

**To access a database from an enterprise application:**

1. (Optional) Create a service locator class to handle getting the reference to the database.
2. Open the Java file from which you want to access the database. In the Source Editor, right-click the file and choose **Insert Code > Use Database**. For more information see [Section 24.1, "About Working and Connecting with Databases."](#)
3. Enter a JNDI name for the database connection. This name must be the same as the JNDI name of the JDBC data source for the database on the target server. See [Section 24.3.2, "How to Connect to the Java DB Database."](#)

If you are deploying to the GlassFish Server, you can also generate a connection pool and data source for the database connection by selecting the **Create Server Resources** checkbox. The resources are created under the project's Server Resources node.

4. Select the database to which you want to connect in the **Connection** drop-down list. If the database is not listed, do the following:
  - a. If the JDBC driver for the database server is not registered in the IDE, click **Add Driver**.
  - b. Click **Add Connection** to connect to the database.
5. Under Service Locator Strategy, specify whether to generate inline lookup code or use an existing service locator. Click **Browse** to search for the service locator class name.
6. Click **OK**.

The Use Database command is only available in EJB module and web application projects.

### 16.3.7 How to Send an Email from a Java File

1. (Optional) Create a service locator class to handle getting the reference to the database.
2. Open the Java file from which you want to send the e-mail message. The file must be in an EJB module project or a standard web application project. You cannot generate lookup code in a free-form web application project.
3. In the Source Editor, right-click the file and choose **Insert Code > Send E-mail**.
4. Specify the JNDI name of the mail resource on the application server.
5. Under Service Locator Strategy, specify whether to generate inline lookup code or use an existing service locator. Click **Browse** to search for the service locator class name.
6. Click **OK**.

The IDE creates two methods:

- **getSession**. This method looks up the mail resource and creates a mail session.
- **sendMail**. This method constructs and sends the e-mail.

The IDE also registers the resource reference in your module's general deployment descriptor.

7. In your code, call the sendMail method and pass the e-mail address, subject, and body as parameters.

---

**Note:** You have to configure the JavaMail resource on the application server yourself. If your project's target server is the GlassFish server, you can use the JavaMail Resource template found in the New File wizard GlassFish.

If your project's target server is the GlassFish server, the IDE automatically configures the resource reference in the application-specific deployment descriptor. If you are deploying to a different application server, you have to configure the server-specific deployment descriptors yourself.

---

For additional information see [Section 12.10.2, "How to Edit Deployment Descriptors,"](#) and [Section 16.5.3, "How to Edit the Deployment Descriptors of an EJB Module."](#)

## 16.4 Calling an Enterprise Bean

Calling an enterprise bean is the process of getting a reference to the enterprise bean so that you can call its methods. The IDE can assist you in calling an enterprise bean from any Java class in an EJB module, web application, or enterprise application client.

### 16.4.1 How to call an Enterprise Bean

1. Open the IDE project that contains the enterprise bean that you want to call.
2. Open the file from which you want to call the enterprise bean.
3. In the Source Editor, right-click anywhere in the body of the class and choose **Insert Code > Call Enterprise Bean**.
4. In the dialog box, select the enterprise bean that you want to call.
5. (Optional) Type a new name for the reference.
6. Specify whether to reference the local interface or the remote interface. To reference a bean's local interface, the class that is referencing the bean must be packaged in the same enterprise application as the enterprise bean.
7. Click **OK**.

When you call an enterprise bean, the following occurs:

#### Java EE projects:

- The IDE uses the @EJB annotation to indicate a dependency on another bean. In the Java class from which you are calling the enterprise bean, the IDE adds code similar to the following:

```
@EJB  
private NewSessionRemote newSessionBean;
```

- The enterprise bean project is added to the caller project's classpath.

#### J2EE 1.4 projects:

- A lookup method is created in the caller class.
- The enterprise bean project is added to the caller project's classpath.
- A reference to the entity or session bean being called is added to the deployment descriptor of the caller project. If and where the reference is added depends on which of the following applies:
  - If the lookup is made from an enterprise bean implementation class, the reference is added to the corresponding bean.
  - If the lookup is made from any class of a web application, the reference is added to the web application.
  - If the lookup is made from a plain Java class of an EJB module, no reference is added. In this case, the reference has to be added manually to the bean calling the plain Java class.

---

**Note:** The IDE cannot assist you when calling an enterprise bean from a free-form web application project.

---

## 16.5 Building and Deploying an EJB Module

There are two ways to build an EJB module:

- **As part of a Java EE application.** The IDE packages all of the necessary JAR files as part of the Java EE application and not as part of the individual modules. This is the most common way of building an EJB module.
- **As a stand-alone EJB module.** The IDE packages the necessary JAR files with the EJB module JAR file.

When an EJB module is part of an enterprise application, its enterprise beans can be accessed from external clients and web application modules through its remote interfaces, and by other modules in the enterprise application through its local interfaces. All of the JAR files needed for execution are packaged in the enterprise application's EAR file and the classpath to these JAR files are written in the EJB module's manifest file. This is the most common way of deploying an EJB module.

There are two deployment commands:

- **Deploy.** This stops the enterprise application, undeploys the application or stand-alone module from the application server, and deploys your local version.
- **Run.** Like Deploy, except that for enterprise applications, this command also opens the client module's default URL in the external browser.

### 16.5.1 How to Build an EJB Module as Part of an Enterprise Application

The IDE packages all of the necessary JAR files as part of the Java EE application and not as part of the individual modules. This is the most common way of building an EJB module.

1. In the Projects window, expand the enterprise application's Java EE Modules node and make sure that the EJB module is listed. If you have not added the EJB module to the enterprise application, right-click the Java EE Modules node, choose **Add Java EE Module**, and select the EJB module.

2. Right-click the enterprise application project and choose **Build**. Or, if the enterprise application project is the main project, choose **Run > Build Main Project** (F11) from the main menu.

---

**Note:** To set a project as the main project right-click the project in the Projects window and choose **Set as Main Project**, or, choose **Run > Set Main Project** from the menu bar and select the project in the sub-menu.

---

When you build an EJB module as part of an enterprise application, the IDE does the following:

- Compiles the classes to the project's `build/jar-module` folder.
- Places any JAR files, libraries, or directories that are on your project's classpath and are scheduled for inclusion in deployment in the enterprise project's build folder.
- Builds the EJB module's JAR file to the EJB module project's `dist` folder and copies it to the enterprise application project's build folder. The EJB module JAR file does not contain any of the classpath elements.
- Puts a `Class-path` entry in the EJB module's manifest file for all included JAR files and puts a `.` entry if there is at least one directory on the project's classpath scheduled for inclusion in deployment. This mechanism ensures that the JAR files and directories can be used from the EJB module even if they reside in the EAR project and not in the EJB module itself.
- Builds the enterprise application EAR file to the enterprise application project's `dist` folder.

### 16.5.2 How to Build a Stand-alone EJB Module

The IDE packages the necessary JAR files with the EJB module JAR file.

#### To build a stand-alone EJB module:

Right-click the EJB module project in the Projects window and choose **Build**. Or, select the EJB module project in the Projects window and choose **Run > Build Project (Build Main Project)** (F11) if the EJB module project is set as the main project).

When you build a EJB module project as a stand-alone module, the IDE does the following:

- Compiles the classes to the project's `build/jar` folder.
- Places any JAR files, libraries, or directories that are on your project's classpath and are scheduled for inclusion in deployment into the `build/jar` folder.
- Builds the EJB module's JAR file to the EJB module's `dist` folder. The JAR file includes all classpath items scheduled for inclusion in deployment.

---

**Note:** To mark a classpath item for inclusion in deployment in the EJB module's Project Properties dialog box, right-click the module's Libraries node and choose **Properties**, then select the checkbox for each classpath item you want to include in deployment.

---

### 16.5.3 How to Edit the Deployment Descriptors of an EJB Module

If you are developing enterprise beans on the Java EE 5 platform and deploying to a Java EE 5 compliant application server, you develop enterprise beans using EJB 3.0 technology.

If you are developing enterprise beans on the Java EE 6 or Java EE 7 platform and deploying to a Java EE-compliant application server, you develop enterprise beans using EJB 3.1 technology. EJB 3.0 and EJB 3.1 technology support the use of annotations in your code to describe behavior that was described using deployment descriptors in EJB 2.1 technology. As a result, deployment descriptors files are no longer required when deploying to a Java EE-compliant server. EJB 3.0 and EJB 3.1 technology also support the use of deployment descriptors. You may need to use a combination of annotations and deployment descriptors in your project when you configure your project, for example to configure security settings.

For EJB modules, there are two types of deployment descriptors:

- The general Java EE deployment descriptor that configures deployment settings on any Java EE-compliant implementation. The general EJB deployment descriptor is called `ejb-jar.xml`. The IDE provides a visual editor for the general deployment descriptor and automatically updates the descriptor file when you edit your code.
- The server-specific deployment descriptor that configures deployment settings for the application server to which you are deploying. For the Sun Java System Application Server, the IDE automatically updates the deployment descriptor files as you edit your files. For all other application servers, you have to write the server-specific deployment descriptors yourself.

The IDE provides a visual deployment descriptor editor to help you edit some descriptor files. You can also use the XML editor to edit your descriptor files.

#### To edit `ejb-jar.xml`:

1. In the Projects window, expand the Configuration Files node for your EJB module project.
2. Double-click `ejb-jar.xml` to open the file in the visual editor.
3. Select one of the following tabs:
  - **General.** This tab enables you to specify general information about the EJB module, details about security roles and descriptor details about each of the enterprise beans in the module.
  - **CMP Relationships.** This tab enables you to view and modify CMP relationships for CMP beans in your module.
  - **XML.** This tab enables you to edit the descriptor file in the XML editor.
4. Edit the properties.
5. Save your changes.

#### To edit `glassfish-ejb-jar.xml`:

1. In the Projects window, expand the Configuration Files node for your EJB module project.
2. Double-click `glassfish-ejb-jar.xml` to open the file in the visual editor.
3. Select a tab at the top of the editor.

4. Edit the descriptor file as necessary and save your changes.

**To edit the server-specific deployment descriptor for any other application server:**

1. Double-click the deployment descriptor node in the Projects window.
2. The IDE opens the XML source of the deployment descriptor in the Source Editor.
3. If you enter any XML syntax errors the IDE automatically alerts you. To fully ensure your changes have not cause any errors, you should verify the EJB module.
4. In the Projects window, right-click the enterprise application project and choose **Run or Deploy**.

For additional information see [Section 23.4.1, "How to Register a Server Instance."](#)

**To deploy an enterprise application:**

If the module is part of an enterprise application, right-click the enterprise application project and choose **Debug**.

If the module is a stand-alone EJB module, right-click the EJB module project and choose **Debug**.

#### 16.5.4 How to Verify the Deployment Descriptors of an EJB Module

Before you deploy an enterprise application or any stand-alone web or EJB module to the application server, you should verify it, to ensure it is properly implemented to the Java EE specification. When the target server for the project is the Sun Java System Application Server or GlassFish application server, use the verifier tool to validate both Java EE and application server-specific deployment descriptors against their corresponding DTD files and display errors and warnings if a module or application is not compliant. You can verify deployment descriptors in EAR, WAR, RAR, and JAR files.

The verifier tool is not simply an XML syntax verifier. Rules and interdependencies between various elements in the deployment descriptors are verified. Where needed, classes in your module are examined to ensure that what is referred to in the deployment descriptors actually exists and works when deployed.

**To verify a project deployment descriptors:**

1. Right-click any web application module, EJB module, or enterprise application project in the Projects window and choose **Verify**.
2. The Verifier window opens and displays all the results.
3. Filter the results according to your needs:
  - Click **Failures Only** to display only failures.
  - Click **Failures and Warnings Only** to display only failures and warnings.
4. Make appropriate changes, if any, and verify the project again.

---

**Note:** Not all failures to comply with the Java EE specification cause the application to fail. However, modules that fail to comply with the specification might not be portable between application servers. The verifier tool helps you ensure your code is not locked into running only on a specific application server.

---

### 16.5.5 How to Deploy a Stand-alone EJB Module

Enterprise beans in a stand-alone EJB modules can only be accessed through their remote interfaces. All of the JAR files needed for execution are packaged in the EJB JAR file itself.

#### To deploy a stand-alone EJB module:

In the Projects window, right-click the EJB module project and choose **Run** or **Deploy**.

### 16.5.6 How to Debug an EJB Module

There are two ways to debug an EJB module:

- As a stand-alone EJB module
- As part of an enterprise application

#### To debug a stand-alone EJB module:

- In the Projects window, right-click the EJB module project and choose **Debug**.

When you debug a stand-alone EJB module, the IDE does the following:

- Compiles the EJB module if necessary.
- Stops the application server and starts it in debug mode.
- Deploys the EJB module to the application server.
- Starts a debugging session, attaches the debugger to the server, and opens the debugger windows at the bottom of the IDE screen. Since an EJB module does not have an executable class, there are no local variables or calls on the call stack. To debug the functionality in the EJB module, use an application client or web application to access the EJB module's methods.

#### To debug an enterprise application:

In the Projects window, right-click the enterprise application project and choose **Debug**.

When you debug an enterprise application, the IDE does the following:

- Compiles the EAR file if necessary.
- Stops the application server and starts it in debug mode.
- Deploys the enterprise application to the application server.
- Starts a debugging session, attaches the debugger to the server, and opens the debugger windows at the bottom of the IDE screen. If the application has a designated web application and URL to run, the IDE opens the URL in the external browser.

---

**Note:**

If your EJB module is part of an enterprise application, you should always debug it by running the debug command on the enterprise application project. Since the IDE does not know which enterprise applications an EJB module project belongs to, running the Debug command on an EJB module project deploys it as a stand-alone module.

When debugging an enterprise application, if your application contains any projects not listed in the Packaging page of the enterprise application, you must explicitly specify these projects if you want the IDE to include them in the classpath when you debug the application. To specify the projects to include, right-click the enterprise application project node in the Projects window and choose **Properties**. In the Properties dialog box, select the **Libraries** category and add the projects to the Embedded Classpath Elements list.

The debugger windows filter out any information from method calls running on the server.

For additional information see [Section 23.3.1, "How to Register a Server Instance."](#)

---

### 16.5.7 How to Profile an EJB Module

There are two ways to profile an EJB module:

- As a stand-alone EJB module
- As part of an enterprise application

If your EJB module is part of an enterprise application, you should running the profile command on the enterprise application project. Since the IDE does not know which enterprise applications an EJB module project belongs to, running the Profile command on an EJB module project deploys it as a stand-alone module.

If you are deploying your EJB module or enterprise application to a local installation of the GlassFish server, go to the main menu and choose **Profile > Profile Project** and choose the project in the sub-menu, or, by right-click the project node in the Projects window and choosing **Profile** from the pop-up menu.

If you are deploying your EJB module or enterprise application to a remote server, you must attach the IDE to the remote server and profile the application in Attach mode. The Attach Wizard can help you configure the remote server to accept attachment.

**To profile a stand-alone EJB module:**

1. In the Projects window, right-click the EJB module project and choose **Profile**.
2. Select a profiling task and click **Run**.

When you profile a stand-alone EJB module, the IDE does the following:

- Compiles the EJB module if necessary.
- Stops the application server and starts it in profile mode.
- Deploys the EJB module to the application server.
- Starts a profiling session, attaches the profiler to the server, and opens the Profiler window in the IDE. After the application is deployed to the server, you should

invoke its code the same way as you would for debugging or just evaluating application functionality. You can use an application client or a web application to access the EJB and invoke its code. You can view the profiling data once the application code is invoked.

**To profile an enterprise application:**

- In the Projects window, right-click the enterprise application project and choose **Profile**.
- Select a profiling task and click **Run**.

---

**Note:** If you are analyzing application performance, choose the **Profile Projects & Subprojects Classes** filter to ensure that the classes in your EJB module and web module projects are profiled. If you do not choose this filter, the classes in the EJB module and web module are not profiled.

---

Follow this procedure to limit profiling the classes in the enterprise application:

1. In the Analyze Performance pane, choose the **Profile Projects & Subprojects Classes** filter and then click **Show Filter Value**.
2. Click **To Quickfilter** in the dialog box.
3. In the Set Quick Filter dialog, select a filter type, edit the filter values to limit the classes that are profiled, and click **OK**.
4. Make sure that **Quick Filter** is selected in the **Filter** drop-down list when you run the profiling session.

When you profile an enterprise application, the IDE does the following:

- Compiles the EAR file if necessary.
- Stops the application server and starts it in profile mode.
- Deploys the enterprise application to the application server.
- Starts a profiling session, attaches the profiler to the server, and opens the Profiler window in the IDE. If the application has a designated web application and URL to run, the IDE opens the URL in the external browser.

For more on profiling see [Chapter 9, "Testing and Profiling Java Application Projects."](#)

### 16.5.8 How to Test an EJB Module

To use local JUnit tests to test an EJB module that is deployed on a server, you must configure the tests to act as a remote client of the EJB module. This means you can only access the EJB module through its remote interfaces. For information on JUnit unit testing, see [Section 9.3, "Creating a Unit Test."](#)

Usually, you have some entity beans and a session bean with remote interfaces that provides clients access to the entity beans. You have to generate tests for the session bean's bean class and modify the test file to reference the session bean and test each of its business methods.

**To generate tests for an EJB module:**

1. Create a service locator file somewhere in the test package source root.

2. Open the enterprise bean's bean class in the Source Editor and choose **Tools > JUnit Tests > Create Tests** (Ctrl+Alt+J). Accept the default options in the dialog box and click **OK**.
3. In the test class, delete the test methods that just test EJB infrastructure methods such as `testEjbCreate` and `testEjbRemove`.
4. Declare a variable for the remote home interface.
5. In the `setUp` method, write code to instantiate this variable, as in the following example:

**Example 16–1 Instantiate a Variable for the Remote Home Interface**

```
protected void setUp() throws NamingException,
    CreateException, RemoteException {
    ServiceLocator sl = new ServiceLocator();
    newSessionHome = (NewSessionRemoteHome)sl.getRemoteHome("ejb/NewSessionBean",
        NewSessionRemoteHome.class);
}
```

6. In each test method, retrieve a remote interface and test the business method, as in the following example:

**Example 16–2 Retrieve and Test a Remote Interface**

```
try {
    NewSessionRemote newSession = newSessionHome.create();
    assertEquals("name", newSession.getName(new Integer(1)));
}
catch (Exception e) {
    fail(e.toString());
}
```

---

**Note:** You do not have to use a service locator class. You can write the lookup code from scratch.

---

## 16.6 Developing EJB 2.1 Entity Beans

EJB 2.1 entity beans are part of the EJB 2.1 specification and are used in J2EE 1.4 enterprise applications. NetBeans IDE 8.0 no longer supports developing applications that use J2EE 1.4 technology. To develop J2EE 1.4 applications you should use NetBeans IDE 7.3 or earlier versions of the IDE.

---

# Developing with Java Persistence

This chapter describes the features the NetBeans IDE provides to support the Java Persistence API.

This chapter contains the following sections:

- [About Developing with Java Persistence](#)
- [Creating a Persistence Unit](#)
- [Creating an Entity Class](#)
- [Generating JPA Controller Classes](#)
- [Adding Support for Java Persistence](#)

## 17.1 About Developing with Java Persistence

The Java Persistence API handles how relational data is mapped to persistent entity objects, how these objects are stored in a relational database, and how an entity's state is persisted. The Java Persistence API is defined as part of the Java EE specifications, but can also be used in Java SE environments.

### Java Persistence Features

The following features are part of the Java Persistence API:

- You can use an entity class to create a persistent entity object in Java EE applications. See [Section 17.3, "Creating an Entity Class."](#)
- You can map persistent entity objects to a relational database using Java annotations or XML descriptors. See [Section 17.3.2, "How to Map Entity Classes."](#)
- You can express queries in the native query language of the database or Java Persistence query language, which is an extension of the EJB query language.
- You can package entity classes in JAR, WAR, EJB JAR, and EAR archives.
- You can use an entity manager to perform Create Read Update Delete (CRUD) operations that involve entities. [Section 17.3.4, "How to Obtain an Entity Manager."](#)
- You can specify the persistence provider you want to use in your application. [Section 17.2.2, "Persistence Provider."](#)
- You can use Java Persistence API technology in Java SE applications. See [Section 17.5.1, "How to Add Support for Java Persistence to the Project."](#)

For more on using Java Persistence in Java SE applications and for deploying application to non-Java containers, see [Section 17.5.1, "How to Add Support for Java](#)

### Persistence to the Project."

For more about the features of the Java Persistence API, see Chapter 16: Introduction to the Java Persistence API in the Java EE 6 Tutorial.

<http://docs.oracle.com/javaee/6/tutorial/doc/>

### Support for XML Descriptors

Although you do not need to specify additional XML descriptors, you have the option of using them as an alternative to annotations or to supplement or override some annotations. Using an XML descriptor might be useful in externalizing object-relational mapping information. Also, multiple XML descriptors can be useful in tailoring object-relational mapping information to different databases.

For related information, see [Section 14.1, "About Developing Enterprise Applications."](#)

## 17.2 Creating a Persistence Unit

A persistence unit is required if you are using Java Persistence in your application. A persistence unit is a uniquely-named collection of properties that are used to determine how a specific set of entities in an application are managed and persisted.

Persistence units are defined in the `persistence.xml` file. You can have more than one persistence unit defined in `persistence.xml`, but each persistence unit must have a unique name.

Properties specified in a persistence unit include the following:

- The entity classes within the scope of the persistence unit
- The persistence provider or library used for managing the set of entities scoped by that persistence unit
- The data source used for the persistent storage of the managed entities
- The transaction type the application uses

Persistence units can be packaged as part of a WAR or EJB-JAR file, or can be packaged as a JAR file that can then be included in your application.

### 17.2.1 Scope of the Persistence Unit

The scope of a persistence unit is determined by the location of `persistence.xml`. When you use a wizard to create a persistence unit, the IDE creates `persistence.xml` in the location appropriate for the scope. For example, if you use the IDE to create a persistence unit for an EJB module, the IDE generates `persistence.xml` in the `src/conf` directory of your EJB module. When the EJB JAR is built, it packages `persistence.xml` in the EJB JAR's `META-INF` directory. The scope of the persistence unit is the set of classes in the EJB JAR file.

If the scope is not explicitly specified in a persistence unit, by default all the entities in the EJB JAR file would be within the scope of the persistence unit defined in `persistence.xml`.

---

**Note:** The location of `persistence.xml` determines the persistence root. The root of the persistence unit is the JAR file or directory that contains the `META-INF` directory containing `persistence.xml`.

---

## 17.2.2 Persistence Provider

A persistence provider refers to an implementation of the Java Persistence API. The persistence provider is a library that provides the functionality to persist objects in the application.

The IDE is bundled with the EclipseLink persistence provider. EclipseLink is the reference implementation and the default Java Persistence provider in the GlassFish application server. You can use EclipseLink as your provider, or specify a different persistence provider.

## 17.2.3 Data Source

A data source refers to the database where persistent entities are stored. The data source must be registered on the server and is specified using the JNDI name. If the transactions are container-managed JTA transactions, the data source must be a JTA data source. If the transactions are application-managed, the data source is specified according to the JDBC database connection registered with the IDE.

In Java SE environments, the database is specified either using a data source or by other means, depending on the requirements of the persistence provider used.

## 17.2.4 Transaction Types

A persistence unit specifies how transactions are managed in the application. The transaction type you can use depends on the target container. If you are deploying to a Java EE container, transactions can be container-managed or application-managed. If you are not deploying to a Java EE container, transactions must be managed by the application.

- Container-managed transactions (JTA transactions). Container-managed transactions are handled by the container using the Java Transaction API (JTA). To use the Java Transaction API, you must deploy your application to a Java EE container and your data source needs to support JTA transactions.

In `persistence.xml`, the transaction type for the persistence unit is set to JTA. If you are deploying to the GlassFish application server, this option is selected by default when you create the persistence unit.

- Application-managed transactions (resource-local transactions). Application-managed transactions are handled by the application.

In `persistence.xml`, the transaction type for the persistence unit is set to `RESOURCE_LOCAL`.

For related information, see [Section 16.1, "About Developing with Enterprise Beans."](#)

## 17.2.5 How to Create a Persistence Unit

A persistence unit is required if you are using Java Persistence in your project.

1. In the Projects window, right-click the project node and choose **New > Other**.
2. Select **Persistence Unit** in the Persistence category and click **Next**.
3. Specify a unique Persistence Unit Name. In most cases you can keep the default name suggested by the IDE.
4. Select a persistence provider **Persistence Provider** or library from the dropdown list, or add a new library by choosing New Persistence Library.

5. Select a data source (see "Scope of the Persistence Unit") from the dropdown list. The data source can be a JDBC connection or a database connection. To appear in the list, the data source needs to be registered with the IDE.
6. Select **Use Java Transaction APIs** if you want the container to manage the entities.

---

**Note:** Java Transaction APIs are only available if you are deploying to a Java EE container. If you are not deploying to a Java EE container, the transaction needs to be managed by the application. For more, see [Section 17.2.4, "Transaction Types."](#)

---

7. Specify a table generation strategy for your database.

8. Click **Finish**.

When you click **Finish**, the file `persistence.xml` opens in the Source Editor.

In EJB and WAR projects, you can find `persistence.xml` in the Projects window under the Configuration Files node. In the Files window, `persistence.xml` is located in the `src/conf` directory.

When packaged, `persistence.xml` is located in the META-INF directory of an EJB JAR archive or the WEB-INF/classes directory of a WAR archive.

## 17.3 Creating an Entity Class

In Java EE applications, you use entity classes to create persistent entity objects ("entities"). Entity classes are "plain old Java objects" (POJOs). Entity classes import the Java persistence library `javax.persistence.Entity` and are marked with the `@Entity` annotation in your source code.

### About Entity Classes

Entities have the following characteristics:

- Each entity class usually represents a table in a relational database.
- Each instance of an entity corresponds to a row in a table.
- Persistent fields or properties correspond to columns in a table.

Entity classes are not restricted to EJB modules in enterprise applications. Entity classes can be located in an EJB module or a web module and can be also used in Java SE applications.

When coding entity classes, you use annotations to map entities and entity relationships to a database. You do not need to use external XML descriptor files to map persistent objects to a database. The information about the data source is contained in a persistence unit (see [Section 17.2, "Creating a Persistence Unit"](#)).

For additional information see [Section 14.1, "About Developing Enterprise Applications."](#)

### 17.3.1 How to Create an Entity Class

1. Right-click the module project node in the Projects window and choose **New > Other**.
2. In the New File wizard, select **Entity Class** from the Persistence category.
3. Enter the name of the class.

4. Select the location where you want to save the entity class.
5. Select an existing package from the **Package** drop-down list or type the name of a new package.
6. Set the type of the variable you want to use as the primary key. The default type for the primary key is Long.
7. Click **Finish**.

When you click **Finish**, the IDE generates the entity class and opens the class in the Source Editor. Annotations in the entity class define the primary key and the primary key generation strategy for the entity.

### 17.3.2 How to Map Entity Classes

An entity class is used to represent a table in a database, and the fields in an entity class correspond to columns in that table. In an entity class, you can use annotations to specify how fields in an entity class are mapped to the corresponding database columns and tables.

For example, the following `@Column` annotation marking the field `address` maps the field to the column named `CUSTOMER_ADDRESS` in the database table.

```
@Column(name = "CUSTOMER_ADDRESS")
private String address;
```

When mapping a persistent field or property name to a column, you do not need to specify a `@Column` annotation for the field or property if the name of the mapped database column is the same as the field or property name because they are mapped by default.

[Table 17-1](#) displays the annotations that are commonly used when mapping entity classes.

**Table 17-1 Entity Class Mapping Annotations**

Annotation	Description
<code>@Id</code>	Specifies the primary key property or field of an entity.
<code>@GeneratedValue</code>	Allows you to specify the strategy that automatically generates the values of primary keys. Used with <code>@Id</code> .
<code>@Column</code>	Specifies a mapped column for a persistent property or field.
<code>@ManyToMany</code>	Defines a many-valued association with many-to-many multiplicity.
<code>@ManyToOne</code>	Defines a single-valued association to another entity class that has many-to-one multiplicity.
<code>@OneToMany</code>	Defines a many-valued association with one-to-many multiplicity.

For more on using annotations and annotation elements to map entities in an enterprise application, see the Java EE 6 Tutorial:

<http://docs.oracle.com/javaee/6/tutorial/doc/>

For more on the specifications on annotations and annotation elements, see the Java EE 6 API specifications for `javax.persistence`:

<http://docs.oracle.com/javaee/6/api/javax/persistence/package-summary.html>

For additional information see [Section 15.1, "About Developing Application Clients."](#)

### 17.3.3 How to Generate Entity Classes from a Database

In addition to writing entity classes from scratch, you can also generate a set of persistent entity classes or mapped superclasses for an existing database. You can use the New Entity Classes from Database wizard to generate entity classes or mapped superclasses from a connected database or from a database schema.

#### To generate entity classes or mapped superclasses from a database:

1. Right-click the module project node in the Projects window and choose **New > Other**.
2. In the New File wizard, select **Entity Classes from Database** from the Persistence category.
3. Select the source database that contains the tables that you want to use to generate the classes:
  - **Data Source.** Choose a data source from the dropdown list. Alternately, you can choose **Add Data Source** to create a new data source. The database must be running to choose this option.

---

**Note:** When choosing a data source, the server must be running and the data source must be registered with the server.

---

---

**Note:** If your target server is not a Java EE container, the dropdown list contains the database connections registered with the IDE.

---

- **Database Schema.** Choose a database schema from the dropdown list. This option is available only if there is a database schema in your project's `src/conf` folder.

After you select the source database, the tables in that database are listed in the Available Tables pane.

4. Select any tables in the left pane and click the **Add** button. Any tables related to the tables you select are automatically added to the list in the right pane. The IDE will generate classes for each table listed in the right pane.

---

**Note:** Deselect **Include Related Tables** if you do not want entity classes created for related tables

---

5. Click **Next**.
6. Confirm the name of the classes that will be generated for each table listed.
7. Select the location where you want to save the entity classes.
8. Select an existing package from the **Package** drop-down menu or type the name of a new package.
9. Confirm the annotations that you want the IDE to generate in the classes.

If you want the IDE to generate mapped superclasses instead of entity classes, select **Generate Mapped Superclasses instead of Entities**. If you select this option

the **Generate Named Query Annotations for Persistent Fields** is automatically deselected.

**10.** Click **Next**.

**11.** (Optional) Specify any mapping options.

**12.** Click **Finish**.

When you click **Finish**, the IDE creates the classes for each of the tables you specified in the wizard. The package containing the generated classes is selected in the Projects window.

---

**Note:** When you select a data source or JDBC connection, the IDE also creates a database schema for the database and saves the schema in your `src/conf` folder.

---



---

**Note:** To persist entity classes, your project requires a persistence unit.

---

For additional information see [Section 14.1, "About Developing Enterprise Applications,"](#) and [Chapter 24, "Working and Connecting with Databases."](#)

#### 17.3.4 How to Obtain an Entity Manager

An entity manager is associated with a group of managed entities called a "persistence context". Entities within a persistence context are associated with an entity manager instance. To use an entity manager, you first need to obtain an entity manager from the container.

**To obtain an entity manager:**

1. Open the class where you want to add the entity manager resource.
2. Right-click in the Source Editor and choose **Persistence > Use Entity Manager** from the popup menu.

The IDE generates the code to obtain an entity manager instance.

How the entity manager instance is obtained depends upon your project. For example, if you are using a container-managed entity manager, you can obtain an entity manager instance by injecting the entity manager resource directly using the `@PersistenceContext` annotation. If entities in your project are application-managed, you first need to obtain an `EntityManagerFactory` before you can obtain an entity manager. You can obtain an `EntityManagerFactory` by using the `@PersistenceUnit` annotation in a Java EE container, or by calling `Persistence.createEntityManagerFactory` in a Java SE environment.

For more on entity managers, see the Java EE 6 Tutorial:

<http://docs.oracle.com/javaee/6/tutorial/doc/>

## 17.4 Generating JPA Controller Classes

A JPA controller class is a wrapper for an entity class that provides clients with access to the database through the methods in the entity class. The JPA controller class contains the logic for creating, editing and destroying an entry in the data source,

getting all of the entries in the data source, and getting a specific entry in the data source.

You can use the JPA Controller Classes from Entity Classes wizard to generate JPA controllers based on entity classes in your application and exception classes that are used by the controller classes. The wizard generates one JPA controller class for each entity class that you select and places the controller class in the specified location. Each generated JPA controller class contains `create`, `edit` and `destroy` methods and methods for retrieving the entities and uses an entity manager for managing entity persistence.

### 17.4.1 How to Generate a JPA Controller Class from an Entity Class

**To generate a JPA controller class from an entity class:**

1. Choose **File > New (Ctrl+N)** from the main menu.
2. From the Persistence category, select **JPA Controller Classes from Entity Classes** and click **Next**. The wizard displays all of the entity classes in the project.
3. Add all of the entity classes for which you want to generate controller classes to the Selected Entity Classes list and click **Next**.
4. Specify a package for the JPA controller classes.
5. Click **Finish**.

After the wizard generates the JPA controller classes, the controller methods can be invoked from JSP pages or JSP converters or other classes in your application. If the database schema changes you can use the IDE wizards to again generate new entity classes and controller classes and then where necessary update the code that invokes the controller methods.

## 17.5 Adding Support for Java Persistence

Support for Java Persistence is available to Java EE and Java SE applications running on the Java EE platform. If you have a J2EE 1.4 application it is possible to use Java Persistence if the target container is running on the Java EE platform. The container does not need to be a Java EE container.

---

**Note:** You cannot use a container-managed entity manager in a J2EE 1.4 application.

---

If your target server is not a Java EE container, you can add support for Java Persistence functionality by adding the EclipseLink library to the project or the classpath of the container. The EclipseLink library contains the necessary libraries to support Java Persistence. The EclipseLink library is bundled with the IDE.

### 17.5.1 How to Add Support for Java Persistence to the Project

Follow these steps to add support for Java Persistence.

1. In the Projects window, expand your project node, right-click the Libraries node and choose **Add Library**.
2. In the Add Library dialog box, select **EclipseLink** and click **Add Library**.

When you add the EclipseLink library to your project, the EclipseLink library will be packaged with the EAR or WAR file when you build the project.



---

## Developing Applications Using XML

This chapter describes how to create well-formed XML documents using the IDE.

This chapter contains the following sections:

- [About Developing Applications Using XML](#)
- [Creating and Editing XML Documents](#)
- [Generating a DTD from an Existing XML File](#)
- [Creating an Empty DTD](#)
- [Generating Documentation for a DTD](#)
- [Registering a Local DTD or XML Schema](#)
- [Removing a DTD or XML Schema Entry from the User Catalog](#)

### 18.1 About Developing Applications Using XML

You can use the IDE to create XML documents.

**To create an XML document using the IDE:**

1. Create a new XML, CSS, DTD, or XSL stylesheet document.
2. Use the Source Editor to open and edit the XML document.
3. Check the XML document.
  - Check that the document is well formed.
  - Validate documents constrained by an XML Schema or DTD.

### 18.2 Creating and Editing XML Documents

The Extensible Markup Language (XML) is a subset of SGML designed to enable generic SGML to be served, received, and processed on the Web easily and efficiently. Because its format is not fixed, XML enables you to design your own customized markup languages with which to create various types of documents.

In the IDE, XML documents are represented by XML nodes (  ). The IDE provides tools to assist you in creating, editing, checking, and validating the various XML document types it supports.

The IDE supports several types of XML documents, including:

- **Cascading Style Sheet.** A CSS (  ) provides a simple mechanism for formatting and adding style to HTML and XML documents.

- **Document Type Definition.** A DTD (  ) describes the grammar that can be used in an XML file and indicates the valid arrangement of the file's tags. It can exist as a prologue to an XML file (internal DTD) or as a separate file (external DTD).

- **Parsed Entity Objects.** Parsed entity nodes represent parsed entity (.ent) files. If several of your XML documents refer to a single piece of information, you can store that information in a parsed entity. Any changes you make to the parsed entity are then automatically reflected in all of the documents that reference it.

For more information about using parsed entities in XML documents, see:

<http://www.w3.org/TR/REC-xml/#TextEntities>

- **XML Catalogs.** XML catalogs (  ) provide mapping information that maps an external entity in an XML document to the actual location of the document being referenced. You can use a catalog to redirect mappings for an external entity, such as a DTD or XML file, to a different location.
- **XSL Stylesheet.** XSL stylesheets (  ) define transformation rules for your XML documents. You can perform an XML transformation to transform the data in an XML document into the output of your choice, such as formatted HTML or a text file.

You can use the New File wizard to create regular well-formed documents with no DTD or schema constraints, DTD-constrained documents, and XML schema-constrained documents.

### 18.2.1 How to Create a Well-formed XML Document

You can use the IDE to create an XML document whose syntax conforms to the XML specification.

#### To create a well-formed XML document:

1. From the main window, choose **File > New File**.
2. In the New File wizard, select the XML node in the Categories pane and the XML Document node in the right pane. Then click **Next**.
3. Specify a name and location for the document and click **Next**.
4. Select **Well-formed Document** and click **Finish**.

The IDE creates the new XML document in the specified location

### 18.2.2 How to Create a DTD-constrained XML Document

You can use the IDE to create a well-formed XML document whose structure is determined by a document type definition (DTD).

#### To create a DTD-constrained XML document:

1. From the main window, choose **File > New File**.
2. In the New File wizard, select the XML node in the left pane and the XML Document node in the right pane. Then click **Next**.
3. Specify a name and location for the document and click **Next**.
4. Select **DTD-Constrained Document** and click **Next**.
5. Type the DTD Public ID or select it from the combo box list of all public IDs for your mounted XML catalogs. Once you have specified the DTD public ID, press Enter to automatically load the DTD information in the wizard.

6. Specify the DTD system ID for your document.
7. Specify the document root for your document. The Document Root combo box contains all of the declared entities in your DTD.
8. Click **Finish**.

The IDE creates the new XML document in the specified location.

### 18.2.3 How to Create an XML Schema-constrained Document

You can use the IDE to create a well-formed XML document whose structure is determined by an XML schema.

**To create an XML schema-constrained document:**

1. From the main window, choose **File > New File**.
2. In the New File wizard, select the XML node in the left pane and the XML Document node in the right pane. Then click **Next**.
3. Specify a name and location for the document and click **Next**.
4. Select **XML Schema-Constrained Document** and click **Next**.
5. In the Schema URI field, type or browse to the schema location. Once you have selected a schema, the wizard suggests values for the remaining fields in the pane.
6. Specify the namespace for your document in the Document Namespace combo box. The wizard automatically enters the schema's target Namespace in this field.
7. Type or select the document root for your document in the Root Element combo box. The combo box contains all of the schema's exposed elements.
8. Click **Finish**.

The IDE creates the new XML document in the specified location.

### 18.2.4 How to Edit an XML Document

The IDE's Source Editor provides the following features for editing XML documents:

- **Syntax coloring.** Colored highlighting of code elements.
- **Indentation and reformatting.** The IDE indents lines as you type. You can format selected lines or the entire file by pressing Alt+Shift+F.
- **Code completion.** Basic code completion, such as auto-completion of element end tags, is available for all XML documents. If your document is defined by a DTD and the IDE can access the DTD, the Source Editor also offers code completion for all supported elements, attributes, and entities. As you type, a combo box displays all matching elements in the DTD. Press Enter to enter only the element name, or press Shift+Enter to enter a start tag and end tag for the element.
- **Navigator window.** The Navigator window (Ctrl+7) displays the structure of your XML documents. The Navigator window displays the name, attributes, and text content of each element. You can double-click any element node to jump to its location in the Source Editor.
- **Code folding.** You can expand and collapse any element in the Source Editor.

**To open an XML file in the Source Editor:**

- Double-click the XML file's node.

## 18.2.5 How to Validate an XML Document

If you have an XML document that is constrained by an XML schema or DTD, you can validate the XML document to check its contents for syntax errors.

If your XML document's DTD is located on a remote machine, ensure that the IDE's proxy is correctly configured and you have access to the DTD. For more information see [Section 19.5.2, "How to Set a Proxy for Web Services and Clients."](#)

---

**Note:** If the IDE cannot access the DTD, it returns a single error message listing the DOCTYPE declaration. If the IDE cannot access the XML schema, it displays Failed to read schema document.

---

### To validate an XML document:

1. Do one of the following:
  - Right-click the document's node in the Files or Projects window.
  - If the file is open in the Editor, right-click anywhere in the file.
2. Choose **Validate XML** (or if the file is open in the Editor, press **Validate XML**).

The Output window lists any errors along with the line number where the error was detected.

---

**Note:** This location is not necessarily the line where the error occurred. You can double-click any error message to go to the line in the XML file where the error was detected.

---

### Common Error Messages

Some common error messages generated during DTD validation include:

- End of entity not allowed; an end tag is missing. There is no final end tag for the document.
- Expected "</tagname>" to terminate element started on line line#. There is no end tag for the specific element.
- Element type "element name" must be declared. The XML document contains an element type that was not declared in the DTD.

## 18.2.6 How to Check that an XML Document is Well-formed

XML documents, including DTD, CSS, and XSL files, are considered well-formed if they adhere to a basic set of grammatical rules. The IDE enables you to check an XML document to ensure it conforms to the necessary rules, including the following requirements:

- Every start tag must have a matching end tag. Tags are case-sensitive.
- Start, end and empty-element tags must have proper nesting, without missing or overlapping tags.
- There must be exactly one root element, also known as the document element, that does not appear in the content of any other element.
- Attribute values must be quoted. Attributes are listed in the start tag.
- An element cannot have two or more attributes with the same name.

- Comments and processing instructions cannot appear inside tags.
- No unescaped < or & signs can occur in the element's character data or attribute's character data, except when used as markup.

**To check an XML document:**

1. Right-click the document's node in the Files window.
2. Choose **Check XML**, **Check DTD**, or **Check CSS**, depending on the document you are checking.
3. The Output window lists any errors along with the line number where the error was detected.

---

**Note:** This location is not necessarily the line where the error occurred. You can double-click any error message to go to the line in the XML file where the error was detected.

---

## 18.3 Generating a DTD from an Existing XML File

You can generate a DTD from an existing XML file.

1. In the Files window, right-click an XML file's node and choose **Generate DTD**. Make sure that you have previously checked the XML file to ensure that it is well-formed.
2. Type a name for the new DTD and click **OK**.

The new DTD node appears in the Files window and is displayed in the Source Editor.

## 18.4 Creating an Empty DTD

A document type definition (DTD) describes the grammar that can be used in an XML file and indicates the valid arrangement of the file's tags. It can exist as a prologue to an XML file (internal DTD) or as a separate file (external DTD). This section describes how to create your own empty DTD. You can also generate a DTD from an existing file. See "["Generating a DTD from an Existing XML File."](#)"

**To create an empty DTD:**

1. From the main window, choose **File > New File**.
2. Expand the XML node and select DTD Entity. Click **Next** to proceed.
3. Type a name and package for the DTD.
4. Click **Finish** to create the new file.

In the Files window, the new DTD node appears under the package you specified. The DTD is also displayed in the Source Editor.

## 18.5 Generating Documentation for a DTD

A document type definition (DTD) describes the grammar that can be used in an XML file and indicates the valid arrangement of the file's tags. It can exist as a prologue to an XML file (internal DTD) or as a separate file (external DTD). You can create a new

DTD yourself, or you can automatically generate a DTD from an existing XML document.

## 18.6 Registering a Local DTD or XML Schema

The DTDs and XML Schemas Manager displays all of the currently registered DTD and XML schemas. To open the DTDs and XML Schemas Manager, choose **Tools > DTDs and XML Schemas** from the main menu.

### To register a local DTD or XML schema:

1. Choose **Tools > DTDs and XML Schemas** from the main menu.
2. Select the User Catalog node and click **Add Local DTD or Schema**.
3. Select which ID to map to the local file:
  - **Public ID.** Generally you use this option to redirect references to a DTD.
  - **System ID.** Generally you use this option to redirect references to an XML schema.
4. Browse to the location of the local DTD or XML schema in the URI field.

The appropriate User Catalog entry node () appears under the User Catalog node. This entry represents the public ID or system ID mapping to the local DTD or XML schema resource.

## 18.7 Removing a DTD or XML Schema Entry from the User Catalog

Follow these steps to unregister a DTD or XML Schema.

1. Choose **Tools > DTDs and XML Schemas** from the main menu.
2. Expand the User Catalog node.
3. Select the XML catalog node you want to remove and click **Remove**.

---

## Developing and Securing Web Services

This chapter describes how to develop and secure JAX-WS and RESTful web services.

This chapter contains the following sections:

- [About Developing and Securing Web Services](#)
- [Working with Web Services](#)
- [Creating Web Services](#)
- [Configuring Web Services](#)
- [Creating JAX-WS Web Service Clients](#)
- [Creating RESTful Web Service Clients](#)
- [Deploying and Testing Web Services and Clients](#)
- [Creating Handlers](#)
- [Using JAXB for Java-XML Binding](#)
- [Configuring Quality of Service](#)
- [Securing an Operation](#)

### 19.1 About Developing and Securing Web Services

Web services are distributed application components that conform to standards that make them externally available. They solve the problem of integrating diverse computer applications that have been developed independently and run on a variety of software and hardware platforms.

The promise of web services architecture is to allow you to connect applications that were developed on different platforms and in different programming languages. This can only work if vendors can agree on common standards.

The following web service programming models are supported by the IDE:

- **SOAP Web Services (JAX-WS).** SOAP is an XML protocol that can be used for messages to and from a web service. Java API for XML Web Services (JAX-WS) is the specification for SOAP web services that is supported by NetBeans IDE. For more details, see [Section 19.3.1, "How to Create SOAP \(JAX-WS\) Web Services."](#)
- **REpresentational State Transfer (REST).** Central to REST is the concept of resources identified by universal resource identifiers (URIs). These resources can be manipulated using a standard interface, such as HTTP, and information is exchanged using representations of these resources. For more details, see [Section 19.3.2, "How to Create RESTful Web Services."](#)

## 19.2 Working with Web Services

This section describes how to create and use a web service.

### To create a web service:

Follow these steps to create a web service:

1. Create the web service.

- a. Make implementation decisions for your web service.
- b. Choose one of the following:

**Create a JAX-WS web service from Java.** When creating a web service from Java, the assumption is that you do not have a WSDL file. However, you can use existing code to implement your web service. See [Section 19.3.1, "How to Create SOAP \(JAX-WS\) Web Services."](#)

**Create a JAX-WS web service from a WSDL File.**

**Create a RESTful web service.** RESTful web services are particularly useful if you need to read and write to a database. See [Section , "Create a RESTful Web Service From Patterns;"](#)

2. Develop a web service.

- a. In the Projects window or Files window, double-click the files that you would like to edit.
- b. Use the Source Editor to develop the web service. Regarding JAX-WS web services, you can use the IDE to generate skeleton code for one or more of the following:
  - Adding a web service operation. See [Section 19.3.1.1, "How to Add Operations to a JAX-WS Web Service."](#)
  - Adding a handler. See [Section 19.8.1, "How to Create a Handler."](#)
  - Calling an EJB. See [Section 16.4.1, "How to call an Enterprise Bean."](#)

3. Build the web service.

In most cases you do not need to build the web service. The IDE builds the service automatically as part of the deployment process. However,

- a. Right-click the project's node in the Projects window and choose **Build**.
- b. Analyze the result and, if necessary, customize the related tool's features. For a JAX-WS web service that is created from a WSDL file, you can use the Web Service Attributes editor to customize XML to Java (JAXB) mappings, WS Security, or other WS\* features.

4. Deploy the web service.

To deploy the web service, simply deploy the web application that contains the service. If you have a web service that delegates to an EJB module, you can combine the service's web application and the EJB module in an enterprise application and deploy the enterprise application. See [Section 12.1, "About Developing Web Applications,"](#) and [Section 14.7, "Deploying Enterprise Applications."](#)

5. Test the web service.

Test the web service or test JAX-WS web services or test RESTful web services.

## 19.3 Creating Web Services

You can create SOAP (JAX-WS) or RESTful Web Services. See [Section 19.3.1, "How to Create SOAP \(JAX-WS\) Web Services,"](#) and [Section 19.3.2, "How to Create RESTful Web Services."](#)

### 19.3.1 How to Create SOAP (JAX-WS) Web Services

JAX-WS is a technology for building web services and clients that communicate using XML. JAX-WS allows developers to write both message-oriented and RPC-oriented web services. JAX-WS simplifies web services compared to the older JAX-RPC standard by using annotations, a Java EE 5 innovation.

A JAX-WS web service consists of an implementation class, which is a Java class that implements the service endpoint. The implementation class defines the service endpoint interface implicitly, by default.

The implementation class is annotated with `@WebService` or `@WebServiceProvider`. The web service's operations are defined in methods in the implementation class. These methods are annotated with `@WebMethod`.

In Java EE 6 and Java EE 7 the JAX-WS web service can be implemented as a stateless session bean. To use an EJB with a web service in EE 5, you have to create a separate EJB module and add that module to the web application's classpath. You can combine the web application and the EJB module in an Enterprise Application.

In JAX-WS, messages to and from the web service use the XML-based protocol known as "SOAP". The SOAP specification defines the envelope structure, encoding rules, and conventions for representing web service invocations and responses. These calls and responses are transmitted as SOAP messages (XML files) over HTTP.

JAX-WS depends on the following specifications:

- **SOAP (Simple Object Access Protocol).** Defines the mechanism by which a web service is called and how data is returned. For detailed information, refer to the SOAP 1.2 Specification.
- **WSDL (Web Services Description Language).** Describes the external interface of a web service. For detailed information, refer to the WSDL 1.1 Specification.
- **UDDI (Universal Discovery, Description, and Integration).** Registries contain information about web services, including the location of WSDL files and the location of the running services. The IDE does not let you publish web services to or browse from a UDDI registry, although the WSDL files that you use can come from a variety of sources, including a UDDI registry.
- **JAXB (Java Architecture for XML Binding).** JAX-WS delegates data binding related tasks to JAXB. The JAXB 2.0/2.1 specification is developed in parallel with JAX-WS 2.0/2.1.

For more information about JAX-WS web services, see the Java EE5 Tutorial.

#### How to Create an Empty JAX-WS Web Service

1. Depending on the implementation form and EE version, create a web application project or create an EJB module project.
2. In the Projects window or Files window, right-click the project node and choose **New > Web Service**.

The New Web Service wizard opens.

3. Type the web service name and specify a package to house the web service.

4. Select the creation type:
  - **Create an Empty Web Service.** Use this when you have no existing code to expose.
  - **Create Web Service From Existing Session Bean.** Use this when you have a session bean in an EJB module. Click **Browse** to select the session bean to which you want to delegate.
5. Click **Finish**.

The IDE creates an implementation class. The implementation class is created with the required web service annotations and import statements.
6. Use the Source Editor to develop the web service.

**To create a JAX-WS web service from a Java class:**

1. Open a Java class.
2. Type `@WebService` above the class declaration.
3. Press Alt+Enter in the line, or click the light bulb in the left sidebar, and select `javax.jws.WebService`.
4. Press Alt+Enter in the class declaration, or click the light bulb in the left sidebar, and click Add new operation.

The Add Operation dialog box appears.
5. Add an operation and click **OK** to exit the Add Operation dialog box.
6. In the Projects window, expand the Web Services node and click on the new node, named after your Java class. For example, if your Java class is called `Test.java`, the new node in the Web Services node is called `TestService`. The web service opens in the Web Service Designer, where you can continue fine tuning your web service, visually.

**To create a web service from a WSDL file:**

When you and your business partners agree on a "contract" in the form of a WSDL file, you can use the IDE to implement it. The WSDL file is an agreement on the data and messages that will be exchanged as well as how these messages will be sent and received. In the IDE, you can use the WSDL file to implement the web service.

Business requirements may demand that you create a platform-independent description of a web service as a set of XML schema files and WSDL files. Based on this platform-independent description, you can use the steps that follow to generate the implementation files. The WSDL file that you use in the steps below can either be available on disk or via a URL.

1. Depending on the implementation form, create a web application project or an EJB module project.
2. In the Projects window or Files window, right-click the project node and choose **New > Other**. In the Web Services folder, choose **Web Service from WSDL**.

The New Web Service from WSDL wizard opens.
3. Type the web service name and specify a package to house the web service.
4. Browse to a WSDL file in your filesystem or type the URL to a WSDL file.
5. Select **Use Provider** if you want to bypass the XML <-> Java binding layer and have the service use raw XML when processing requests. Instead of XML <->

binding, the Provider interface is used. This is an advanced feature and is unselected by default.

**6. Click Finish.**

The IDE runs the `wsimport` tool, which reads the WSDL file and generates all the required artifacts for web service development, deployment, and invocation. Finally, the IDE generates the skeleton implementation class, which corresponds to the WSDL port selected in the wizard.

#### 19.3.1.1 How to Add Operations to a JAX-WS Web Service

After you create a web service, you implement its operations in the implementation class. You can add operations manually in the Source Editor, by using the Web Service Designer, or by using the Add Operation dialog box. In the latter case, the IDE automatically declares the operation in the Service Endpoint Interface (the "interface"), if there is one (it is optional in the JAX-WS programming model) and adds a skeleton method in the implementation class.

---

**Note:** You can also add operations to a web service created from a WSDL file. In this case, you can specify the XML schema types as parameters and return types.

---

##### To add a web service operation:

1. Open the Add Operation dialog box in one of three ways:
  - In the Projects window, expand the Web Services node, right-click the web service instance node, and choose **Add Operation**.
  - Open the web service's implementation class in the Source Editor. Right-click anywhere in the body of the class and choose **Web Service > Add Operation**.
  - In the implementation's Design view, click the **Add Operation** button.
2. Define the name, return type, parameters, and exceptions of the web service operation. For example, do the following if you want to create this operation:

```
public int add(int a, int b) throws MyException {
    return a + b;
}
```

- a. Type `add` in the Name text box and choose **int** from the **Return Type** drop-down.
- b. Click **Add**.
- c. Choose **int** from the **Type** drop-down and type `a` in the Name text box. Click **OK**.
- d. Click **Add**.
- e. Choose **int** from the **Type** drop-down and type `b` in the Name text box. Click **OK**.
- f. Click **OK** to create the operation.

The IDE adds the skeleton of the operation to the implementation class and declares it in the interface.

- g. In the Source Editor, implement the web service operation. For example, you could add `return a + b;` between the braces. Do this in the implementation class, not in the interface (if you have a separate interface class).

#### 19.3.1.2 How to Use the JAX-WS Web Service Designer

The Web Service Designer is the primary workspace within which SOAP web service design takes place in the IDE. It unifies the two ways in which you can develop web services—from Java classes and from WSDL files. Tabs are provided for switching from the source view to the design view. The Web Service Designer enables you to lay out web services by visually adding operations and enabling features such as reliable message delivery. In particular, you can do the following in the design view:

- Add web service operations
- Delete web service operations
- Change the web service operation name
- Optimize binary data transfer (MTOM)
- Set up the reliable messaging delivery
- Set up the web service security
- View what the SOAP requests and responses will look like on the wire

To open the Web Service Designer, create a SOAP web service, expand the Web Services node in the Projects window, and double-click on the node representing the web service.

#### 19.3.2 How to Create RESTful Web Services

For a full tutorial that shows how to work with RESTful web services, see: Getting Started with RESTful Web Services in NetBeans IDE.

You can use NetBeans IDE to create a RESTful web service in a Java Web application. You can use the IDE to create either an "empty" service based on a pattern (singleton or item/container) or a service configured to communicate with an existing database. In addition, you can expose any suitable Java class as a RESTful web service by appending it appropriately.

##### Create a RESTful Web Service From Patterns:

1. Create a web application project.
2. In the Projects window or Files window, right-click the web application project node and choose **New**.
3. Then choose one of the following patterns:
  - **Simple Root Resource.** Creates a RESTful root resource class with GET and PUT methods. This pattern is useful for creating a simple HelloWorld service and wrapper services for invoking WSDL-based web services.
  - **Container-Item.** Creates a pair of RESTful resource classes, consisting of an item resource class and its container resource class. Item resources can be created and added to the container resource using the POST method on the container resource class.

---

**Note:** The URI for the newly created item resource is determined by the container resource.

---

- **Client-Controlled Container-Item.** Creates a pair of RESTful resource classes, consisting of an item resource class and its container resource class. This pattern is a slight variation of the Container-Item pattern. The difference is that there is no POST method on the container resource class for creating item resources. Instead, item resources are created using the PUT method on the item resource class. The reason this is called Client-Controlled Container-Item pattern is because the URI for the item resource is determined by the client and not the container resource.

Alternatively, you can choose the above templates by right-clicking the project node, choosing **New > Other** and then selecting them from the Web Services category.

4. Click **Next** and complete the wizard.

#### **Create a RESTful Web Service From a Database or From Entity Classes:**

For a full tutorial that shows how to work with RESTful web services, see: Getting Started with RESTful Web Services in NetBeans IDE.

You can use the IDE to create either an "empty" service based on a pattern (singleton or item/container) or a service configured to communicate with an existing database. In addition, you can expose any suitable Java class as a RESTful web service by appending it appropriately.

1. Create a web application project.
2. In the Projects window or Files window, right-click the web application project node and choose **New**.
3. Choose one of the following:
  - **RESTful Web Service from Database.** This wizard creates entity classes from a database and then creates RESTful web services from those entity classes.

---

**Note:** If you do not have existing entity classes, it is safer and more convenient to use this wizard instead of the Entity Classes From Database wizard followed by the RESTful Web Services from Entity Classes wizard.

---

4. Click **Next** and complete the wizard.

## 19.4 Configuring Web Services

You configure your web services at several different levels.

### **Project Contents and Classpath**

Basic project settings like the web application or EJB module's source roots and classpath are set in the module's Project Properties dialog box. You open this dialog box by right-clicking the project node and choosing **Properties**.

For standard projects, you can add source roots in the Sources page of the Project Properties dialog box. You can quickly add to the compilation and test classpath by right-clicking the Libraries or Test Libraries node in the Projects window. To further

configure the classpath, or to specify which items should be included in deployment, use the Libraries page of the Project Properties dialog box.

### JAX-WS Service Compilation Settings

For JAX-WS web services, you can either use the Web Service Attributes editor or generate WSDL and Schema files and edit them manually.

To use the Web Service Attributes editor, go to the Projects window, find the project's Web Services node, right-click the web service and select **Edit Web Service Attributes**.

To generate WSDL and Schema files, go to the Projects window, find the project's Web Services node, right-click the web service and choose **Generate and Copy WSDL**. A wizard opens in which you select the location for the WSDL and schema files. After you generate the schema and WSDL, you can edit them manually. After editing these files, you need to add a wsdlLocation attribute to the service's @WebService annotation. The wsdlLocation attribute's value is the path to the generated WSDL.

If you copy and edit the WSDL and schema files, you must add the wsdlLocation to the web service. The IDE generates a WSDL file at build time, and if the wsdlLocation is not specified, the IDE uses the WSDL it generates itself instead of the one you generated.

### RESTful Service Compilation Settings

The main decision you make when you create a RESTful service is how to register that service. When you create the service, the IDE opens a dialog where you choose whether to use the default web.xml deployment descriptor created by the IDE or to write the registration yourself. For EE services, you also have the choice of using an IDE-generated subclass of javax.ws.rs.core.Application. After you create the service, you can change how the service is registered by going to the Projects window, right-clicking the application's RESTful Web Services node, and choosing REST Resources Configuration.

You can also generate and edit a WADL for a web application. To generate the WADL, select **New File > Web Services > WADL Document**. To help you edit the WADL document, install the WADL Designer plugin.

### Deployment Settings

There are several key steps to configuring deployment settings:

- **Setting the target server.** You can set the target server instance for a project by doing any of the following:
  - Specifying the server in the New Project wizard when creating the project.
  - Going to a project's Project Properties dialog box and setting the target server in the Run page.
  - Adding the module that contains the web service to an enterprise application. The module is then deployed to the same server instance as the enterprise application.
- **Adding the project module to a Java EE Enterprise Application.** You can add a web application or EJB module to an enterprise project by doing either of the following:
  - Specifying the enterprise application in the New Project wizard when creating the project.
  - Right-clicking the Java EE Modules node for any enterprise application project and choosing Add Java EE Module.

- **Configuring web services via annotations.** The IDE generates annotation snippets in the artifacts that it creates and lets you use code completion to fill out the annotation snippets.
- **Configuring web services via deployment descriptors.** You can manually configure your deployment descriptors by opening the project's `web.xml` deployment descriptor. In the case of EE6 RESTful services, which by default do not have deployment descriptors, you can generate a deployment descriptor by adding **New > Web > Standard deployment descriptor (web.xml)**.

Changes made to the `web.xml` deployment descriptor override configuration settings in annotations.

### 19.4.1 Using Annotations and Deployment Descriptors

Annotations play a critical role in JAX-WS web services. Annotations are used in mapping Java to WSDL and schema files. They are used at runtime to control how the JAX-WS runtime processes and responds to web service invocations. In the IDE, you can use code completion when working with annotations.

Deployment descriptors are XML-based text files whose elements describe how to assemble and deploy a module to a specific environment. The elements also contain behavioral information about components not included directly in code.

#### About Annotations

At the time of writing, the annotations utilized by JAX-WS are defined in separate JSRs:

[JSR 181: Web Services Metadata for the Java Platform](#)

[JSR 222: Java Architecture for XML Binding \(JAXB\) 2.0](#)

[JSR 224: Java API for XML Web Services \(JAX-WS\) 2.0](#)

[JSR 250: Common Annotations for the Java Platform](#)

For details, see [Java APIs for XML Web Services Annotations](#).

#### About Deployment Descriptors

Deployment descriptors specify two kinds of information:

- Structural information describes the different components of the JAR (or EAR) file, their relationship with each other, and their external dependencies. Environment entries and resource requirements are part of the structural information.
- Assembly information describes how contents of a JAR (or EAR) file can be composed into a deployable unit.

There are different types of deployment descriptors: EJB deployment descriptors described in the Enterprise JavaBeans specification, web deployment descriptors described in the Servlet specification, and application and application client deployment descriptors described in the Java EE specification. For each type of module or application, there are two deployment descriptors:

- A general deployment descriptor that configures deployment settings on any Java EE-compliant implementation. The general deployment descriptor is named `moduleType.xml` (for example, `ejb-jar.xml` for EJB modules and `web.xml` for web application modules.)
- A server-specific deployment descriptor that configures deployment settings for a specific server implementation. For example, the deployment descriptors for the

GlassFish Server are named `glassfish-moduleType.xml`. The deployment descriptor for the Tomcat Web Server is named `context.xml`.

### Deployment Descriptors in the IDE

The IDE generates a deployment descriptor from the settings for EJB modules, web application modules, and enterprise applications, when you create an empty enterprise project. Deployment descriptors are NOT created automatically for EE6 projects. See "[Deployment Descriptors in EE 6 Projects](#)." It also reads your existing deployment descriptors when you import existing enterprise beans, enterprise applications, and web applications into the IDE. The deployment descriptors are located in the `conf` folder of your project directory, and are displayed under the Configuration Files node in the Projects window.

As you work with your projects, the IDE updates the general deployment descriptors to reflect changes you have made, such as adding business methods to an enterprise bean or adding a web application module to an enterprise application.

The IDE also automatically updates your server-specific deployment descriptors if the target server is the Tomcat Web Server or the GlassFish Server. For all other application servers, you have to write the deployment descriptors yourself.

You can open a graphical editor for a deployment descriptor by double-clicking its node in the Projects window. You can edit the deployment descriptor's XML by right-clicking its node and choosing Edit.

The IDE also automatically registers the DTDs and schemas for all the Java EE deployment descriptors, as well as the Oracle Java System Application Server DTDs. You can therefore check and validate the XML in your deployment descriptors according to the DTDs and schemas. You can view the DTDs by choosing **Tools > DTDs and XML Schemas** from the main menu.

### Deployment Descriptors in EE 6 Projects

By default, Java EE applications use annotations instead of deployment descriptors. However, in some cases a deployment descriptor is still necessary, such as when customizing the application. In these situations you can have the NetBeans IDE create a deployment descriptor. Right-click the project's node in the Project Manager and select **New > Standard Deployment Descriptor**.

For more information see [Section 18.1, "About Developing Applications Using XML."](#)

## 19.4.2 How to Configure a Web Service With Annotations

Annotations serve two purposes. Firstly, they affect the WSDL file that will be generated for the service. Secondly, they provide deployments and runtime information to the container.

For example, you can use the `@WebService` annotation to specify the WSDL file's target namespace. You can also use the `@SOAPBinding` annotation to specify the style/use configuration of the service messages, for example, `document/literal` or `rpc/literal`.

Before there were annotations, web services were deployed with a standard deployment descriptor, accompanied by service-specific descriptors. This can now be replaced by annotations. For example, the location of the WSDL file in the deployed archive can now be specified through the `wsdlLocation` attribute of the `@WebService` annotation. Likewise, message handlers can be configured using annotations, instead of including them in the deployment descriptor.

### 19.4.3 How to Configure a Web Service With Its Deployment Descriptor

For JAX-WS web services, you usually use annotations instead of deployment descriptors, except when you want to override annotations by means of deployment descriptors.

Deployment descriptors are XML-based text files whose elements describe how to assemble and deploy a module to a specific environment. The elements also contain behavioral information about components not included directly in code.

For web services, the `webservices.xml` file configures web services, but you should seldom have a need for editing this file.

---

**Note:** In most cases, there is no need to manually edit the deployment descriptors. The IDE's wizards normally take care of configuring deployment descriptors correctly. Normally, only if your deployment descriptor becomes corrupted for some reason, does it make sense to manually edit it.

---

#### To edit `webservices.xml`:

1. In the Projects window, expand the Configuration Files node for your project.
2. Double-click `webservices.xml` to open it in the Source Editor.
3. Edit the deployment descriptor as necessary.
4. Choose **File > Save** to save your changes.

---

**Note:** If you introduce any XML syntax errors, the IDE automatically alerts you. To ensure your changes have not cause any errors, verify the web application.

---

#### To edit `filename-config.xml`:

1. In the Projects window, expand the Source Packages node for your project.
2. Double-click `filename-config.xml` to open it in the Source Editor.
3. Edit the configuration file as necessary.
4. Choose **File > Save** to save your changes.

---

**Note:** If you enter any XML syntax errors the IDE automatically alerts you. To ensure your changes have not cause any errors, verify the web application.

---

## 19.5 Creating JAX-WS Web Service Clients

A web service client is created to consume (that is, use) a specific web service. The way in which a web service client consumes a web service depends on the way in which the provider makes the web service available:

- The provider publishes the WSDL file of a running web service.
- The provider distributes a WSDL file, which is available to you on your local filesystem.

- The provider distributes a NetBeans project that defines the web service, for deployment to a test container.

For detailed information, refer to the full specifications:

- The JSR-109 Specification at <http://jcp.org/en/jsr/detail?id=109>
- The Metro Web Services Specification at Project Metro:  
<http://www.oracle.com/technetwork/java/index-jsp-137004.html>

### 19.5.1 How to Work with JAX-WS Web Service Clients

These steps summarize the JAX-WS client development process.

#### To create a JAX-WS web service client:

1. Create the web service client.
  - a. From the Projects window or Files window, right-click a node in the project and choose **New > Other**. The New File wizard appears. Under **Categories**, select **Web Services**. Under **File Types**, select **Web Service Client**. Click **Next**.
  - b. Create the web service client proxies/stubs.
- If you have problems consuming a web service, see the steps for configuring your proxy settings: [Section 19.5.2, "How to Set a Proxy for Web Services and Clients."](#)
2. Develop the web service client application
  - a. In the Projects window or Files window, double-click the files that you would like to edit.
  - b. Use the Source Editor to develop the web service client.
  - c. When you right-click in the Source Editor, you can use the IDE to generate skeleton code for calling a web service operation.
3. Build the web service client
4. Deploy the web service client

### 19.5.2 How to Set a Proxy for Web Services and Clients

If you want to access a remote web service, but your system is behind a firewall or you use a proxy server, you need to configure the IDE with your proxy settings. For working with remote web services, setting a proxy can be important in one or more of the following areas:

- When retrieving the WSDL file in the Web Service Client wizard.
- When using the IDE to test the web service.
- When deploying to the Oracle GlassFish Open Source Edition Server.
- When deploying to the Oracle WebLogic Server.
- When deploying to the JBoss Server.
- When deploying to the Tomcat Web Server.
- When deploying a web service client in a Java application.

The proxy settings for each of the situations above are described below.

## Setting a Proxy When Retrieving WSDL Files in the Web Service Client Wizard

Typically, an error such as the following is displayed in the Web Service Client wizard when the proxy settings for retrieving a WSDL file have not been set correctly:

```
Download failed. I/O exception: (Check the proxy settings.)
```

Do the following to check and set the proxy:

- Click Proxy Settings in the Web Service Client wizard.
- In the HTTP Proxy Settings window, set the proxy host and port number.

The changes take effect when you click **OK**.

## Setting a Proxy When Testing a Web Service

Typically, an error such as the following is returned when the proxy settings for testing a web service from the IDE have not been set correctly:

```
org.netbeans.modules.websvc.registry.ui.ReflectionHelper.callMethodWithParams(ReflectionHelper.java:449)
```

Do the following to check and set the proxy:

1. Choose **Tools > Options**.
2. In the Options window, set the proxy host and port number.

The changes take effect when you exit the Options window.

## Setting a Proxy on the GlassFish Server

Typically, an error such as the following is returned when the proxy settings for a web service or web service client deployed to the GlassFish Server have not been set correctly:

```
java.rmi.RemoteException: HTTP transport error:  
java.net.UnknownHostException:
```

Do the following to check and set the proxy:

1. Open the Services window (Ctrl+5).
2. If the GlassFish Server is not started, start it.
3. Right-click the GlassFish Server node and select **View Admin Console**. The Administration Console opens in a browser.
4. In the left side tree menu, go to **Configurations > server-config > JVM settings**. The JVM General Settings page opens.
5. Click **Add JVM Option**. An empty field appears in the top of the list of JVM options.
6. Type in the following property:  
`-Dhttp.proxyHost=your.proxy.host`
7. Click **Add JVM Option** again and type in the following property:  
`-Dhttp.proxyPort=your.proxy.port.number`
8. Click **Save**.

Stop and restart the server for the new proxy settings to take effect.

### Setting a Proxy on the WebLogic Server

Typically, an error such as the following is returned when the proxy settings for a web service or web service client deployed to the WebLogic Server have not been set correctly:

```
java.rmi.RemoteException: HTTP transport error:  
java.net.UnknownHostException:
```

Do the following to check and set the proxy:

1. Open the Services window (Ctrl+5).
2. If the WebLogic Server is not started, start it.
3. Right-click the WebLogic Server node and select **View Admin Console**. The Administration Console opens in a browser.
4. Log in. The main page of the Admin Console opens.
5. Go to **Environment > Servers**. A table of servers appears.
6. In the table of servers, click the name of the server for which you want to set a proxy. The Settings page for that server appears.
7. Select the **Configuration** tab and the **Server Start** sub-tab. A page opens for configuring Node Manager startup settings.
8. In the Arguments field, type or paste in the following two arguments, separated by a space:  
`-Dhttp.proxyHost=your.proxy.host  
-Dhttp.proxyPort=your.proxy.port.number`
9. Click **Save**.

### Setting a Proxy on the JBoss Server

Typically, *nothing* is displayed in the browser when the proxy settings for a web service or web service client deployed to JBoss have not been set correctly.

Do the following to check and set the proxy:

1. In your filesystem, go to `jboss_install_dir\bin\run.bat`.
2. Add this line: `set JAVA_OPTS=-Dhttp.proxyHost=your.proxy.host  
-Dhttp.proxyPort=your.proxy.port.number`

Stop and then restart the server for the new proxy settings to take effect.

### Setting a Proxy on the Tomcat Web Server

Typically, nothing is displayed in the browser when the proxy settings for a web service or web service client deployed to JBoss have not been set correctly.

Do the following to check and set the proxy:

1. Open the Services window (Ctrl+5).
2. If the Tomcat Web Server is started, stop it.
3. Right-click the Tomcat node and choose **Properties**.
4. In the Platform tab, add the following properties in the VM Options text box:  
`-Dhttp.proxyHost=your.proxy.host -Dhttp.proxyPort=your.proxy.port`

Start the server for the new proxy settings to take effect.

### Setting a Proxy When Deploying a Web Service Client in a Java Application

Typically, an error such as the following is returned when the proxy settings for a web service or web service client deployed from a Java application have not been set correctly:

```
java.rmi.RemoteException: HTTP transport error:  
java.net.UnknownHostException:
```

Do the following to check and set the proxy:

1. Right-click the Java application project node in the Projects window and choose **Properties**.
2. In the Project Properties dialog box, click **Run**.
3. Add the following properties to the VM Options field:

```
-Dhttp.proxyHost=your.proxy.host -Dhttp.proxyPort=your.proxy.port
```

The changes take effect when you click **OK**.

For client deployment from Java applications, you must set the proxy for each project, because each Java application runs as a stand-alone JVM process and each can provide different JVM parameters. For web applications sharing the same instance of a server, you need set the proxy only once -- on the server itself, as described above.

### 19.5.3 How to Generate a JAX-WS Web Service Client

A web service can be consumed in a web application, a Java application, or a MIDP client (MIDlet). For information on consuming web services in a MIDlet, see the Java ME Mobility documentation.

For JAX-WS clients, all imported WSDL files and schemas are resolved automatically by the IDE. For JAX-RPC clients, before you begin, be aware that if the WSDL file that you want to use imports schemas, other WSDL files, or both, from the local file system, the web service client will only work if the imported schemas and WSDL files have already been copied into the WEB-INF/wsdl (or META-INF/wsdl) folder. If you do not do this, the wscompile tool will not be able to locate these imported files at the end of the procedure below.

#### To create a web service client:

1. Create the project to contain the web service client. Depending on how you want to consume the web service, create a web application project or a Java application project.
2. From the Projects window or Files window, right-click a node in the project and choose **New > Other**. The New File wizard appears. Under **Categories**, select **Web Services**. Under **File Types**, select **Web Service Client**. Click **Next**.
3. Access the WSDL file of the web service that the web service client is to consume. Depending on what the provider has distributed, do the following:
  - To generate a client from a project on your local filesystem, click **Project** and browse to a project's web service port icon.
  - To generate a client from a WSDL file on your local filesystem, click **Local File** and browse to the WSDL file on your local filesystem.
  - To generate a client from a running web service, click **WSDL URL**, then type or paste the web service's URL. If you are behind a corporate firewall, click **Proxy Settings** and set your proxy host and port number.

---

**Note:** The WSDL file is downloaded when you finish the wizard.

---

4. Optionally, specify the package where the client files will be generated. When you complete the wizard, you can find the generated client files in the Projects window, in the Generated Sources node of the client project, or in the Files window, within the build folder.  
If you do not specify a package name, the IDE generates the client files in the default package, which is based on the namespace in the WSDL. You can change this package name later in the WSDL Customization editor.
5. Specify whether to use JAX-WS or the older JAX-RPC client style. JAX-WS is selected by default. However, older services may require JAX-RPC clients. Install the JAX-RPC Web Services plugin to create JAX-RPC clients.
6. If you want to use raw XML messages when invoking the web service, select **Generate Dispatch Code**. The client code is generated using the `java.xml.ws.Dispatch` interface instead of the usual service endpoint interface. This is an advanced feature and is unselected by default.
7. Click **Finish**.

#### 19.5.4 How to Call a Web Service Operation

A web service operation can be called from a web application, a Java application, or a MIDP client (MIDlet). For information on calling a web service operation from a MIDlet, see the Java ME Mobility documentation.

##### To call a web service operation:

1. Create a web service client.
2. If the web service client is deployed from a web application, you can call the web service from a servlet or from a JSP page. If the web service client is deployed from a Java application, use a Java source file instead. So, do one of the following:
  - Create a Java source file, such as a servlet.
  - Use the default `index.jsp` file that is created for you when you create a web application or a JSP file.
3. Open the file in the Source Editor and do one of the following:
  - Expand the Web Service References node and continue expanding subnodes until you get to the node representing the operation. Using your mouse, drag and drop the node to where you need it to be in the file.
  - Right-click in the method (for Java files) or anywhere in the Source Editor (for JSP file) from where you want to call the web service, and choose **Insert Code > Call Web Service Operation**. The Select Operation to Invoke dialog box appears. Expand the nodes and select a web service operation. Click **OK**.

The IDE adds the code required for calling the web service to the file.

#### 19.5.5 How to Asynchronously Call a Web Service Operation

When a client calls a JAX-WS web service operation asynchronously, the client does not need to wait for the response to be received. When a client uses this communication style, it consumes the web services either through the "polling" approach or the "callback" approach.

- **Polling.** The client invokes a web service method and repeatedly asks for the result. Polling is a blocking operation because it blocks the calling thread, which is why you do not want to use it in a GUI application.
- **Callback.** The client passes a callback handler during the web service method invocation. The handler's `handleResponse()` method is called when the result is available. This approach is suitable to GUI applications because you do not have to wait for the response. For example, you make a call from a GUI event handler and return control immediately, keeping the user interface responsive.

---

**Note:** You can use the IDE to generate skeleton asynchronous methods, as explained below.

---

#### To call a web service operation asynchronously:

1. Create a web service client.
2. If the web service client is deployed from a web application, you can call the web service from a servlet or from a JSP page.

If the web service client is deployed from a Java application, use a Java source file instead. The source file can be:

- A Java source file, such as a servlet, that you have created
  - The default `index.jsp` file that is created for you when you create a web application or a JSP file
3. In the Projects window, expand the Web Service References node, right-click the web service node (the first node within the Web Service References node), and choose **Edit Web Service Attributes**.
  4. In the Edit Web Service Attributes editor, within the PortType Operations node, expand the node with the same name as the web service operation you want to invoke.
  5. Select **Enable Asynchronous Client**. Click **OK**.
  6. Open the file in the Source Editor and do one of the following:
    - Expand the Web Service References node and continue expanding subnodes until you get to the node representing the asynchronous operation. Using your mouse, drag and drop the node to where you need it to be in the file.
    - Right-click in the method (for Java files) or anywhere in the Source Editor (for JSP file) from where you want to call the web service, and choose **Insert Code > Call Web Service Operation**. The Select Operation to Invoke dialog box appears. Expand the nodes and select the asynchronous operation. Click **OK**.

The IDE adds the code required for asynchronously calling the web service to the file.

#### 19.5.6 How to Deploy a Web Service Client

Right-click the project and choose one of the following:

- **Run.** Only takes the minimal steps needed to run the project. For example, if you only changed one file, only that file is sent to the server.
- **Deploy.** Undeploys and then deploys the complete module no matter what changes you made, even if none.

---

**Note:** After the first time a web application is deployed, it is redeployed automatically whenever you save changes to that application.

---

For web applications, unless the project in which the web service client is implemented is designed to be deployed as a stand-alone application, you should always deploy it by deploying the enterprise application that contains it. If you run the Run or Deploy commands on an individual module that is part of an enterprise application, the IDE only deploys that module itself. Any changes you have made in the other modules in the project are not propagated to the server.

**To deploy a client from within an enterprise application project:**

1. In the Projects window, add the web application module to the enterprise application.
2. Right-click the enterprise application project and choose one of the following:
  - Run. Only takes the minimal steps needed to run the project. For example, if you only changed one file, only that file is sent to the server.
  - Deploy. Undeploys and then deploys the complete module no matter what changes you made, even if none.

## 19.6 Creating RESTful Web Service Clients

NetBeans IDE helps you create two kinds of clients for RESTful web services:

- **Java Clients.** The IDE generates a Java client for a RESTful service based on the Jersey Client API. The client code can be generated in an existing Java class, or the IDE can generate a client class in a Java application, Web application, or NetBeans module. Java Client generation uses the Web Services Manager.
- **JavaScript Client.** The IDE generates a JavaScript file for a RESTful web service. For more information, see [Section 19.6.3, "How to Generate RESTful Web Service JavaScript Clients."](#).

### 19.6.1 How to Work with RESTful Web Service Clients

The procedure for developing a RESTful web service client is as follows:

1. Create the web service client.

Do one of the following:

- Create a Java client, either in an existing class or as a new class in a web application, Java application, or NetBeans module. See [Section 19.6.2, "How to Generate RESTful Web Service Java Clients."](#)
- Create JavaScript client in a web application. See [Section 19.6.3, "How to Generate RESTful Web Service JavaScript Clients."](#)

2. Build the web service client.

---

**Note:** In most cases it is not necessary to build the web service client. The IDE builds the client in the course of running it.

---

Do one of the following:

- If the client is in a Java application or stand-alone Web application, right-click the application's node and select **Build**.
  - If the client is in a Java EE module in an enterprise application, and you want to build the entire application, right-click the enterprise application's node and select **Build**.
  - If the client is in a Java EE module in an enterprise application and you only want to build the Java EE module, right-click the module and select **Build**.
  - If the client is in a NetBeans module, right-click the module's parent platform application and select **Build**.
3. Run the web service client.

You can deploy a RESTful web service client as a stand-alone Java or web application, as a part of an enterprise application, or as a NetBeans module in a Platform application.

Do one of the following:

- If the client is in a Java application or stand-alone Web application, right-click the application's node and select **Run**. See [Section 19.4.1, "Using Annotations and Deployment Descriptors."](#)
- If the client is in a Java EE module in an enterprise application, right-click the enterprise application's node and select **Run**. See [Section 19.5.6, "How to Deploy a Web Service Client."](#)
- If the client is in a NetBeans module, right-click the module's parent platform application and select **Run**.

For troubleshooting, see [Section 19.5.2, "How to Set a Proxy for Web Services and Clients."](#)

## 19.6.2 How to Generate RESTful Web Service Java Clients

NetBeans IDE can generate a Java client for a RESTful service based on the Jersey Client API. The client code can be generated in an existing Java class, or the IDE can generate a client class in a Java application, Web application, or NetBeans module. Java Client generation uses the Web Services Manager.

### How To Generate a RESTful client In an Existing Java Class

Use the Insert Code feature to insert RESTful service client code into a Java class.

#### To insert RESTful client code:

1. In the Java editor, Press Alt+Insert, or right-click and select **Insert Code** from the context menu. A menu of code to insert opens.
2. Choose **Generate REST Client**. The Available REST Resources dialog opens.
3. Browse for either a NetBeans project or a service registered in the IDE's Web Services Manager. You can register a service in the Web Services Manager if you have the URL of the service's WSDL or WADL.
4. You might need to set the authentication and class name. In most cases, the authentication and class name are read from the WADL and set automatically. You can also select whether to authenticate over SSL.
5. Click **OK**. A dialog opens asking if you want to generate Java objects from the XML schema references in the WADL file. Click **Yes**.

6. The IDE generates the RESTful client code. You still need to put in any authentication keys and write additional implementation code.

### How to Create a Client Class In an Application or Module

NetBeans IDE can generate a client class in a Java application, web application, or NetBeans module.

If you are generating a class in a NetBeans module, that module needs the RESTful Web Service Libraries and their dependencies on the module's classpath.

#### To create a client class:

1. In the Projects window, select the node of the Web application, Java application, or NetBeans module in which you want to create a RESTful client class.
2. Launch the New File wizard (Ctrl+N, or New File icon, or context menu of the node).
3. In the New File wizard, select the Web Services category and the RESTful Java Client file type. Click **Next**. The New RESTful Java Client panel opens.
4. Name the class and name the class package.
5. Browse for either a NetBeans project or a service registered in the IDE's Web Services Manager. You can register a service in the Manager if you have the URL of the service's WSDL or WADL.
6. You might need to set the authentication and class name. In most cases, the authentication and class name are read from the WADL and set automatically. You can also select whether to authenticate over SSL.
7. Click **OK**. A dialog opens asking if you want to generate Java objects from the XML schema references in the WADL file. Click **Yes**.
8. (NetBeans module only) Another warning might appear asking you to add modules to the classpath. Click **OK**.
9. (NetBeans module only) If you need to add modules to the classpath, right-click the module's node and open its Project Properties. Go to the Libraries section, and add the modules with the **Add Dependency** button. This button opens a list of module dependencies to browse.
10. The IDE generates the RESTful client code. You still need to put in any authentication keys and write additional implementation code.

### 19.6.3 How to Generate RESTful Web Service JavaScript Clients

For a local project that contains REST resource you can generate a JavaScript client that makes it easy for client applications to access the RESTful service. You can use a wizard in the IDE to generate JavaScript clients in web applications and HTML5 applications. In the wizard you also have the option of specifying a table as the user interface.

#### Perform the following steps to generate a JavaScript client for a RESTful web service.

1. Confirm that you have a local project with web service resources.
2. Right-click the project node and choose **New > Other > HTML5 > RESTful JavaScript Client**.
3. Click **Browse** to specify the location of the REST resource.

4. Select a UI option and click **Next**.
5. Specify a name and location for the generated HTML file. Click **Finish**.

When you click **Finish** the IDE generates the JavaScript file in the location that you specified in the wizard. If you selected Tablesorter as the UI for the client, the IDE generates the HTML file with the name and in the location that you specified.

#### 19.6.4 How to Use the Web Services Manager

The Web Service Manager is an expanded set of functionality added to the Web Services node in the IDE's Services tab. The RESTful Java Client wizards use the Web Service Manager. In addition, you can drag and drop SaaS operations from the Manager into PHP code, to create PHP RESTful clients. You can register a web service from a local file or from the service's WSDL or WADL.

**To register a web service in the Web Services Manager:**

1. In the Services window, right-click the Web Services node and choose **Add Web Service**. The Add Web Service dialog box opens.
2. In the Add Web Service dialog box, browse for the local file or type the URL of the service's WSDL or WADL file.
3. If necessary, set the proxy for the service or manually set the package name. Click **OK**.

**To add SaaS operations to a web application:**

1. Go to the related homepage of the web service and obtain the required keys and codes, if necessary.
2. Open the web service class where you want to add the service in the editor.
3. Expand the Web Services node in the Services window and drag the resource from the tab into the editor.
4. Supply any required information in the dialog boxes

All the plumbing code for accessing the service is generated by the IDE when you finish the drag-and-drop action.

**To access SaaS Service API documentation:**

- Right-click the resource node under the **Web Services** tab in the Services window and choose **View API Documentation** in the popup menu.

**To open the service WSDL or WADL**

- Right-click the resource node under the **Web Services** tab in the Services window and choose **View WSDL/WADL** in the popup menu.

### 19.7 Deploying and Testing Web Services and Clients

This section describes how to test JAX-WS or RESTful web services.

#### 19.7.1 How to Test a JAX-WS Web Service

For JAX-WS web services, depending on the container to which you deploy the web service, the IDE may provide you with functionality for testing the web service:

- **GlassFish Open Source Edition and Oracle WebLogic Application Servers.** The GlassFish and WebLogic servers come with their own Tester applications. When you right-click a web service node in the Projects window and choose **Test Web Service**, the IDE's default browser opens at this location:

```
context_path/web_service_nameService?Tester
```

The form that opens lets you test your web service implementation.

- **Tomcat Web Server.** When you test a web service deployed to the Tomcat Web Server, the browser opens and a web page is displayed, informing you that deployment has succeeded. To access this web page, right-click a web service node in the Projects window and choose **Test Web Service**. The IDE accesses this URL:

```
context_path/web_service_name?Tester
```

## 19.7.2 How to Test a RESTful Web Service

The IDE can generate a test client for RESTful web services.

---

**Note:** For a full NetBeans tutorial that shows how to test RESTful web services, see the following document:

<https://netbeans.org/kb/docs/websvc/rest.html#test-rest>

---

### To test RESTful web services:

1. Expand the web application project that contains the RESTful web services.
2. Right-click the RESTful Web Services node in the Projects window and choose **Test RESTful Web Services** in the popup menu.

The server is started, if it is not running already, and the web services are shown in a web page in the default browser. In the browser, you will find buttons and drop-down lists for testing the methods exposed by the web services made available by the project.

## 19.8 Creating Handlers

A handler is a Java class that provides a filtering mechanism for preprocessing and postprocessing the web service message, by intercepting it and acting on the request or response.

SOAP message handlers are user-written Java classes that can modify a SOAP message, which represents an RPC request or response. Handlers can be associated with a web service or web service client. A handler has access to the body and header blocks of a message, but not to the application code. Handlers are usually designed to work with the header blocks and to supplement the processing done by the application code.

Typical uses of handlers include:

- Logging and auditing
- Encryption and decryption
- Caching of data

There are two types of handlers: message handlers and logical handlers.

### 19.8.1 How to Create a Handler

Follow these steps to create a handler.

1. Right-click the project node for a web application or an EJB module and choose **New > Other**.
2. Under **Categories**, select **Web Services**. Under **File Types**, choose one of the following, depending on your needs:
  - **Message Handler**. Creates a simple message handler. Message handlers intercept the message as it makes its way from the client to the service and vice-versa. The generated class performs simple logging of the message request.
  - **Logical Handler**. Allows access to the message payload (but not headers or other protocol-specific information).
3. Name the handler and select or define the package that will contain it.
4. In the Source Editor, modify the default handler to suit your requirements.

### 19.8.2 How to Configure Handlers

When you configure a handler, you register it in the application's general deployment descriptor. The IDE automates the configuration of message handlers, when you take the steps below.

#### To configure a message handler:

1. In the Projects window, do one of the following:
  - For web services, expand the **Web Services** node, right-click the node for the web service and choose **Configure Handlers**.
  - For web service clients, expand the **Web Service References** node, right-click the node for the web service and choose **Configure Handlers**.
2. In the Configure SOAP Message Handlers dialog box, click **Add** and browse to the `MessageHandler` class. Click **OK**.
 

The message handler class is listed in the dialog box.
3. Click **OK** to complete the configuration of the SOAP message handler.

### 19.8.3 Testing and Using Handlers

1. Build and deploy the web service or client.
2. Expand the Servers node, right-click the server's node, and choose **View Server Log**.

The `server.log` file is displayed. For example, a logging message similar to the following is included if you used the default message handler:

```
message: Wed Jan 12 16:56:48 CET 2012--sayHi String_1:John |#]
```

This is the logging message generated by the SOAP message handler.

## 19.9 Using JAXB for Java-XML Binding

The Java Architecture for XML Binding API (JAXB) makes it easy to access XML documents from applications written in the Java programming language. It is the standard API for this activity.

The IDE provides tooling support for JAXB, principally by means of a wizard that turns various types of XML documents into Java classes. (To use it, choose **XML > JAXB Binding** in the New File wizard.) In addition, code templates are provided for marshalling and unmarshalling Java code to XML and back.

For the full JAXB specification, see the JAXB Homepage.

For a full tutorial on JAXB in the IDE, see: Getting Started with JAXB in NetBeans IDE.

### 19.9.1 How to Generate Java Classes from XML Schema

You can use the JAXB Wizard to generate Java classes from an XML schema file. Once you have the classes, you can use them in a variety of scenarios to traverse the elements and attributes of the XML schema file.

#### To generate Java classes from an XML schema document:

1. Make sure that you have installed the full distribution of the IDE, since the JAXB Wizard is only provided with the full distribution.
2. Create a project type to store the code that you will generate from the XML file. The JAXB Wizard can be used with the 'Java Application', 'Java Class Library', 'Web Application' and 'EJB Module' project types.
3. In the Projects window or Files window, right-click the project node and choose **New > Other**. In the New File wizard, choose **JAXB Binding** from the **XML** category. Click **Next**.

In the wizard, fill in the fields as described below:

- **Binding Name.** Specifies the name of the new JAXB binding, which will be used to identify it.
- **Project.** Displays the name of the current project.
- **Schema File.** The file that you want to work with can either be available locally or on-line.
- **Schema Type.** The following types of XML document are supported:
  - XML Schema
  - Relax NG
  - Relax NG Compact
  - XML DTD
  - WSDL
- **Package Name.** Specifies the package to which the Java objects will be generated.
- **Compiler Options.** Many compiler options are available, as described here in the Java EE 5 Tutorial. However, in relation to the JAXB Wizard, only the following are relevant and you can set them using checkboxes in the wizard:
  - **nv.** Do not perform strict validation of the input schemas. By default, strict validation of the source schema is performed before processing. This does not mean the binding compiler will not perform any validation; it simply means that it will perform less-strict validation.
  - **readOnly.** Force the compiler to mark the generated Java sources read-only. By default, the compiler does not write-protect the Java source files it generates.

- **npa.** Suppress the generation of package level annotations into `**/package-info.java`. Using this switch causes the generated code to internalize those annotations into the other generated classes.
  - **verbose.** Produce maximum compiler output, such as progress information and warnings.
  - **quiet.** Suppress compiler output, such as progress information and warnings.
  - **Use Extension.** By default, the compiler strictly enforces the rules outlined in the Compatibility chapter of the JAXB Specification. In the default (strict) mode, you are also limited to using only the binding customizations defined in the specification. By using this option, you will be allowed to use the JAXB Vendor Extensions.
  - **Use Binding File.** Lets you import and edit one or more JAXB binding customization files.
  - **Use Catalog File.** Lets you import and edit OASIS catalog files.
4. Click **Finish**.
  5. Open the Files window, and notice that the Java classes have been generated in the build folder.

### 19.9.2 How to Marshall XML Elements From Java Classes

The Java Architecture for XML Binding API (JAXB) provides a client application the ability to convert Java classes into a hierarchy of XML elements. The IDE provides a code template that you can use to generate a code snippet as the basis for this task.

#### To marshall Java classes to XML elements:

1. Type `jaxbm` in the Source Editor for Java files and then press Tab. The snippet is created, as follows:

```
try {
    javax.xml.bind.JAXBContext jaxbCtx= javax.xml.bind.JAXBContext.newInstance(
Object.class.getClass().getPackage().getName());
    javax.xml.bind.Marshaller marshaller = jaxbCtx.createMarshaller();
    marshaller.setProperty(javax.xml.bind.Marshaller.JAXB_ENCODING, "UTF-8");
//NOI18N
    marshaller.setProperty(javax.xml.bind.Marshaller.JAXB_FORMATTED_OUTPUT,
Boolean.TRUE);
    marshaller.marshal(Object.class , System.out);
} catch (javax.xml.bind.JAXBException ex) {
    // XXXTODO Handle exception

    java.util.logging.Logger.getLogger("global").log(java.util.logging.Level.SEVERE
, null, ex); //NOI18N
}
```

2. Now incorporate it in the rest of your code.

For more information see [Section 18, "Developing Applications Using XML."](#)

### 19.9.3 How to Unmarshall XML Elements to Java Classes

The Java Architecture for XML Binding API (JAXB) provides a client application the ability to convert XML data into a tree of Java classes. The IDE provides a code template that you can use to generate a code snippet as the basis for this task.

**To unmarshal XML elements to Java classes:**

1. Type jaxbu in the Source Editor for Java files and then press Tab. The snippet is created, as follows:

```
try {  
    javax.xml.bind.JAXBContext jaxbCtx =  
    javax.xml.bind.JAXBContext.newInstance(Object.class.getClass().getPackage().get  
    Name());  
    javax.xml.bind.Unmarshaller unmarshaller = jaxbCtx.createUnmarshaller();  
    Object.class = unmarshaller.unmarshal(new java.io.File("File path"));  
    //NOI18N  
} catch (javax.xml.bind.JAXBException ex) {  
    // XXXTODO Handle exception  
  
    java.util.logging.Logger.getLogger("global").log(java.util.logging.Level.SEVERE  
    , null, ex); //NOI18N  
}
```

2. Now incorporate it in the rest of your code.

For more information see [Section 18, "Developing Applications Using XML."](#)

## 19.10 Configuring Quality of Service

Web Service 'Quality of Service' encompasses components that enable interoperability between Java web services and .Net web services.

These components fall into four main categories:

- **Bootstrapping and configuration.** The process that is executed to create a Web service client that can access and consume a Web service.
- **Message security.** The following security configuration options are supported:
  - Targets to sign and encrypt options
  - Client/service Web services security
  - Client/service trust options
  - Advanced configuration options
  - Keystore configuration options
- For a tutorial on this topic, see: [Advanced Web Service Interoperability](#).
- **Message optimization.** Ensures that Web services messages are transmitted over the Internet in the most efficient manner. Message optimization is achieved in Web services communication by encoding messages prior to transmission and then de-encoding them when they reach their final destination.
- **Reliable messaging.** Measured by a system's ability to deliver messages from point A to point B without error.

For a full guide on all these topics, see the WSIT Tutorial.

### 19.10.1 How to Configure Quality of Service

The **Quality of Service** tab in the Web Service Attributes Editor is for configuring components that enable interoperability between Java web services and .Net web services.

**To access the Quality of Service tab:**

1. In the Projects window, expand the project node for the project where your web service or client is defined.
2. For web services, expand the Web Services node, for web service clients, expand the Web Service Clients node. Right-click the node for your web service or client and choose **Edit Web Service Attributes** from the pop-up menu.

The Web Service Attributes Editor opens with the **Quality of Service** tab open.

On the service side, you can now use the Service Port Binding subsection, the Operation subsection, the Input Message subsection, and the Output Message subsection. On the client side, you can use the Transport subsection, the Security subsection, and the Advanced Configuration subsection.

### **19.10.2 How to Set Port Binding**

In the **Quality of Service** tab's Port Binding section, on the service side, you can configure MTOM, reliable messaging, and interoperable security policy.

**To set port binding options:**

1. Right-click the web service that you want to edit in the Projects window and choose **Edit Web Service Attributes** in the popup menu.
2. Click the **Quality of Service** tab and expand the **Port Binding** node (if not already expanded).
3. Specify the Version Compatibility option.

You can select the .NET/METRO version compatibility from the drop-down list. To enable more recent versions of .NET, you must add the latest METRO library to your project classpath.

4. Select any other options that you want to enable.

The port binding options in the Quality of Service tab are disabled by default.

5. Click **OK**.

**To add a METRO library to the classpath:**

1. Right-click your project's node in the Projects window.
2. Select **Properties**. The Properties dialog opens.
3. In the Properties dialog, select the **Libraries** category.
4. Leave the **Compile** tab selected and click the **Add Libraries** button. A list of available libraries opens.
5. Browse for the latest METRO library, select it, and click **Add Library**. The list of libraries closes.
6. The selected METRO library appears in your list of compile-time libraries. Click **OK** to exit the Properties dialog.

### **19.11 Securing an Operation**

In the Operation section of the Quality of Service tab, on the service side, you can add Quality of Service security to a specific web service operation.

At times, you may need to configure different operations with different supporting tokens. You may wish to configuring security at the operation level, for example, in the

situation where only one operation requires a UsernameToken to be passed and the rest of the operations do not require this, or in the situation where only one operation needs to be endorsed by a special token and the others do not.

In the Operation section of the Quality of Service configuration editor, you can select the following options for securing a web service operation:

- **Transaction.** Select an option from the Transactions list to specify a level at which transactions will be secured. For this release, transactions will only use SSL for security.
- **Secure This Operation.** Select this option to secure the web service operation. This option will be grayed out if Secure All Service Operations (in the PortBinding section) is selected because it would be redundant. Once selected, the list of security mechanisms is enabled.
- **Security Mechanism.** Select a security mechanism from the list. The security mechanisms are fully functional as selected.

### 19.11.1 How to Secure an Operation's Input Messages

In the Input Message section you can specify the parts of the incoming message that require integrity protection (digital signature) and/or confidentiality (encryption). When you do this, the specified part of the message, outside of security headers, requires signature and/or encryption. For example, a message producer might submit an order that contains an orderID header. The producer signs and/or encrypts the orderID header (the SOAP message header) and the body of the request (the SOAP message body). Parts that can be signed and/or encrypted include the body, the header, the local name of the SOAP header, and the namespace of the SOAP header.

You can also specify arbitrary elements in the message that require integrity protection and/or confidentiality. Because of the mutability of some SOAP headers, a message producer may decide not to sign and/or encrypt the SOAP message header or body as a whole, but instead sign and/or encrypt elements within the header and body. Elements that can be signed and/or encrypted include an XPath expression or a URI which indicates the version of XPath to use.

#### How to secure an operation's input message:

1. Right-click the web service that you want to edit in the Projects window and choose **Edit Web Service Attributes** in the popup menu.
2. Click the **Quality of Service** tab and expand the Operation node.
3. Expand the **Input Message** node.

The options that are available under the Input Message node depend upon the Security Mechanism of the service.

4. Select the security options for the message.

If available for your security mechanism, you can click Message Parts to specify which parts of the message need to be encrypted, signed, and/or required.

5. Click **OK**.

### 19.11.2 How to Secure a Message

You can use the Message Security Bindings dialog box to specify the message security bindings for a web service endpoint or port. You can create and modify the security bindings by opening the deployment descriptor XML file in the visual editor. You can

edit the XML file manually or use the dialog boxes in the visual editor to help you edit the bindings.

**To edit message security bindings:**

1. Expand the Configuration Files node in the Projects window and double-click the XML file to open the file in the visual editor.
2. Click the Web Services tab in the visual editor.
3. Expand the node for the Service Reference.
4. Expand the **Port Information** node.
5. Click **Edit Bindings** to open the Message Security Bindings dialog box.

In the Message Security Bindings dialog box the Authorization Layer specifies the message layer at which authentication is performed. The value must be SOAP.

6. Specify or edit the Provider ID.

The Provider ID specifies the authentication provider used to satisfy application-specific message security requirements. If not specified, a default provider is used, if it is defined for the message layer. If no default provider is defined, authentication requirements defined in the message-security-binding are not enforced.

7. Click New to create a Message Security Entry or select an entry and click Edit to open the Message Security dialog box.
8. Enter a Java Method Name and select the authorization values from the drop-down list. Click **OK**.

### 19.11.3 How to Secure an Operation's Output Messages

In the Output section you can specify the parts of an outgoing message that require integrity protection (digital signature) and/or confidentiality (encryption). When you do this, the specified part of the message, outside of security headers, requires signature and/ or encryption. For example, a message producer might submit an order that contains an orderID header. The producer signs and/or encrypts the orderID header (the SOAP message header) and the body of the request (the SOAP message body). Parts that can be signed and/or encrypted include the body, the header, the local name of the SOAP header, and the namespace of the SOAP header.

You can also specify arbitrary elements in the message that require integrity protection and/or confidentiality. Because of the mutability of some SOAP headers, a message producer may decide not to sign and/or encrypt the SOAP message header or body as a whole, but instead sign and/or encrypt elements within the header and body. Elements that can be signed and/or encrypted include an XPath expression or a URI which indicates the version of XPath to use.

**How to secure an output message:**

1. Right-click the web service that you want to edit in the Projects window and choose **Edit Web Service Attributes** in the popup menu.
2. Click the **Quality of Service** tab and expand the **Operation** node.
3. Expand the **Output Message** node.

The options that are available under the Input Message node depend upon the Security Mechanism of the service.

4. Select the security options for the message.

If available for your security mechanism, you can click Message Parts to specify which parts of the message need to be encrypted, signed, and/or required.

5. Click **OK**.

#### 19.11.4 How to Set Transport Options

In the Quality of Service tab's Transport section, on the client side, you can select optimal encoding or optimal transport for web service clients interoperating with web services.

**How to set transport options (client side):**

1. Right-click the web service that you want to edit in the Projects window and choose **Edit Web Service Attributes** in the popup menu.
2. Click the **Quality of Service** tab and expand the **Transport** node.
3. Select the transport options for the client.
4. Click **OK**.

#### 19.11.5 How to Set Client Security

In the Quality of Service tab's Security section, on the client side, you may configure a user name and password for some of the security mechanisms. For this purpose, you can use the default Username and Password Callback Handlers (when deploying to the Application Server), specify the default SAML Callback Handler, specify a default user name and password for development purposes, or create and specify your own Callback Handlers if the container you are using does not provide defaults.

**How to set client security (client side):**

1. Right-click the web service that you want to edit in the Projects window and choose **Edit Web Service Attributes** in the popup menu.
2. Click the **Quality of Service** tab and expand the **Security** node.
3. Select the security options for the client.
4. Click **OK**.

# 20

---

## Developing HTML5/JavaScript Applications

This chapter describes the procedures and tools available for developing HTML5/JavaScript applications in the NetBeans IDE, including information on creating, running, integrating and debugging your application.

This chapter contains the following sections:

- [About Developing HTML5 Applications](#)
- [Working with HTML5/JavaScript Applications](#)
- [Creating an HTML5/JavaScript Application Project](#)
- [Running an HTML5 Application](#)
- [Integrating an HTML5 Project with a Browser](#)
- [Inspecting HTML5 Code with the Browser](#)
- [Changing Browser Screen Sizes](#)
- [Creating HTML5 Templates](#)
- [Creating Cascading Style Sheets](#)
- [Creating JavaScript Files](#)
- [Using Grunt and Gulp Build Tools](#)

### 20.1 About Developing HTML5 Applications

An HTML5 application project is an application that is displayed in a browser on your desktop or a mobile device. An HTML5 application is typically comprised of HTML, CSS and JavaScript files. The JavaScript that is used to manipulate and process objects in the application is generally handled on the client side by the JavaScript engine in the web browser. An HTML5 application is often used as a client that consumes various web services.

The IDE provides the following tools to support the development of HTML5 applications and HTML5 in PHP and Java web applications:

- Wizards for creating HTML5/JavaScript applications and files
- JavaScript code-completion, debugging and testing
- Code inspection using Chrome or bundled WebKit browsers
- CSS and CSS preprocessor file editors and styling tools
- Wizard to generate JavaScript clients for RESTful web services.
- Network Monitor for REST web service calls and WebSocket communication

The IDE also provides tools for packaging HTML5 applications as native mobile applications via Apache Cordova.

## 20.2 Working with HTML5/JavaScript Applications

The following work flow shows how to work with HTML5/JavaScript applications.

### How to develop an HTML5/JavaScript application:

1. Create a new HTML5/JavaScript Application project in the New Project wizard. See [Section 20.3, "Creating an HTML5/JavaScript Application Project."](#)
2. Create and edit HTML, JavaScript and CSS files using wizards and the editor.
3. Specify the target web browser and platform. See [Section 20.4, "Running an HTML5 Application."](#)
4. Run the application and debug the JavaScript.

## 20.3 Creating an HTML5/JavaScript Application Project

The IDE provides wizards for creating HTML5 projects and the HTML, CSS and JavaScript files that HTML5 projects typically contain. When you use the New Project wizard to create an HTML5 project you have the option to create the project based on a site template. In the wizard you can specify a site template that is located on your local system or download one of the listed site templates from an online repository.

### To create an HTML5/JavaScript application project:

1. Choose **File > New Project** from the main menu.
2. Select the **HTML5/JS Application** project template from the **HTML5/JavaScript** category.

Alternatively, select the **HTML5/JS Application with Existing Sources** to import an existing project as an HTML5/JS application.

Click **Next**.

3. Specify the project name and location.
4. Click **Next** in the Name and Location panel.
5. Specify a site template for the application or select **No Site Template** to create an empty application that only contains an `index.html` file. Click **Next**.
6. Specify the files to be created for such tools as npm, Bower, Grunt, Gulp and added to the project (optional).
7. Click **Finish**.

When you click **Finish**, the IDE creates the project and displays a node for the project in the Projects window.

### 20.3.1 How to Create HTML5/JavaScript Applications for Mobile Platforms

The IDE includes an HTML5/JavaScript site template that enables you to easily create a mobile application that is built on the Cordova platform. Cordova enables you to create applications that take advantage of some of the native functionality that is provided on Android and iOS mobile devices. You can create an HTML5/JavaScript application using the Cordova template or add support for Cordova to your HTML5/JavaScript application.

You must install Cordova on your local system before you can create a Cordova project. The IDE will prompt you to install Cordova if your Cordova installation is not detected when you create a Cordova application or when you add support for Cordova to an application. You can find the installation instructions and additional details about the platform at the following site.

<http://cordova.apache.org/>

---

**Note:** The IDE uses Git and NodeJS to work with Cordova. You must properly configure the proxy settings for Git and NodeJS if you are using a proxy.

---

To develop an application for a mobile platform you must first install the required platform software and the platform must be recognized by the IDE. If you are developing for the Android platform you need to specify the location of the Android SDK on your local system.

---

**Note:** The iOS device and emulator are only available on OS X. The IDE automatically detects the iOS installation details on OS X.

---

#### **How to register the Android mobile platform with the IDE:**

1. Open the Options window in the IDE.
2. Click the **Mobile Platforms** tab in the Miscellaneous category of the Options window.
3. Type the location of the Android SDK installation or click **Browse** to locate the installation on your local system.

#### **How to create a new Cordova application:**

1. Choose **File > New Project** from the main menu.
2. Select the Cordova Application project template from the HTML5/JavaScript category. Click **Next**.
3. Specify the project name and location.
4. Click **Next** in the Site Template panel.  
The Cordova Hello World template is already selected.
5. Specify the JavaScript files to add to the project (optional).  
You can modify the JavaScript Files that are included in the project at any time in the JavaScript Files category of the Project Properties window.
6. Specify the Cordova project configuration details (optional).  
You can edit the configuration details at any time in the Cordova category of the Project Properties window.
7. Click **Finish**.

#### **How to add Cordova support to an HTML5 application:**

1. Right-click the project node in the Projects window and choose **Properties**.
2. Select the **Cordova** category in the Project Properties window.
3. Click **Create Cordova Resources** in the Cordova panel.

After you click **Create Cordova Resources**, the IDE adds creates a Cordova configuration file in the project and enables two tabs in the Cordova panel where you can specify additional details.

4. Specify an application ID, name and version for the Cordova application. Click OK.

#### **How to add Cordova plugins to a Cordova application:**

1. Right-click the Cordova application in the Projects window and choose Properties in the popup menu.
2. Select the **Cordova** category in the Project Properties window and click the **Plugins** tab.
3. Select the Cordova plugin from the list of available plugins and click the right arrow ( > ) to move the plugin to the list of selected plugins. Click OK.

### **20.3.2 How to Add Support for HTML5 Features to an Application**

You can incorporate JavaScript and CSS to Java web applications and PHP applications by adding the files to the project. You can also use the Project Properties window to include JavaScript libraries and configure CSS preprocessors that you want to use in the project. You can run the applications on mobile browsers and on Android and iOS devices and emulators and debug JavaScript files. It is not necessary to change the project type of an application or create a new project.

#### **How to add JavaScript libraries to a Java web or PHP application:**

1. Right-click the project node in the Projects window and choose Properties in the popup menu.
2. Select the JavaScript Libraries category.
3. Select the required category under the JavaScript Libraries node that you want to add, update, or remove from the project and click OK.

### **20.3.3 How to Create a Node.js Application**

The IDE provides the following wizards for creating Node.js projects:

- Node.js Application
- HTML5/JS Application with Node.js
- Node.js Application with Existing Sources

#### **To create a new Node.js application project:**

1. Choose **File > New Project** from the main menu.
2. Select the Node.js Application project template from the HTML5/JavaScript category. Click **Next**.
3. Specify the name and location for your project. Click **Next**.
4. In the Tools panel select the options required for your project.
5. Click **Finish**.

When you click **Finish**, the IDE creates the project and displays a node for the project in the Projects window.

**To create a new HTML5/JS Application with Node.js:**

1. Choose **File > New Project** from the main menu.
2. Select the HTML5/JS Application with Node.js project template from the HTML5/JavaScript category. Click **Next**.
3. Specify the name and location for your project. Click **Next**.
4. In the Express panel select Express Generator options (optional). Click **Next**.
5. In the Tools panel select the options required for your project (optional). Click **Next**.
6. Click **Finish**.

When you click **Finish**, the IDE creates the project and displays a node for the project in the Projects window.

**To create a Node.js Application with existing sources:**

1. Choose **File > New Project** from the main menu.
2. Select the Node.js Application with Existing Sources project template from the HTML5/JavaScript category. Click **Next**.
3. Specify the source folder, site root, name and location for your project. Click **Next**.
4. Click **Finish**.

When you click **Finish**, the IDE creates the project and displays a node for the project in the Projects window.

## 20.4 Running an HTML5 Application

An HTML5 application project is typically run in a browser on your desktop or a mobile device. When you run an HTML5 application on your local system for testing or debugging you do not need to compile or deploy the application to a server because the JavaScript engine in the browser generally handles any scripts in the application. To run the application you first specify the browser and then run the application.

You can run an HTML5 application on an emulator for the mobile platform or deploy the application to the mobile device.

### 20.4.1 How to Specify the Browser

When you want to run an HTML5 application or file from the IDE you need to specify the target browser. The IDE enables you to easily select any of the browsers that are installed on your local system or the embedded WebKit browser. If you are developing an application for a mobile device you can choose to run the application in an emulator for the mobile device or deploy directly to a mobile device.

**To specify the browser for an HTML5 application:**

- To specify the target browser for the application, perform one of the following steps:
  - Select the application in the Projects window and then select the browser from the drop-down list in the Toolbar.
  - Open the Project Properties window and select the browser in the Run panel of the window.

The list of browsers is organized in a table to help you select the target browser. If you select a browser or device in the top row of the table you can debug the application in the IDE and use other features that are provided when the IDE is integrated with the browser. The table organizes the browsers in the following columns:

- The **Browsers** column displays the web browsers that are available on your local system. When you select a browser in the **Browsers** column the application will launch in a window in the selected browser when you run the application.
- The **Mobile Device Browser** column displays a list of mobile devices that you can select as the target platform and the browsers that are available on that platform. When you select a browser in the **Mobile Device Browser** column, the IDE will either run the application in the browser on the selected mobile device or in an emulator for the device.
- The **Cordova** column displays a list of mobile devices that you can select as the target platform. When you select a device in the **Cordova** column, the IDE will package the project as a native mobile application and deploy the application to the mobile device or emulator. Click **Configure Cordova** to open the **Cordova** panel in the Project Properties window.

You must install Apache Cordova on your local machine if you want to specify a target in the **Cordova** column.

If you want to run an application in a browser on an Android mobile device or in the Android emulator you need to install the Android SDK on your local system and register the location of the Android SDK in the Options window. The iOS device and simulator are only available on OS X.

If you want to debug an HTML5 application or file, or synchronize the IDE with your browser to inspect your code, you should select either Chrome with NetBeans Connector or Embedded WebKit Browser as the target browser.

## 20.4.2 How to Run an Application

After you specify the browser, you can run the application in the following ways.

- Right-click the project node in the Projects window and choose **Run** in the popup menu.
- Select the application in the Projects window and then click **Run** in the Toolbar or choose **Run > Run Project** (F6) in the main menu.

When you run an application, the IDE automatically opens the application in the selected browser or mobile device and opens the **Browser Log** in the **Output** window.

If more than one project is open you can specify one of the projects as the Main Project. After you specify the project as the Main Project, the F6 keyboard shortcut will always run the Main Project regardless of the project that is selected in the Projects window.

## 20.4.3 How to Run a File

You can run an HTML file in an HTML5 application in the following ways.

- Right-click the HTML file in the Projects window and choose **Run File** from the popup menu (Shift-F6).
- If the HTML file is open in the editor, you can right-click in the file and choose **Run File** (Shift-F6) from the popup menu or click the **Run File** button in the Selection tab of the CSS Styles window.

When you run a file, the IDE opens the file in the target browser for the application.

## 20.5 Integrating an HTML5 Project with a Browser

A NetBeans extension is available for the Chrome browser that enables the IDE and the browser to communicate. To integrate the IDE with the Chrome browser, you need to install the NetBeans Connector extension from the Chrome Web Store or install the extension manually. After you install the extension you can perform the following tasks in the Chrome browser:

- Use Inspect mode to select and locate elements in your code
- Debug JavaScript files
- Select various browser sizes from the NetBeans actions menu

Do not close the infobar in the tab in the Chrome browser that informs you that "NetBeans Connector is debugging this tab". Closing the infobar will disconnect the IDE from the Chrome browser. To reconnect the IDE, close the tab where the application was running and re-run the application from the IDE.

**Note:** You only need to install the extension one time.

### 20.5.1 How to Install the Extension from the Chrome Web Store

After downloading, you can perform the following steps to install the NetBeans extension for the Chrome browser.

#### To install the NetBeans extension for Chrome:

1. Right-click an HTML5 project node in the Project Projects window and choose Properties in the popup menu.  
Alternatively, you can select a target browser for the project from the drop-down list in the toolbar.
2. In the Run panel of the Properties window, select **Chrome with NetBeans Connector** in the server drop-down list. Click OK.
3. Right-click the HTML5 project node in the Projects window and choose Run.  
When you click Run, the IDE opens a dialog that prompts you to install the extension from the Chrome Web Store.
4. Click **Go to Chrome Web Store** in the Install Chrome Extension dialog box.
5. Click **Free** in the Chrome Web Store to install the extension and confirm that you want to add the NetBeans Connector extension.
6. After the extension is installed in the Chrome browser, click **Re-run Project** in the Install Chrome Extension dialog box to open the HTML5 application in the Chrome browser.

You will see an infobar in the tab in the Chrome browser that informs you that "NetBeans Connector is debugging this tab".

### 20.5.2 How to Install the Extension Manually

If you are unable to connect to the Chrome Web Store you can install the NetBeans Connector extension manually by performing the following steps.

#### To install the Chrome extension manually:

1. Open your Chrome browser to `chrome://extensions/`.

2. On your local system, navigate to the `NETBEANS_INSTALL/webcommon/modules/lib` directory, where `NETBEANS_INSTALL` is the installation of the IDE.
3. Drag the NetBeans Connector extension (`netbeans-chrome-connector.crx`) from the `lib` directory into the Extensions tab in the Chrome browser window and approve the addition of the extension.
4. Right-click the HTML5 project node in the Projects window and choose **Run** to open the application in the browser.

You will see an infobar in the tab in the Chrome browser that informs you that "NetBeans Connector is debugging this tab".

## 20.6 Inspecting HTML5 Code with the Browser

You can use the Inspect mode to help you locate and edit the source code in an HTML5 application. When Inspect mode is enabled, information about some of the actions that you perform in the browser such as selecting and highlighting elements is communicated to the IDE. When you run the application, you can use your browser to help you locate HTML elements in your source code. When you select or place your cursor over elements in the browser window, the element is simultaneously highlighted or selected in the Browser DOM window of the IDE. When you select an element, the style rules for the element are automatically displayed in the Selection tab of the CSS Styles window.

### To inspect your application with a browser:

1. Set the target browser for the application to one of the browsers or devices listed in top row in the browser selection drop-down list.

The browsers and devices in the top row of the browser selection drop-down list support Inspect mode.

2. Run the application or file.

When you run the application the target browser opens and displays the index page or file in the window.

3. Enable Inspect mode (Ctrl+Shift+S) in the browser.

In the Chrome browser you can enable Inspect mode in the NetBeans Actions menu. In the Embedded WebKit Browser you can click the Inspect button in the toolbar. Alternatively, you can click the Inspect icon in the Browser DOM window.

4. Move your cursor over or select an HTML element in the page in the browser window.

When you move your cursor over or select an element in the browser the contents of the following windows in the IDE are updated to synchronize with your actions in the browser.

- Browser DOM window. The Browser DOM window displays a DOM tree of the current page. When you move your cursor over an element in the browser the element is highlighted in the Browser DOM window. You can double-click an element in the window to open the source file in the editor.
- CSS Styles window. The Selection tab of the CSS Styles window displays the style rules for the selected element. You can edit the rules for the element in the CSS Styles window.

When you enable the Inspect mode in the browser, any element that is in the 'hover' state remains in the state until you disable the Inspect mode. You can select an element

in the hover state and inspect and modify the properties of the hover state and view the changes to the element in your browser.

To enable Inspect mode in the Chrome browser you need to install the NetBeans extension to integrate the IDE with the browser.

## 20.7 Changing Browser Screen Sizes

A single HTML5 application might be viewed on various devices with different screen sizes. The IDE can help you develop projects that can respond and adapt to different display devices by enabling you to view pages at different screen sizes. When you run the HTML5 application from the IDE you can choose to view the pages in one of the default screen sizes or create a custom screen size.

You can view your application in any browser, but if you want to be able to quickly switch between screen sizes you need to specify one of the following integrated browsers as the target browser:

- Chrome with NetBeans Connector
- Embedded WebKit Browser

When you view the application in one of the integrated browsers you can choose the display size in a menu in the browser.

You can specify the target browser for the application in the Project Properties window or in the Set Project Configuration drop-down list in the toolbar of the IDE.

### 20.7.1 How to Switch Between Screen Sizes

The integrated browsers provide menus that enable you to switch between common screen sizes that are pre-configured in the menu. When you select a new screen size the display area of the browser resizes to the selected size. By changing the screen size you can view how the layout of the application responds to different sizes, for example, as a result of media queries that are specified in the style sheet.

After you launch the application you can select a screen size from the menu in the browser. The location of the menu for selecting the screen size options depends upon the target browser.

- **Chrome with NetBeans Connector.** Click the NetBeans icon in the URL location bar to open the menu and select a screen size.
- **Embedded WebKit Browser.** Select a screen size in the Other Options menu or click an icon in the toolbar. The toolbar in the Web Browser window contains the following elements.
  - **Display device icons.** Click a display device icon to resize the display area to the selected size. You can place your cursor over an icon to display the dimensions of the display device. You can modify which icons are displayed in the toolbar by choosing Customize in the Other Options drop-down list.
  - **Other Options menu.** Select a window size in the drop-down list to view the page contents in the selected window size. The drop-down list displays all the available window sizes. Choose Customize to open a dialog that enables you to modify, create or remove window sizes.
  - **Magnification selector.** Type a value in the text field or select a value in the drop-down list to change the magnification of the contents of the page.
  - **Inspect icon.** Select to enable Inspect mode in the page.

When you run an HTML5 application the location of the current page is displayed in the Location bar. Pages are rendered using the WebKit browser engine running on the embedded lightweight server (localhost:8383). You can modify the server details in the Project Properties window.

### 20.7.2 How to Create a Custom Screen Size

You can create custom sizes for windows by selecting Customize in the screen size menu. When you select Customize the browser displays a window that enables you to create a new size and to modify the default screen size options. You can also edit the contents of the menu.

## 20.8 Creating HTML5 Templates

An HTML5 template enables you to quickly create an HTML5 project that contains the files and a structure that is based on the template. A site template can help you quickly develop a project by providing some JavaScript, CSS and HTML files that you can then modify according to your needs.

When you create an HTML5 project in the New Project wizard, you can choose to use a site template .zip archive that is located on your local system, specify the URL of an online template or select a template from the list of popular site templates in the wizard.

### 20.8.1 How to Create a Site Template

You can use the IDE to create a site template that is based on your project by performing the following steps.

**To create a site template:**

1. In the Projects window, open the HTML5 project that is the model for your template.
2. Right-click the HTML5 project and choose Save as Template in the popup menu.
3. Specify the name and where you want to save the template.
4. Specify the files that you want to include in the template. Click Finish.

Any template that you save is automatically added to the Select Template drop-down list in the New HTML5 Application wizard.

## 20.9 Creating Cascading Style Sheets

You can generate a cascading style sheet (CSS) from an existing document type definition (DTD), create an empty style sheet and edit the file in the CSS editor or use a CSS preprocessor to generate style sheets.

When you are editing a CSS file the IDE provides windows and dialog boxes that can help you create CSS rules and declarations. After you create a rule you can add declarations to the rule by editing the style sheet in the editor or in the CSS Styles window. When editing CSS files, you can take advantage of standard editor features, including completion, indentation, syntax coloring, and standard clipboard commands.

If you want to use a CSS preprocessor to generate a style sheet the IDE supports the LESS and Sassy CSS syntax. To compile the LESS or Sass files and generate the style sheet you must first install the LESS or Sass preprocessor software on your local

system. After you install the preprocessor software you can configure the software to compile the files and generate the CSS files each time that the LESS or Sass file is saved.

### 20.9.1 How to Create a Cascading Style Sheet

You can use the New File wizard to create a new cascading style sheet in your project. After you create the style sheet you can see the new file in the Projects window and Files window. When you create the style sheet the file opens in the Source Editor.

**To create a cascading style sheet:**

1. Choose **File > New File** from the main menu.
2. Choose **HTML5/JavaScript > Cascading Style Sheet**. Click Next.
3. Enter a name and location for the CSS file. Click Finish.

### 20.9.2 How to Add a CSS Rule to a Cascading Style Sheet

You can create a CSS rule in a style sheet by typing directly in the source editor or by using the Edit CSS Rules dialog box. You can open the Edit CSS Rules dialog box from the CSS Styles window.

**To add a CSS Rule to a style sheet:**

1. Open the CSS file in the source editor.
2. Open the CSS Styles window (**Window > Web > CSS Styles**).
3. Click **Document** in the CSS Styles window if the Style Sheets tree is not displayed in the CSS Styles window.
4. Click the Edit CSS Rules icon ( ) to open the Edit CSS Rules dialog box.
5. Select a Selector Type in the dialog box.
6. Type a name in the Selector text field for the `id` or `class` element or select an element from the drop-down list.
7. Confirm that the correct style sheet is selected in the Style Sheet drop-down list.
8. Specify any additional properties for the rule. Click OK.

After you click OK you can see that the new rule is listed in the CSS Styles window and the Navigator window and that the rule is added to the CSS file in the editor.

### 20.9.3 How to Add a Property to a CSS Rule

You can add properties and values to a CSS rule by manually editing the style sheet in the source editor, by selecting values in the CSS Styles window or by specifying values in the Add Property dialog box.

**To add a property to a CSS rule in the CSS Styles window:**

1. Open the CSS file in the source editor.
2. Open the CSS Styles window (**Window > Web > CSS Styles**).
3. Click **Document** in the CSS Styles window if the Style Sheets tree is not displayed in the CSS Styles window.
4. Select the rule that you want to edit in the top section of the CSS Styles window.

The properties of the selected rule are displayed in the bottom section of the CSS Styles window.

5. Click **Add Property** in the left column of the table in the bottom section of the CSS Styles window and choose a property in the drop-down list.
6. Type a value for the property in the text field or choose a value from the drop-down list in the right column of the table and hit the return key on your keyboard.

When you hit the return key the property and value are added to the rule in the CSS editor.

**To add a property to a CSS rule in the Add Property dialog box:**

1. Open the CSS file in the source editor.
2. Open the CSS Styles window (**Window > Web > CSS Styles**).
3. Click **Document** in the CSS Styles window if the Style Sheets tree is not displayed in the CSS Styles window.
4. In the source editor, place the insert cursor between the brackets of the CSS rule that you want to edit.
5. Click the Add Property icon ( ) in the bottom section of the CSS Styles window to open the Add Property dialog box.
6. Select a value for the property from the drop-down list or type a value in the text field in the right column of the table.
7. Specify any additional properties for the rule. Click OK.

#### 20.9.4 Creating a CSS Preprocessor File

You can create Sass or LESS preprocessor files that can be compiled to generate your CSS files. The IDE provides wizards to generate the preprocessor files and to configure the preprocessor software to watch a location for changes to the files. You must install the CSS preprocessor software on your local system if you want to locally compile the preprocessor files to generate the CSS files.

You can use code completion and syntax coloring in the editor to help you edit CSS preprocessor files.

**How to create a CSS preprocessing file:**

1. Choose **File > New File** from the main menu.
2. Choose **Sass File** or **LESS File** in the **HTML5/JavaScript** category. Click **Next**.
3. Enter a name and location for the new file.
4. (Optional) Select **Compile Sass (or LESS)** files on **Save**.

When you enable compile on save for CSS preprocessor files you can specify the directory that contains the preprocessor files and the directory that will contain the compiled CSS file.

5. (Optional) Modify the input and output directories.

When the preprocessor file in the input directory is modified and saved the CSS file is automatically recompiled and created in the specified output directory.

6. Click **Finish**.

## 20.9.5 How to Generate CSS Files Using a CSS Preprocessor

The IDE can be configured to compile LESS and Sass preprocessor files to generate CSS files. When you are editing the CSS preprocessor files in your project you can compile the files locally to view how your changes affect the layout of the application. The CSS preprocessor software must be installed on your local system if you want to compile the files locally.

You can find instructions for installing LESS and Sass at the following sites.

<http://sass-lang.com/>

<http://lesscss.org/>

If you want the preprocessor files automatically compiled when you save them you need to register the CSS preprocessor software with the IDE. You can specify the location of the executables for the CSS preprocessor software in the Options window.

### How to register the CSS preprocessor software in the IDE:

1. Open the Options window.
2. Click the **CSS Preprocessors** tab in the HTML/JS category.
3. Type the path to the executable of the preprocessor software or click Browse to locate the executable on your local system.
4. (Optional) Select Open Output on Error if you want the IDE to open the Output window when an error is encountered during compilation. This option is enabled by default.
5. (Optional) Select Generate extra debug information if you want debug information generated in the CSS file. This option is enabled by default.

You can configure the software to watch the location of your preprocessor files in your application and automatically compile the CSS file when the preprocessor file is saved. You can enable compile on save for CSS preprocessor files when you create the file or in the Project Properties window.

---

**Note:** If you modify the generated CSS files your changes will be lost the next time that the CSS preprocessor files are compiled.

---

### How to enable compile on save in the Project Properties window:

1. Right-click the project node in the Projects window and choose Properties in the popup menu.
2. Select the **CSS Preprocessors** category and select the tab for the preprocessor for your project (LESS or Sass).
3. Select **Compile Sass Files on Save** (or Compile LESS Files on Save).

If you see a warning in the Project Properties window that the executable is not selected, click Configure Executables to open the Options window and specify the location of the preprocessor executable.

4. In the Watch table, specify the Input directory that contains the preprocessor files.
5. Specify the Output directory for the CSS file that is generated.
6. (Optional) Specify any compiler options. Click OK.

## 20.10 Creating JavaScript Files

The IDE provides a wizard for creating JavaScript files and support for JavaScript in the source editor. When you run the application the Browser Log tab opens in the Output window.

### 20.10.1 How to Create a JavaScript File

You can add JavaScript files to your project in one of two ways:

- as a JavaScript file
- as a JavaScript library

You can use the New File wizard to add a JavaScript file to a project at any time. You can also use the wizard to create a file of common JavaScript functions which you can use to provide functionality to multiple HTML5 or web projects. If the HTML or Web categories are not available in your installation you can choose JavaScript File in the Other category in the New File wizard. If you want the JavaScript file to be visible in the Projects window, create the JavaScript file in an appropriate subfolder of the project, such as `public_html` or `src`.

The default template that is used for JavaScript files contains only licensing information. You can use the Templates Manager to modify the JavaScript template.

**To create a standalone JavaScript file:**

1. Select **File > New File > HTML5/JavaScript > JavaScript File**.
2. Enter the desired file name in the File Name field.
3. Click **Finish** to save the file.

**To edit the JavaScript File template:**

1. Select **Tools > Templates** in the main menu to open the Templates Manager.
2. Expand the **HTML5/JavaScript** node in the list of templates and select the **JavaScript File** template.
3. Click **Open in Editor** to edit the template in the source editor.

### 20.10.2 How to Edit a JavaScript File

You can edit JavaScript files in the source editor. The source editor provides standard features such as code completion and syntax coloring. The editor provides code highlighting for methods and variables, as well as global variables and properties of literal objects. When a JavaScript file is open in the source editor the Navigator window displays the structure of the JavaScript file and enables you to easily navigate to elements in the code. JavaScript editing features also work for JavaScript code embedded in PHP, JSP and HTML files.

The IDE provides some JavaScript code templates to help you to generate code. You can view the list of default JavaScript code templates and add your own code templates in the **Code Templates** tab in the **Editor** category in the **Options** window.

The IDE provides tools to help you with your JavaScript code. When you are editing your code a parser runs in the background and provides detailed warnings and hints to resolve potential problems. You can configure the JavaScript hints that are displayed in the **Options** window.

JavaScript code completion gives you a choice of the IDE's built-in JavaScript core classes to insert into your code and can show you the methods available for JavaScript strings. Code completion works with all the literal types in JavaScript. The type analysis and code completion machinery also knows about prototype style classes (regular functions only) and the operator for constructing them.

NetBeans IDE consults type parameters for function parameters and return types (@type, @param, and other JSDoc tags). These types are shown in code completion: in the list, in the documentation, and in parameter hints. Return types are also shown in the code completion dialog after the function name, separated by a colon. The IDE can determine the return type for many functions. For example, methods which return true or false have a Boolean return type, those returning literal numbers have a Number return type, and so on. The IDE both infers types and explicitly tracks declared types via comments. The most important implication is that the IDE follows types through calls.

#### To edit a JavaScript file:

- Expand the project node in the **Projects** window and double-click the JavaScript file to open the file in the editor.

The source editor supports versions 5.1 and 6 of ECMA Script and a few ECMA Script 7 features that are experimental.

#### To switch between different versions of the JavaScript language:

1. Right-click a project name in the **Projects** window.
2. Choose **Properties** from the context menu and **JavaScript** from the **Categories** list.
3. Expand the **ECMA Version** drop-down list and choose the required option and click **OK**.

The IDE displays different hints to let you know that the version of an expression or syntax is not supported by the chosen ECMAScript version.

#### To display comments in JSON files:

1. Right-click a project name in the **Projects** window.
2. Choose **Properties** from the context menu and **JavaScript** from the **Categories** list.
3. Select the **Allow comments in JSON files** checkbox and click **OK**.

---

**Note:** JSON files cannot contain comments according to the JSON specification. The IDE does not report comments as errors if the **Allow comments in JSON files** option is selected.

---

### 20.10.3 How to Debug a JavaScript File

You can use the IDE to debug JavaScript files in HTML5 applications. JavaScript debugging starts automatically when you run the application or unit tests in one of the browsers that support debugging. When you are debugging the application the following windows provide debugging details.

**Table 20–1    JavaScript Debugging Windows**

Window	Description
Breakpoints window	Displays a list of breakpoints in all open applications.

**Table 20–1 (Cont.) JavaScript Debugging Windows**

Window	Description
Call Stack window	Displays the current execution stack of a JavaScript program.
Variables window	Displays the variables and watches that are valid in the current scope. You can create a new watch from the Variables window. For details about creating watches, see <a href="#">Section 10.9.6.5, "How to Create a Watch."</a>
Browser Log	Displays the exceptions, errors and warnings that are generated by the browser.

When you run the application the debugger will pause execution of the application at the first breakpoint in the JavaScript that is encountered and the JavaScript file that contains the breakpoint opens in the editor. The IDE also automatically opens the JavaScript debugger windows. You can use the debugging buttons in the toolbar to step through the JavaScript files.

To debug JavaScript files in the IDE the IDE needs to be able to communicate with the debugger. To use the JavaScript debugger in the IDE, you need to select one of the following browser options as the target browser.

**Table 20–2 Browsers that support JavaScript Debugging**

Browser	Description
Embedded WebKit Browser	The IDE includes a browser that is based on WebKit. The embedded WebKit browser enables you to view and debug HTML5 applications in the IDE. If you select Embedded WebKit Browser as the browser in the Project Properties window the IDE will open the browser in a window in the IDE when you run the application and debugging is automatically enabled.
Chrome with NetBeans Connector	The IDE includes an extension that you can install in the Chrome browser that enables the IDE and the browser to communicate. If you select Chrome with NetBeans Connector as the browser in the Properties window, the IDE will open the Chrome browser when you run the application. Debugging is automatically enabled if the extension is installed when you run the application. If the extension is not installed, the IDE will prompt you to install the extension.

#### 20.10.4 How to Set JavaScript Breakpoints

To debug the JavaScript in your application in the IDE, you need to specify breakpoints that will pause execution of the scripts when they are hit. You can set the following JavaScript breakpoints in the IDE:

**Table 20–3 Types of JavaScript Breakpoints**

Breakpoint	Description
DOM	Use this breakpoint to debug changes that are made to the DOM tree.
Line	Use this breakpoint to debug specific lines of code in your JavaScript file.
Event	Use this breakpoint to debug actions in your application that are invoked by events.
XMLHttpRequest	Use this breakpoint to debug requests for URLs.

**To open the New Breakpoint dialog box:**

- Choose **Debug > New Breakpoint** from the main menu.

Alternatively, you can open the Breakpoints window by clicking the New Breakpoint icon (  ) in the left margin.

The steps for setting JavaScript breakpoints will depend upon the type of breakpoint.

**To set a DOM breakpoint:**

1. Open the HTML file in the editor.
2. Open the Browser DOM window, if not already open.
3. In the Browser DOM window, locate the object in the DOM tree that you want to debug.
4. Right-click the object and choose one of the DOM breakpoint properties in the popup menu.

**To set a line breakpoint:**

1. Open the JavaScript file in the editor.
2. Click in the left margin of the editor next to the line where you want to set the breakpoint.

The breakpoint icon (  ) will appear in the margin.

**To set an event breakpoint:**

1. Open a JavaScript file or HTML file in the editor.
2. Place the insert cursor in the file and choose **Debug > New Breakpoint** in the main menu to open the New Breakpoint dialog box.
3. Select JavaScript in the Debugger drop-down list and select Event in the Breakpoint Type drop-down list.
4. Select any actions that you want to trigger the breakpoint. Click **OK**.

**To set an XMLHttpRequest breakpoint:**

1. Open a JavaScript file or HTML file in the editor.
2. Place the insert cursor in the file and choose **Debug > New Breakpoint** in the main menu to open the New Breakpoint dialog box.
3. Select JavaScript in the Debugger drop-down list and select XMLHttpRequest in the Breakpoint Type drop-down list.
4. Specify the URL that will trigger the breakpoint when it is requested by a script. Click **OK**.

## 20.10.5 How to Run Unit Tests on JavaScript Files

You can configure the IDE to use Karma, JS Test Driver or Mocha to run the unit tests for your JavaScript files. To use the test runner you need to install the test runner and then configure the test runner to specify the browsers and testing framework that you want the test runner to use and the files to load when running the tests. After you configure the test runner for your application you can run the tests from the Projects window. You can view the results of the tests in the Output window of the IDE.

Debugging of unit tests is automatically enabled if you specify Chrome with NetBeans Connector as one of the JS Test Driver browsers when you configure JS Test Driver in the Services window.

**To run unit tests with Karma:**

1. Prior to installing Karma on your local system you need to download and install Node.js. For more information see  
<http://karma-runner.github.io/latest/intro/installation.html>.
2. Choose **File > New File** to open the New File wizard.
3. Select **Karma Configuration File** in the **Unit Tests** category in the New File wizard. Click **Next**.
4. Use the default name `karma.conf` for the File Name. Click **Finish**.

When you click **Finish** the IDE creates the `karma.conf.js` file and opens the file in the editor.

5. Edit `karma.conf.js` to specify the location of the JavaScript files to be included and excluded when you run the tests and Karma plugins that are required to run the tests with this configuration.
6. Right-click the project node in the **Projects** window and choose **Properties** in the popup menu.
7. Select **JavaScript Testing** category in the **Categories** pane of the **Project Properties** window.
8. Select **Karma** in the **Testing Provider** drop-down list.
9. To specify the location of your Karma configuration file, click **Search** and the IDE will find the default Karma configuration file. Alternatively, click **Browse** to manually locate a configuration file.
10. Click **OK**.

When you click **OK** a Karma node appears under the project node in the **Projects** window.

11. To start the Karma server, right-click the Karma node in the **Projects** window and choose **Start** in the popup menu.
12. (Applies if Karma script is not automatically found by the IDE and the path to Karma needs to be set manually.) In the **Options: HTML/JS: Karma** window specify the path to the Karma executable file. Click **OK**.

When you click **OK** the Karma server starts.

13. Right-click the project node in the **Projects** window and choose **Test** in the popup menu.

The IDE runs the unit tests. You can open the **Test Results** window to view the results. You can see details about the specific tests in the **Output** window.

**To run unit tests with JS Test Driver:**

1. Download the `js-test-driver` JAR to your local system.

---

**Note:** To use JS Test Driver in the IDE you need to download the JS Test Driver JAR that is available from the following location:

<http://code.google.com/p/js-test-driver/>

The JS Test Driver site also provides documentation and details about using JS Test Driver.

---

2. In the Services window, right-click the JS Test Driver node and choose **Configure** in the popup menu.
3. In the Configure JSTestDriver dialog box, click Browse and locate the js-test-driver JAR that you downloaded.
4. Select the browsers that you want to run the tests against. Click OK.  
Only browsers that are installed on your local system are displayed in the list.
5. Right-click the JS Test Driver node and choose Start to start the js-test-driver server.

When you choose Start, each of the browsers that you specified will open a window and will wait to run the unit tests. You can see the status in the js-test-driver server tab in the Output window.

6. Choose **New > New File** to open the New File wizard.
7. Select **JsTestDriver Configuration File** in the **Unit Tests** category in the New File wizard.
8. Use the default name jsTestDriver for the File Name. Click **Finish**.  
When you click Finish the IDE creates the jsTestDriver.conf file and opens the file in the editor.
9. Edit jsTestDriver.conf to specify the location of the JavaScript source files and test files.
10. Right-click the project node and choose **Test** in the popup menu.
11. In the **Select Testing Provider** dialog choose **JS Test Driver** from the drop-down list and click **OK**.

When you click **OK** the IDE runs the unit tests. You can open the **Test Results** window to view the results. You can see details about the specific tests in the **Output** window.

#### To run unit tests with Mocha:

1. Prior to installing Mocha on your local system you need to download and install Node.js. For more information see <https://mochajs.org/#installation>.
2. Right-click the project node in the **Projects** window and choose **Properties** in the popup menu.
3. Select **JavaScript Testing** category in the **Categories** pane of the **Project Properties** window.
4. Select **Mocha** in the **Testing Provider** drop-down list.
5. To specify the location of Mocha install location, click **Browse** to manually locate a installation directory. Alternatively, if the location automatically found by the IDE is correct, click **OK** to accept it.

6. Click **OK**.
7. Right-click the project node in the **Projects** window and choose **Test** in the popup menu.

When you click **OK** the IDE runs the unit tests. You can open the **Test Results** window to view the results. You can see details about the specific tests in the **Output** window.

## 20.10.6 How to Add a JavaScript Library to a Project

If you are using a JavaScript library that will provide specific functionality in your project you can add JavaScript libraries to a project in the Project Properties window.

### To add the npm library to an HTML5/JavaScript application:

1. Download and install Node.js from <https://nodejs.org>.
2. Open the selected project in the IDE.
3. Right-click the project node in the **Projects** window and choose **Properties**.
4. Select the **JavaScript Libraries: npm** category in the **Project Properties** window.
5. Select a required tab to add a library from the list of available tabs and click **Add**.

---

**Note:** Choose the **Regular** tab for npm dependencies that the project requires when it is run, the **Development** tab for npm dependencies that are not required for the execution of the project (for example, testing tools), and the **Optional** tab for optional npm dependencies.

---

6. In the **Add npm package** dialog type in the name of an npm dependency and click **Search**.
7. When the dependency is found click **Add**.  
The npm dependency is added to the project.
8. Click **OK** to confirm the use of the added npm library.

### To add the Bower library to an HTML5/JavaScript application:

---

**Note:** Before installing Bower you need to have Node.js and Git installed on your machine.

---

1. Download and install Node.js from <https://nodejs.org>
2. Download and install Git from <https://git-scm.com>
3. Run `npm install -g bower` from your command line.
4. Open the selected project in the IDE.
5. Right-click the project node in the **Projects** window and choose **Properties**.
6. Select the **JavaScript Libraries: Bower** category in the **Project Properties** window.
7. Select a required tab to add a library from the list of available tabs and click **Add**.

---

**Note:** Choose the **Regular** tab for Bower dependencies that the project requires when it is run or the **Development** tab for Bower dependencies that are not required for the execution of the project (for example, testing tools).

---

8. In the **Add Bower package** dialog type in the name of a Bower dependency and click **Search**.
9. When the dependency is found click **Add**.  
The Bower dependency is added to the project.
10. Click **OK** to confirm the use of the added Bower library.

**To add the CDNJS library to an HTML5/JavaScript application:**

1. Open the selected project in the IDE.
2. Right-click the project node in the **Projects** window and choose **Properties**.
3. Select the **JavaScript Libraries: CDNJS** category in the **Project Properties** window.
4. Click **Add**.
5. In the **Add CDNJS Library** dialog type the name of the library and click **Search**.
6. When the library is found click **Add**.  
The CDNJS library is added to the project.
7. Click **OK** to confirm the use of the added CDNJS library.

## 20.11 Using Grunt and Gulp Build Tools

NetBeans IDE provides support for Grunt and Gulp JavaScript task runners. The support involves:

- running tasks via the context menu of the project
- running tasks from the Navigator window
- specifying custom tasks in the Manage Tasks window

### 20.11.1 Installing Grunt.js

To use Grunt.js in a project, you must install it on your local system globally for executing Grunt commands and locally for building a project tasks tree. The IDE will prompt you to install Grunt if your Grunt installation is not detected when you add support for Grunt to an application. You can find the installation instructions and additional details about the platform at <http://gruntjs.com/getting-started>.

### 20.11.2 Installing Gulp.js

To use Gulp.js in a project, you must install it on your local system globally for executing Gulp commands and locally for building a project tasks tree. The IDE will prompt you to install Gulp if your Gulp installation is not detected when you add support for Gulp to an application. You can find the installation instructions and additional details about the platform at <http://gulpjs.com/>.

### 20.11.3 Running Grunt Tasks

To run Grunt tasks in a project, do one of the following:

- Right-click the project name in the **Projects** window and choose **Grunt Tasks > default** (or **build** or **test** as appropriate) from the context menu.
- Choose **Window > Navigator** to open the Navigator window and select the required Grunt task.

**To specify a custom Grunt task:**

1. Right-click your project name and choose **Grunt Tasks > Manage Task(s)**.
2. In the **Manage Task(s)** dialog box, enter the details of your custom task.
3. Click **OK** to save the new custom task.

You can run custom tasks like **default** ("built-in") tasks from the Grunt tool.

For example, if a project has a **build** task, you can name the custom task as **build (dev)** and specify it as follows: `build --env=dev`. The newly created **build (dev)** task becomes available for running in both the context menu and in the Navigator window.

### 20.11.4 Running Gulp Tasks

To run Gulp tasks, do one of the following:

- Right-click the project name in the **Projects** window and choose **Gulp Tasks > default** (or **build** or **test** as appropriate) from the context menu.
- Choose **Window > Navigator** to open the Navigator window and select the required Gulp task.

**To specify a custom Gulp task:**

1. Right-click your project name and choose **Gulp Tasks > Manage Task(s)**.
2. In the **Manage Task(s)** dialog box, enter the details of your custom task.
3. Click **OK** to save the new custom task.

You can run custom tasks like **default** ("built-in") tasks from the Gulp tool.

For example, if a project has a **build** task, you can name the custom task as **build (dev)** and specify it as follows: `build --env=dev`. The newly created **build (dev)** task becomes available for running in both the context menu and in the Navigator window.

# 21

---

## Developing PHP Applications

This chapter describes developing PHP applications for NetBeans IDE, including editing, debugging, and using continuous build servers.

This chapter contains the following sections:

- [About Developing PHP Applications](#)
- [Working with PHP Applications](#)
- [Editing PHP Files](#)
- [Debugging PHP Applications](#)
- [Testing PHP Applications](#)
- [Using a Continuous Build Server With PHP](#)

### 21.1 About Developing PHP Applications

NetBeans IDE (Integrated Developer Environment) for PHP provides tools for PHP programmers. The core of NetBeans PHP support is the PHP Editor, which includes code completion and other programming aids. NetBeans IDE for PHP can also be integrated with the Xdebug debugger, PHPUnit tester, and PHPDocumentor documentation generator.

Please note that the NetBeans IDE is designed to be as lightweight as possible. It does not contain visual design tools or other heavy-duty features.

#### 21.1.1 Databases

The IDE supports working with databases. Open the Services window and register your database by right-clicking the Databases node. You may need to add the path to your database driver as well. NetBeans IDE provides a tree view of database contents and an SQL editor. For more information, see [Section 24.3, "Setting up a Database Connection."](#)

#### 21.1.2 Remote Development

The IDE includes support for Subversion, Mercurial, and Git versioning systems. You can also add CVS support as a plugin. For more information, see [Section 3.1, "About Versioning Applications with Version Control."](#)

The IDE also provides basic support for FTP/SFTP, sufficient for a lone developer to work on a simple project. If you are working on a complicated project or with multiple developers, you should use a version control system. If you must use FTP/SFTP with

a complicated project, consider using the IDE in combination with a full-featured FTP client.

### 21.1.3 Frameworks

The IDE supports some PHP frameworks. You can configure the global options for the framework in the **Frameworks & Tools** tab of the PHP category in the **Options** window.

You can add a framework to a project when you create the project by selecting the framework in the last page of the **New PHP Project** wizard. If a project includes framework support, that framework's commands are available in the project's context menu in the Projects window.

## 21.2 Working with PHP Applications

Before you begin to work with PHP applications in the IDE you need to install and configure your PHP environment. The PHP environment includes:

- A PHP engine
- A web server, such as Apache
- Optionally, a database, such as MySQL or Oracle Database
- Optionally, a debugger supported by the IDE

### 21.2.1 How to Create a PHP Project

A project is a group of source files and the settings with which you run and debug those source files. In the IDE, PHP development takes place within a project, though you can also edit, run and debug a PHP file that is not in a project.

The IDE provides a wizard that enables you to create a new empty PHP project or to create a PHP project from existing sources that are located on your local system or on a remote server. The IDE provides basic support for FTP and SFTP, however, it is preferable to use version control with remote projects and use a version control system to commit your changes to the remote server.

By default the IDE places all project files in your web server's public folder (such as /htdocs). The IDE reads the location of your web server from your PHP environment. However, you have other options for file locations. These options appear in the New Project wizard:

- Have the NetBeans metadata stored in another folder.
- Copy the source files to another location. With this option, you could keep the PHP project in one location (such as the same folder as your Java projects) and the IDE could copy it to the public folder of your web server.

#### To create a new empty project:

1. Open the New Project wizard and select the PHP project category.
2. Select the PHP Application project template. Click Next.
3. Specify the project name and location of the project and files. Click Next.
4. Select an option in the Run As drop-down list to create the default run configuration for the project and specify any required details. Click Next

You can modify the run configuration and create new run configurations in the Project Properties window.

5. Select any PHP frameworks that you want to use in your project

You can add support for PHP frameworks and Composer by installing plugins in the Plugins Manager. If you are using Composer you can click Next to provide Composer details.

6. Click Finish.

**To create a project from sources on a remote server:**

1. Open the New Project wizard and select the PHP project category.
2. Select PHP Application from Remote Server as the project type. Click Next.
3. Specify the project name and location for the local sources. Click Next.
4. Specify the URL of the remote server.
5. Select a connection from the Remote Connection drop-down list

If the remote connection does not exist, click Manage to open the Manage Remote Connections dialog box where you can create, edit and delete remote connections. The remote connections that you create are also available when you create a run configuration that targets a remote web server.

6. Specify the directory on the remote server that contains the sources. Click Next.  
When you click Next the IDE attempts to connect to the remote server.
7. Click Finish.

### 21.2.2 How to Create a Run Configuration for the Project

Every NetBeans PHP project has at least one run configuration. You set this default configuration when you create the project. You can add as many additional run configurations as you want in the Project Properties window. In a run configuration, you can select any of the following Run As options:

- As a local web site, on your local web server (default)
- As a remote web site, over FTP/SFTP. You can also set up remote synchronization.
- As a command-line script

You can switch between run configurations at any time by right-clicking the project's node and selecting Set Configuration and the configuration in the popup menu.

**To create a run configuration:**

1. Open the Project Properties window and select the Run Configuration category in the left pane.

Alternatively, right-click the project node in the Projects window and choose Set Configuration > Customize in the popup menu.

2. Click New next to the Configuration drop-down list and type a name for the new run configuration. This name will be used in the Set Configuration popup menu to identify the configuration.
3. Select a Run As option from the drop-down list.
4. Specify any options and details required by the Run As option that you select.

The required details for the run configuration will change according to the Run As option that you select. For example, if you select Remote Web Server you must provide the connection details for the remote server.

5. Click OK.

### 21.2.3 How to Synchronize Local and Remote Sources

You can synchronize the files on your local file system with the files that are located on a remote server. To synchronize the contents of a local folder with the contents of a folder on a remote server you must set the run configuration to a configuration where the Run As option is set to Remote Web Site and you must be able to connect to the remote server with the specified connection details.

Remote Synchronization is a feature of PHP projects set to run as Remote Web Site, over FTP or SFTP. Remote Synchronization allows you to compare your local version of a project with the version on a server and to diff, upload, or download files so the local and remote versions of the project match. The Remote Synchronization dialog box opens from the popup menu (right click) of the Source Files folder of a PHP project.

The IDE can perform one of several operations when synchronizing a file. The table in the Remote Synchronization dialog box displays the operations that the IDE will perform and the files that will be affected during synchronization.

**To synchronize local and remote sources:**

1. Set the run configuration of your project to a configuration where the Run As option is set to Remote Web Server.
2. Right-click the Source Files node in the Projects window and choose Synchronize in popup menu.
3. Select the operations to perform during the synchronization.
4. Modify the operations for specific files by selecting the files in the table and using the buttons below the table to modify the operations for the selected files.
5. Click Synchronize.

### 21.2.4 How to Run a Project

Every NetBeans PHP project has at least one default run configuration that you set when you create the project. When you run the PHP application the IDE performs actions that are based on the current run configuration. You can switch between run configurations to easily change the actions that the IDE performs when you run the application. For example, you can create one run configuration that deploys the application to your local server and a second run configuration that deploys to a remote staging server. You can switch between run configurations at any time by right-clicking the project's node and selecting Set Configuration and the configuration in the popup menu.

**To run a PHP application:**

1. In the Projects window, right-click the project node and select **Set Configuration > <default>** in the popup menu.

The <default> configuration is the configuration that you created when you created project. Any other run configurations that you created are also listed in the Set Configuration sub-menu.

2. Right-click the project node in the Projects window and choose Run.

Depending on your run configuration, the IDE might open a web browser window to display a page in your project or run your project as a script using a PHP interpreter. If the project is configured to run as a command-line script, you have to run the application from the command line.

### 21.2.5 How to Test a PHP Project

You can use PHPUnit and Selenium to test a PHP project.

#### To test a PHP project:

1. Open the Project Properties window and select the Run Configuration category in the left pane

Alternatively, right-click the project node in the Projects window and choose **Set Configuration > Customize** in the popup menu.

2. Click New next to the Configuration drop-down list and type a name for the new run configuration. This name will be used in the Set Configuration popup menu to identify the configuration.
3. Select a Run As option from the drop-down list.
4. Specify any options and details required by the Run As option that you select.

The required details for the run configuration will change according to the Run As option that you select. For example, if you select Remote Web Server you must provide the connection details for the remote server.

5. Click OK.

## 21.3 Editing PHP Files

The IDE contains a number of features that simplify editing PHP files, including syntax highlighting, code completion, using templates, and more. This section describes these features.

### 21.3.1 How to Use Syntax Highlighting

The editor provides syntax highlighting for PHP, HTML, JavaScript, and CSS code blocks.

The following syntax elements are highlighted in the current version:

- PHP keywords
- PHP variables
- PHP constants
- HTML tags
- HTML input form attributes

The current line is displayed with a light-blue background. By double-clicking a PHP variable, a function, or a class, all the occurrences of the variable are highlighted in olive green. Lines where errors have been detected are underlined in red.

To change the settings for highlighting, choose **Tools > Options** and switch to the Fonts&Colors tab. Specify the settings of your choice.

### 21.3.2 How to Use Code Completion

Code Completion is a common name for a set of features that facilitate and speed up the process of coding.

The following types of code completion are provided:

- Snippets
- Context-sensitive proposals
- Abbreviations
- Code Completion in Constructors
- SQL Code Completion
- PHP Namespaces
- Overridden and Implemented Methods
- Annotations

#### 21.3.2.1 Snippets

Snippets enable the user to generate code for various elements automatically.

1. Choose **Tools > Palette > HTML/JSP Code Clips**. A palette containing various user interface elements appears in the right-hand panel.
2. Drag the required icon on the palette to the relevant position in the code. A dialog box for specifying the parameters of the corresponding elements appears. Fill in the data.
3. The code that displays the chosen element is generated and inserted in the chosen location.

#### 21.3.2.2 Context-Sensitive Proposals

The editor provides context-sensitive proposals for completing any number of starting symbols of the following:

- A PHP keyword (for example, if, else, elseif, while, switch, function, and so on)
- A PHP built-in function (for example, substr, count, and so on)
- A pre-defined or user-defined variable

The editor not only suggests expansions but also provides parameter hints.

#### To apply Code Completion:

1. Type the starting symbols of the required character string.
2. Press Ctrl+Space. A drop-down list shows the context-sensitive proposals. Each proposal is supplied with a description and parameter hints. The contents of the list change as you continue typing.
3. To obtain a list of the PHP key words that are available in the current context, press Ctrl+Space without any previous typing.
4. To obtain a hint on a variable, type the dollar symbol "\$". A list of all the currently available local and global variables appears.

### 21.3.2.3 Code Templates and Abbreviations

In the current context, the term "abbreviations" refers to a set of predefined character strings that correspond to the key words used in a programming language. Each abbreviation is associated with an expanded text which contains the full key word and a code template for the key word with parameter hints. To apply this functionality, type an abbreviation and press Tab. The abbreviation is replaced with the corresponding key word and the code template for the key word is provided.

#### To view the list of defined abbreviations with code templates:

1. Choose Tools > Options > Editor > Code Templates.
2. From the Language drop down list, select PHP. The list of PHP abbreviations and code template defined for them is displayed.
3. To add or remove a definition from the list, use the **New** or **Remove** buttons respectively.
4. To edit a definition, select the relevant row and edit the text in the edit field below the list.

For more information, see the following entry on the PHP blog:

<https://netbeans.org/kb/docs/php/code-templates.html>.

### 21.3.2.4 Code Completion for Constructors

After the new keyword, the code completion window is displayed with constructors and parameters for all available classes in the project.

### 21.3.2.5 SQL Code Completion

SQL code completion displays when a string begins with the SQL keyword "select". The first step is to select the database connection.

All database connections registered with the IDE are displayed. After you select the connection, SQL code completion offers all tables from that database connection. If the table has columns, those are displayed as well.

SQL code completion also works with table aliases.

### 21.3.2.6 PHP Namespaces

Code completion supports fully qualified, partially qualified, and unqualified namespace names. The IDE also helps you resolve missing namespace use statements. Inside a namespace where you want to fix missing use statements, either right-click and select Fix Uses..., or press Ctrl+Shift+I and go to Source > Fix Uses... A dialog opens offering fully qualified names for each needed use statement. For more information, see the following entry on the PHP blog:

[https://blogs.oracle.com/netbeansphp/entry/how\\_to\\_fix\\_your\\_use](https://blogs.oracle.com/netbeansphp/entry/how_to_fix_your_use)

### 21.3.2.7 Overridden and Implemented Methods

Code completion between class members offers to override or implement methods.

### 21.3.2.8 Annotations

NetBeans IDE code completion supports the following types of PHP annotations:

- ApiGen (legacy PHPDoc annotations)
- PHPUnit

- Doctrine 2 (ORM and ODM)
- Symfony 2 and 3

Every annotation can be associated with a context. NetBeans IDE recognizes four contexts:

- function
- class/interface (type)
- method
- field

You can add more annotations to code completion by choosing:

- **Tools > Options > PHP > Annotations** for globally available annotations
- **Project Properties > Annotations** for project specific annotations

### 21.3.3 How to Use Bracket Completion

The editor automatically adds and removes matching brackets and quotes while the user is typing the code.

- Paired single (' ') and double quotes (" " "), braces (( )) and brackets ([ ]) are added when the user has typed the first symbol.
- At the end of a line, a semicolon is added in addition to the closing single or double quote.
- The closing curly brace is added after the user presses Enter, which also activates the Smart Indent function.
- Deleting the opening single or double quote, brace, or bracket causes automatic removal of the corresponding closing symbol but does not affect the line end semicolon.
- When the cursor points at a closing brace, a closing curly brace or a closing bracket, the paired opening symbol is highlighted in yellow.

### 21.3.4 How to Use Parameter Hints

The editor prompts the user regarding the formal parameters of a function or a method in the context where the function or method is called.

#### To use parameter hints:

1. Type the starting characters of the function you want to call.
2. Press Ctrl+Space. A drop-down list shows the context-sensitive proposals with the formal parameters for each proposal.
3. Choose the relevant proposal and press Enter. The name of the chosen function is inserted in the code and a template for entering the parameters is shown in brackets.

### 21.3.5 How to Use the Go To Declaration Feature

The Go To Declaration function navigates the user from an occurrence of a variable to the line where the variable is declared or initialized. To use this functionality, position the cursor on the relevant variable occurrence and choose **Navigate > Go to Declaration** from the context menu, or press Ctrl+B.

### 21.3.6 How to Use Rename Refactoring and Instant Rename

Rename Refactoring lets you rename an element such as a class name across all files in a project. The feature forces you to preview your changes before you can make them. The preview window shows you every location of the element and lets you exclude individual occurrences of the element from being renamed.

Instant Rename is a simpler function that only works in "non-public" contexts, such as renaming a variable inside a method, or renaming private variables and fields. Instant Rename lets you only rename an element within a file and does not provide a preview window.

To use Instant Rename, place the cursor on a name you want to change and press Ctrl+R. If Instant Rename applies to that variable, all instances of that variable or function name are highlighted. Change one instance of the name and all other instances in the file are changed simultaneously.

To use Rename Refactoring, select the element you want to rename and either press Ctrl+R or right-click and select **Refactor > Rename**. A dialog opens for you to rename the element. Rename the element and press Preview. The Refactoring window opens. In this window, you can find every instance of the element in your project and decide whether or not to rename it.

### 21.3.7 How to Use Code Generators

When you press the combination Alt+Insert (Ctrl+I on Mac), a menu opens with all possible code generators. The list of generators is context sensitive. It depends on the position of the caret in the code when the key combination is pressed. Depending on your position in the code, you can generate a database connection, database tables, lorum ipsum text, and several others. This section describes only the following code generators:

- Constructors
- Getters and Setters
- Overridden and Implemented Methods

#### 21.3.7.1 Constructors

You can generate constructors by pressing Alt+Insert (Ctrl+I on Mac) when the caret is inside a class body, but not inside any functions contained in that body. When you select **Generate... Constructor**, a dialog opens listing the fields you can initialize by the constructor. The field names are used as parameters of the constructor.

You can decide not to select any fields. In this case, the IDE generates an empty constructor, with no parameters. If the field is a class with properties, you can either select individual properties, or you can select the class, in which case all the class' properties are selected automatically.

For more information, see the following NetBeans PHP blog post:  
[https://blogs.oracle.com/netbeansphp/entry/generate\\_constructor\\_getters\\_and\\_setters](https://blogs.oracle.com/netbeansphp/entry/generate_constructor_getters_and_setters).

#### 21.3.7.2 Getters and Setters

You can generate getters and setters by pressing Alt+Insert (Ctrl+I on Mac) when the caret is inside a class body and selecting Getter, Setter, or Getters and Setters. Only the possible functions are displayed. For example, if you already have setters for the available properties, only the getter option appears.

When you select **Generate... Getter/Setter/Getter and Setter**, a dialog appears with the properties for which you can generate a getter or setter. The properties are displayed in a tree. If you select a parent class, you automatically select all that class' properties.

You can name a getter or setter according to either the convention `getName` or the convention `get_name`.

For more information, see the following NetBeans PHP blog post:  
[https://blogs.oracle.com/netbeansphp/entry/generate\\_constructor\\_getters\\_and\\_setters](https://blogs.oracle.com/netbeansphp/entry/generate_constructor_getters_and_setters).

#### 21.3.7.3 Overridden and Implemented Methods

You can generate overridden or implemented methods by pressing Alt+Insert (Ctrl+I on Mac) when the caret is inside a class declaration and there are multiple class members. A dialog opens showing the methods you can insert and indicating whether they are overridden or implemented.

This feature complements the "Implement all abstract methods" hint and code completion for overridden and implemented methods.

For more information, see the following NetBeans PHP blog post:  
[https://blogs.oracle.com/netbeansphp/entry/generate\\_overriden\\_implemented\\_methods](https://blogs.oracle.com/netbeansphp/entry/generate_overriden_implemented_methods)

#### 21.3.8 How to Define Variable Type in Comments

You can define a variable and its type in a comment. The comment has to be in the format `/* @var $variable type */`. If the comment is written correctly, the `var` tag is in bold font.

You can use the code template `vdoc`, followed by Tab, to generate a comment that defines a variable. The variable name is selected, and you can change it. Then press Tab again, and the type is selected.

The code template automatically sets the variable name and type. If a variable is used after the place where you insert the template, then that following variable name is suggested by default. If there is not any variable used after the place where you insert the template, but there is a variable used above the template, then that preceding variable name is suggested as default. If the IDE is not able locate any variable near where you use the template, then the default name is variable. The variable type is set automatically according to the same rules.

For more information, see the following screencast:

<https://netbeans.org/kb/docs/php/php-variables-screencast.html>

#### 21.3.9 About PHP 7 Support

NetBeans Editor for PHP offers a number of features specific to developing with PHP 7, discussed in detail in the following blog entries:

- Return type declarations, scalar types, combined comparison (spaceship), null coalesce operators, generator delegation:  
[https://blogs.oracle.com/netbeansphp/entry/php\\_7\\_support\\_part\\_1](https://blogs.oracle.com/netbeansphp/entry/php_7_support_part_1)
- Group use declarations, anonymous classes:  
[https://blogs.oracle.com/netbeansphp/entry/php\\_7\\_support\\_part\\_2](https://blogs.oracle.com/netbeansphp/entry/php_7_support_part_2)

## 21.4 Debugging PHP Applications

Debugging is the process of examining your application for errors. The process of debugging is accomplished by setting breakpoints and watches in your code and running it in the debugger. This enables you to execute your code one line at a time or breakpoint-to-breakpoint and examine the state of your application in order to discover any problems.

When you start a PHP debugging session, a browser window opens. By default the application stops at the first line of the program. You must switch back to the IDE to continue the debugging session. You switch back and forth between the IDE and your browser throughout your debugging session.

Additional debugger windows also appear automatically at the bottom of your screen. These windows help you keep track of breakpoints, variables, sources, the call stack, and threads.

You can also debug applications that are running on a remote machine by setting up path mapping to a custom URL.

For more details see the following document:

<https://netbeans.org/kb/docs/php/debugging.html>

### 21.4.1 How to Debug a PHP Project

The following is a typical work flow for debugging a PHP project.

**To debug a PHP project:**

1. Set up XDebug on your local system.
2. Set breakpoints and watches in your code.
3. Start the debugging session.
4. Step through your code.
5. Finish the debugging session.

The following sections show in greater detail the steps for setting up XDebug and debugging your project.

### 21.4.2 How to Set Up XDebug

The IDE supports the XDebug third-party PHP debugger. If XDebug is installed correctly and the PHP environment is configured correctly, you can debug PHP projects inside the IDE. You can also debug PHP files remotely if you set up path mapping.

When you run XDebug from NetBeans IDE, PHP program execution pauses at every line where you set a breakpoint. When the program execution is paused, XDebug can retrieve information about the current program state, such as the values of the program variables.

Additional information can be found in the following wiki document:

<http://wiki.netbeans.org/HowToConfigureXDebug>

#### 21.4.2.1 Check If XDebug Is Installed

The first step you should take is to see if XDebug is already installed on your system. Some AMP packages come with XDebug. Run `phpinfo()`. If there is an XDebug section in your PHP info, XDebug is installed. If XDebug is installed, see

[Section 21.4.2.3, "Setting Up XDebug."](#)

### 21.4.2.2 Install XDebug

The NetBeans IDE for PHP community consensus is that the XDebug installation wizard should be used to install XDebug.

1. Run `phpinfo()`.
2. Copy the results of `phpinfo()` from your browser into the clipboard.
3. In a browser window, open the following URL: <http://xdebug.org/wizard.php>
4. Paste your PHP info into the XDebug wizard form and click **Analyze**.
5. Follow the instructions that appear.

### 21.4.2.3 Setting Up XDebug

The following section assumes you have XDebug installed and you need to set it up.

1. Run `phpinfo()`.
2. In the PHP info, find the entries for Loaded Configuration File and Additional .ini Files Parsed.
3. For each `php.ini` or `xdebug.ini` file you find, open that file and perform the following steps:

1. Make sure the file contains the following settings:

```
zend_extension=[path to xdebug .so or .dll]
xdebug.remote_enable=1
xdebug.remote_handler=dbgp
xdebug.remote_mode=req
xdebug.remote_host=127.0.0.1
xdebug.remote_port=9000
```

These settings are usually in a section labeled [XDebug]. If such a section does not exist, create one. However, some AMPS may differ. For example, in some versions of XAMPP AMP, the line `zend_extension = "G:\xampp\php\ext\php_xdebug.dll"` is outside of the [XDebug] section, and the name of the .dll file is nonstandard (`php_xdebug.dll` instead of `xdebug.dll`).

2. Comment out all entries for Zend Optimizer, Zend Studio Debugger, or other Zend extensions.

### 21.4.2.4 Testing XDebug

Before trying out XDebug from the IDE you should test your XDebug installation from the command-line. One way to test XDebug is to write a test in PHP and run a debug session on it.

1. You need an arbitrary PHP file. If you do not have one, create a file called `index.php`, put any code into it and save it.
2. Using a text editor, create a file called `dbgttest.php` and type or paste the following code into the file (save the file afterward):

```
<?php
$address = '127.0.0.1';
$port = '9000';
$sock = socket_create(AF_INET, SOCK_STREAM, 0);
```

```

socket_bind($sock, $address, $port) or die();
socket_listen($sock);
$client = socket_accept($sock);
echo "Connection established: $client";
socket_close($client);
socket_close($sock);
?>

```

3. Run the code from the command line (for example, `/usr/bin/php dbgtest.php`).
4. Start a debug session for an arbitrary PHP file from within a browser window (for example, `http://localhost/index.php?XDEBUG_SESSION_START=mysession`).

If XDebug is properly configured, the script started in step 3 should print a message similar to "Connection established: Resource id #5". If no message is printed and the script is still running, XDebug is not properly configured or uses a different port or has another problem. In this case, kill the running process and investigate where the problem is.

#### 21.4.2.5 Setting Up Debugging Options in NetBeans IDE

After you install, configure, and successfully test XDebug, you can set up XDebug options in the IDE. To learn more about these options, open **Tools > Options** (NetBeans Preferences on Mac), go to the PHP category, select the **Debugging** tab, and click **Help**.

### 21.4.3 How to Set PHP Breakpoints in the IDE

A breakpoint is a flag in the source code that tells the debugger to stop execution of the program. When your program stops on a breakpoint, you can perform actions like examining the value of variables and single-stepping through your program.

#### 21.4.3.1 Setting Breakpoints

To set a breakpoint for a line in your PHP code, left-click the left margin or line number of the line. To remove the breakpoint, left-click the breakpoint icon.

To temporarily disable a breakpoint, right-click a breakpoint badge and deselect **Breakpoint > Enabled** in the popup menu. This toggles the breakpoint into a disabled state and the breakpoint badge in the left margin is now grey.

The Source Editor indicates a line breakpoint by highlighting the line at which the breakpoint is set in red and placing an annotation in the left margin. Users of other languages supported in the IDE may be familiar with other types of breakpoints but only line breakpoints are supported in PHP projects. For a table of breakpoint annotations that are used in the editor, see [Section 10.9.4, "Managing Breakpoints."](#)

You can view and organize all IDE breakpoints by choosing **Windows > Debugging > Breakpoints** (Alt+Shift+5). For details on organizing breakpoints, see [Section 10.9.4.3, "How to Organize Breakpoints Into a Group."](#)

### 21.4.4 How to Set the Current Context in the PHP Debugger

The current context is the portion of your program on which the debugger is currently focusing. When multiple sessions are running, only one session is current. Within the current session, the thread from which the debugger regained control is the default current thread. Inside the current thread, the most recent call is the default current call.

You can make any session, thread, or call current by right-clicking its node in the appropriate debugger window and choosing **Make Current**.

#### 21.4.4.1 Debugger Windows and Context

Most debugger windows depend on the current context. When you change the current context, the contents of these windows are updated to reflect the new context.

For example, the Debugging window shows the threads in the current session, while the Call Stack window shows the call stack for the current thread. The Variables window shows the variables that are local to the current call.

The exceptions are the Breakpoints and Watches windows. These windows list all breakpoints and watches set in the IDE. While the set of watches is shared by all sessions, an individual watch expression is evaluated and displayed based on the current context.

#### 21.4.4.2 The Source Editor and Context

When a variable is active in the current context, the Source Editor displays the value of the variable when you move the pointer over it. In cases where a program includes different variables with the same name, the Source Editor displays the value based on the current context, and not on the instance of the variable in the source code.

### 21.4.5 How to Start a PHP Debugging Session

PHP debugging sessions use debugger windows inside the IDE. You can debug projects, or you can remotely debug Web pages.

#### 21.4.5.1 Starting a PHP Debugging Session

The IDE starts the debugger, then runs the application inside the debugger. When you start a debugging session, the IDE automatically opens the debugger windows, opens a browser window, and prints debugger output to the Output window. You start the debugging session in the browser window.

By default, the debugging session stops at the first line of the PHP code. You can disable stopping at the first line in the Debugging tab in the PHP category of the Options window. When the session stops, you have a number of options for stepping through/into or continuing the debugging session.

##### To start debugging a project:

1. Locate the project's node in the Projects window.
2. Either right-click the project node and select Debug, or select the project node and run the Debug Project command (Ctrl+F5).

Alternatively, you can debug an individual PHP file:

1. Expand the project's node in the Projects window. Navigate the tree until you locate the file you want.
2. Either right-click the file node and select Debug, or select the file node and run the Debug File command (Ctrl+Shift+F5).

The process for starting a debugging session is the same for local and remote projects. However, path mapping must be set up to debug remote projects.

#### 21.4.5.2 Remote PHP Debugging and Path Mapping

It is possible to debug both scripts and web pages, and web pages can be debugged either locally or remotely. For Remote Debugging, unfortunately the debugged PHP file on the remote server is not the same as the file opened in the IDE running on a local machine. Debugger support in the IDE must thus be able to map server paths to

local paths. However, due to many complications, path mapping cannot be resolved automatically for every individual scenario. Therefore, you must manually define path mapping through the project setup for individual run configurations. You can also specify the proxy server, if any, and the URL at which the debugging session starts. If you do not specify this URL, debugging starts at your index file.

#### To set up path mapping and enable custom debugging URLs:

1. Right-click the project's node in the Projects window and open the project's Properties from the context menu.
2. In the Project Properties dialog, go to the Run Configuration category.
3. Click the Advanced button. The Advanced Web Configuration dialog opens.
4. Add the server path and the project path for path mapping.
5. Under "Debug URL", select one of the following (do not leave the default selected):
  - Ask Every Time, which has the IDE prompt you for the URL when you start a debugging session.
  - Do Not Open Web Browser, which requires you to open the browser and enter the URL manually (you need the GET/POST XDEBUG\_SESSION\_START variable).
6. If you are using a proxy server for debugging, enter the server's host name and port in the Debugger Proxy section.

For more information, please see the following post in the NetBeans for PHP blog:

[https://blogs.oracle.com/netbeansphp/entry/path\\_mapping\\_in\\_php\\_debugger](https://blogs.oracle.com/netbeansphp/entry/path_mapping_in_php_debugger)

#### 21.4.5.3 Stepping Through Your Program

By default, the debugging session stops at the first line of the PHP code. You can disable stopping at the first line in the Debugging tab in the PHP category of the Options window. When the session stops, you can step through your lines of code using the following commands on the Debug menu or toolbar:

You should set breakpoints before you start a debugging session or you will have to step over/into/out of your entire application. If you disable Stop at First Line and do not set any breakpoints, your application never stops during debugging and you get no useful results.

Command	Shortcut	Description
Debug > Continue	F5	Runs the program until it reaches the next breakpoint or until the program terminates normally.
Debug > Step Over	F8	Executes one source line. If the source line contains a call, executes the entire routine without stepping through the individual instructions.
Debug > Step Into	F7	Runs the program to the next line and pauses execution before any changes have been made to the state of the program.
Debug > Step Out	Ctrl+F7	Executes one source line. If the source line is part of a routine, executes the remaining lines of the routine and returns control to the caller of the routine. The completed method call is highlighted in the Source Editor.

Command	Shortcut	Description
Debug > Run to Cursor	F4	Runs the program to the cursor location in the Source Editor and pauses the program. The file you have selected in the Source Editor must be called from the main class of the main project.

For more details, see the following document:

<https://netbeans.org/kb/docs/php/debugging.html>

#### 21.4.5.4 Finishing a PHP Debugging Session

If necessary, you can stop the current debugging session using the Shift+F5 shortcut. You can also close a specific debugging session using the Sessions window.

##### To finish the current debugging session:

1. Choose **Debug > Finish Debugger Session** (Shift+F5).

##### To finish one of several debugging sessions:

1. Open the Sessions window by choosing **Window > Debugging > Sessions** (Alt+Shift+6).
2. Right-click the debugging session you want to stop and choose **Finish**.

#### 21.4.6 How to Use PHP Debugger Windows

When you start a debugging session the IDE opens some debugging windows by default as tabs below the editor. You can open any debugger window by choosing **Window > Debugging > window-name** (for example, **Window > Debugging > Breakpoints**).

Each debugger window displays a variety of icons to relay information about the object. For example, the Breakpoints window uses a small red square to indicate a breakpoint set on a line. Some windows also include a node expansion control to the left of the icon. Clicking this control expands and collapses the object.

In the debugger tabs, information is organized into lists. Each list item represents a single object. Each column represents a property of the object. Data displayed in blue underlined text is linked to the source code.

Some elements of list in the debugger tabs have editable properties, such as the value property of a variable in the Variables window. If you select a property and the property has a white background you can edit the property. A selected property with a gray background cannot be edited.

You can move a column in a debugger tab by dragging the column header. You can click to the right of the column titles to open a dialog box that enables you to choose the columns that are displayed. You can right-click an item in the list to open a pop-up menu where you can choose to perform actions on that item and modify how the items are displayed in the tab.

##### 21.4.6.1 Using the Breakpoints Window

The Breakpoints window lists all the breakpoints that you set in your projects. You can open the Breakpoints window by choosing **Window > Debugging > Breakpoints**. If you open the Breakpoints window when a debugging session is running, it closes automatically when you end the debugging session. If you open the window when no debugging session is running, it stays open until you close it.

By default, each entry contains a short text description of the breakpoint and a checkbox indicating whether the breakpoint is enabled or disabled. You can enable or disable a breakpoint directly in the Breakpoints window by selecting or deselecting the checkbox.

#### 21.4.6.1.1 Icons

The icon to the left of the breakpoint summary indicates the type of the breakpoint.

Icon	Description
	Breakpoint set on a specific line.
	Invalid line breakpoint.
	Disabled breakpoint.

#### 21.4.6.1.2 Actions

The pop-up menu in the Breakpoints window includes the following items.

Menu Item	Description
Go to Source	Finds the location of the selected breakpoint in the source code.
Disable	Turns off the selected breakpoint, but preserves an entry for it in this window.
Move Into Group	You can organize breakpoints into groups so that you can enable, disable, and delete the breakpoints as a group. You can select one or multiple breakpoints and then choose this option to create or add the breakpoints to a group. The group name is displayed as an expandable node in the Breakpoints window in the Custom Group view. You can also choose to display custom groups in the Nested view.
New Breakpoint	Creates a new breakpoint.
Enable All	Enables all the breakpoints in this window.
Disable All	Disables all the breakpoints in this window.
Delete	Deletes the selected breakpoint(s). <b>Note:</b> If you choose this option, the breakpoint(s) is deleted immediately. There is no confirmation dialog box.
Delete All	Deletes all the breakpoints in this window. <b>Note:</b> If you choose this option, all breakpoints are deleted immediately. There is no confirmation dialog box.

#### 21.4.6.1.3 Grouping Breakpoints

You can modify how breakpoints are organized in the Breakpoints window by clicking the Breakpoint Groups button and selecting a grouping strategy from the popup window. If you choose Nested you can specify the specific groups and the order of the groups that are displayed.

#### 21.4.6.2 Using the Call Stack Window

The Call Stack window lists the sequence of calls made during execution. When the debugger is suspended, the Call Stack window shows the sequence of function calls

(that is, the call stack). On initial suspension, the top-most stack frame is selected automatically.

Double-click on a function call in the window to go to that line in the editor. If the call is made to a PHP class, the Navigator window will also go to that line when you double-click the call.

You can double-click on a call stack frame to select it, then explore any variable or expression values for that frame in the Variables and Watches windows.

#### 21.4.6.2.1 Icons

The following table describes the icons displayed to the left of the call name.

Icon	Description
	The current call on the call stack.
	A call other than the current call.

#### 21.4.6.3 Using the Threads Window

The Threads window lists all threads in the current debugging session. You open the Threads window by choosing **Window > Debugging > Threads**.

The information given for each thread is the thread name, state and if the thread is suspended. One thread is the current thread. By default, the current thread is the thread in the current session from which the debugger gained control. When you select a different current session, the Threads window is updated to show the threads for that session.

#### 21.4.6.4 Using the Variables Window

The Variables window lists the local variables in the current call. By default, the Variables window opens automatically whenever you start a debugging session. You can open the Variables window by choosing **Window > Debugging > Variables**.

If you open the Variables window when a debugging session is running, it closes automatically when you end the debugging session. If you open the window when no debugging session is running, it stays open until you close it.

The information given for each variable includes the variable name, type, and value. You can click the control to the left of the name to expand or collapse the variable. If the object type is displayed in blue underlined text, clicking the text jumps to the object type in the source code. You can click the Value cell to edit the value directly in the Variables window.

In some cases, the debugger assigns a pound sign (#) and a number as the variable's value. This number is a unique identifier of the given instance. You can use this identifier to determine if a variable points to the same or to a different instance. You cannot edit this value.

##### 21.4.6.4.1 Properties

By default, all properties are displayed directly in the Variables window, except for the `toString()` property. To display the `toString()` property in the Variables window, right-click in the window and choose **List Options > Change Visible Columns** and then select the `toString()` checkbox in the dialog box.

### 21.4.6.5 Using the Watches Window

The Watches window lists all variables and expressions that you have specified to watch while debugging. By default, the Watches window opens automatically whenever you start a debugging session. You can open the Watches window by choosing **Window > Debugging > Watches**.

If you open the Watches window when a debugging session is running, it closes automatically when you end the debugging session. If you open the window when no debugging session is running, it stays open until you close it.

The information given for each watch includes the variable or expression name, type, and value. If the variable has a control to its left, you can click the control to expand or collapse the object. If the object type is displayed in blue underlined text, clicking the text jumps to the object type in the source code. You can click in the Value cell to edit the value directly in the Watches window.

When you add a new variable or expression to the Watches window, the value of the variable or expression is immediately evaluated and displayed. The value of a watch is based on the current context. As you move through your program code, the Watches window is updated to show the value of the watch for that context.

In some cases, the debugger assigns a pound sign (#) and a number as the variable's value. This number is a unique identifier of the given instance. You can use this identifier to determine if a variable points to the same or to a different instance. You cannot edit this value.

#### 21.4.6.5.1 Properties

You can see all the properties for a watch by right-clicking the node for the watch and choosing Properties.

By default, all properties are also displayed directly in the Watches window, except for the `toString()` property.

## 21.4.7 How to View Program Information

While debugging PHP applications, you can view two types of programming information:

- variables, either in the variables window or in the source editor
- watches, which are mechanisms for observing the changes in the value of a variable or expression during execution.

### 21.4.7.1 Viewing Variables When Debugging PHP Applications

In the IDE, local variables are listed in the Variables window, however it is also possible to evaluate variables directly in the Source Editor.

#### Using the Variables Window

For each variable within the current call, the Variables window displays information including the variable name, type, and value. The Variables window also displays all of the static fields from the present class and all superclasses for each variable, as well as all of the inherited fields from all superclasses.

You can change the value of a local variable directly in the Variables window and then continue running your program with the new value in place.

In some cases, the debugger assigns a pound sign (#) and a number as the variable's value. This number is an unique identifier of the given instance. You can use this

identifier to determine if a variable points to the same instance or to a different instance. You cannot edit this value.

### Evaluating Variables in the Source Editor

You can also evaluate a variable directly in the Source Editor by moving the insertion point over the variable. If the variable is active in the current context, the value of the variable is displayed in a tool tip. In cases where a program includes different variables with the same name, the Source Editor displays the value based on the current context, and not on the instance of the variable in the source code.

You can track the changes in the value of a variable during program execution by setting a watch on the variable. When you create a watch, the value of the variable is immediately evaluated and displayed in the Watches window.

#### 21.4.7.2 Enabling and Creating Watches in PHP Code

A watch enables you to track the changes in the value of a variable or expression during program execution. The Watches window lists all of the watches you defined for all IDE projects, of all programming languages. You can open the Watches window by choosing **Window > Debugging > Watches** (Alt+Shift+2).

Before you can set watches in PHP code, you must enable PHP watches in the Options window. Watches can destabilize XDebug, especially if a watch is set on invalid code.

##### To enable watches for PHP:

1. Open **Tools > Options** (NetBeans Preferences on Mac).
2. Click the PHP icon.
3. Open the Debugging tab.
4. Select Watches and Balloon Evaluation.
5. Optionally, change the Maximum Depth of Structures and/or the Maximum Number of Children. These numbers set the visibility of nested structures and array items, respectively, during Watch evaluation.
6. Click OK.

For more information about PHP Debugging Options, click Help at the bottom of the Debugging tab.

If watches and balloon evaluation is enabled in PHP Options, you can create watches in the Source Editor.

##### To create a watch from the Source Editor:

1. Select the variable or expression in the Source Editor, right-click, and choose **New Watch** (Ctrl+Shift+F7).

The New Watch dialog box opens with the variable or expression entered in the text field.

2. Click **OK**.

The Watches window opens with the new watch selected.

For more details, see the following document:

<https://netbeans.org/kb/docs/php/debugging.html#usingAdditionalWatches>

## 21.5 Testing PHP Applications

You can configure the global settings for the frameworks and tools in the Options window. You enable support for a test framework or tool for each project in the Project Properties window.

### 21.5.1 How to Configure PHP Unit Test Frameworks

You can run PHP unit tests and view test results in the IDE. After you configure the IDE to use the framework you can configure your project to create test skeletons and run tests using the test framework.

---

**Note:** To use a test framework you need to download the required files for the framework to your local system and specify the framework settings in the Options window. The files are not bundled with the IDE.

---

**To configure the testing framework for a project:**

1. Confirm that the files that are required by the test framework are saved to your local system.
2. Open the **Options** window and click the **Frameworks & Tools** tab.  
For more information, see [Section 2.2, "Working with the Options Window."](#).
3. Select the framework in the left pane of the tab and specify the path to the files that are required by the selected framework. Click **Apply** and close the **Options** window.
4. Right-click the project node in the **Projects** window and choose **Properties** in the popup menu.
5. Select **Testing** in the Categories pane.
6. Select a Testing Provider for the project and specify the location of the test directories for the project.
7. Select the testing framework in the left pane of the **Project Properties** window under the **Testing** node and configure the test settings in the right pane. Click **OK**.

### 21.5.2 How to Run PHP Tests

After you write a unit test for your source code you can run tests and view the results in the IDE. You can run tests on the entire project or run tests for individual files. You can also organize tests into suites if you want to run a specific group of tests.

**To run a unit test:**

1. Create a test for your source code.

You can create a skeleton test file for a file by right-clicking the file in the Projects window and choosing **Tools > Create Tests** in the popup menu.

If the test directory for the project is not specified you will be prompted to specify a test directory. If the test framework is not specified you will be prompted to select a test framework and specify the path to any required files.

2. Right-click the project node and choose **Test** to run all the tests in the project.

An overview of the results of the test are displayed in the Test Results window. You can view more detailed information about the tests in the Output window.

### 21.5.3 How to Analyze PHP Code

You can use code analysis tools to inspect your PHP code to compare your code to various PHP code standards. After you configure the global tool settings in the Options window you can use the IDE to inspect files and projects for potential problems in the source code.

---

**Note:** To use a code analysis tool you need to download the required files to your local system and specify the tool settings in the Options window. The tools are not bundled with the IDE.

---

#### To analyze PHP code with an analysis tool:

1. In the Options window, click the Code Analysis tab in the PHP category.
2. Select a code analysis tool in the left pane of the tab.
3. Specify the location of the script for the code analysis tool.  
You can click Browse or Search to locate the script on your local file system.
4. Specify any default global settings for the analysis tool.
5. Click Apply to apply the settings. Click OK to close the Options window.
6. Open the file that you want to analyze in the editor.  
When you open one of the project files in the editor you will also have the option to analyze the entire project.
7. Choose **Source > Inspect** in the main menu to open the Inspect dialog box.
8. Select a Scope in the drop-down list  
You can select the current file, the project containing the current file or all open PHP projects.
9. Select a code analysis tool in the Configuration drop-down list.
10. Click Inspect to run the analysis.

When you click Inspect the results of the analysis are displayed in the Inspector window. In the Inspector window you can navigate the tree to view the potential problems that are identified by the analysis tool.

## 21.6 Using a Continuous Build Server With PHP

The IDE supports creating and starting build jobs using the Hudson build server. Hudson's usefulness for PHP projects is that it allows PHPUnit tests to be performed automatically according to a schedule.

For more information about setting up and using a Hudson build server, see the following documents:

- <http://hudson-ci.org/>
- [http://wiki.eclipse.org/Hudson-ci/Meet\\_Hudson](http://wiki.eclipse.org/Hudson-ci/Meet_Hudson)

The Jenkins build server should also work with the IDE, though this is not so thoroughly tested.

Hudson is usually run on a remote machine. This section contains information for the administrator of the Hudson or Jenkins host machine.

In order for Hudson (or Jenkins) to be used with PHP and to provide test results, the Hudson or Jenkins server should be set up with the plugins described on the web page. (Hudson has equivalent plugins.) The host machine for the Hudson or Jenkins server should also have PHP and PEAR installed with the PHP plugins listed on the Template for Jenkins Jobs for PHP page.

#### To add a Hudson instance:

1. Right-click the Hudson Builders node in the Services window and choose **Add Hudson Instance**.
2. Type the Name for the instance that will be displayed under the Hudson Builders node.
3. Specify the server URL, the auto-refresh setting. Click **Add**.

In most cases the continuous integration server is run on a remote machine. The administrator of the remote machine provides you with the URL.

After you add the server instance, a node for the instance is added below the Hudson Builders node. You can expand the node to view the status of builds on that instance.

For a PHP project to build on Hudson, the project must have the following set up:

- Version control. The project must be under version control, and the type of version control (Git, Subversion, Mercurial, or CVS) must be supported by the continuous integration server, meaning that server must have the support plugins installed for that version control system. Check with your server administrator.
- PHPUnit tests. The project must have PHPUnit tests defined. See the page on PHPUnit Test and Selenium at  
<https://netbeans.org/kb/docs/php/phpunit.html>.

For more details about required plugins, tools and templates that you might need to install, see:

<http://jenkins-php.org/>.

#### To set PHP options for Hudson:

- Open **Tools > Options > PHP** (NetBeans Preferences > PHP on Mac) and go to the Hudson tab.

After PHP options are set you can proceed with setting up a new build job.

#### To set up a new build job:

1. Choose **Team > Create Build Job** from the main menu. Alternatively, in the Services window, right-click the Hudson instance you want to use and choose New Build.
2. Select the build server instance from the drop-down list.
3. Specify the name for the build job.
4. Select the project from the drop-down list.  
The build server will use the sources in the project's repository.
5. Click Create. The IDE generates build.xml and phpunit.xml.dist files in the project.

6. Commit the project's new build.xml and phpunit.xml.dist files in your version control system.

After you supply the details for the build, you can start the build process on the server by right-clicking the build that you want to start and choosing Start Job. When a job is building, the node for the job is displayed as running. You can expand the node for the job to view past builds and build artifacts.

To create builds and start jobs you must have access to a Hudson server. For more information on Hudson, see the Hudson wiki at <http://www.eclipse.org/hudson/>.

# 22

---

## Developing Java ME Applications

This chapter describes how to create, test, debug, and deploy Java ME applications using NetBeans IDE.

This chapter contains the following sections:

- [About Developing Java ME Applications](#)
- [Understanding Java Platform Manager](#)
- [Creating Java ME Projects](#)
- [Customizing MIDlet Properties](#)
- [Building Java ME Embedded Applications](#)
- [Working with Security and MIDlet Signing](#)
- [Working with Java ME Embedded Project Configurations](#)
- [Running Java ME Embedded Applications](#)
- [Using Java ME Emulator Platforms](#)

### 22.1 About Developing Java ME Applications

The Java ME project development support for NetBeans enables you to create applications for mobile devices with the following features:

- Java ME Embedded Profile development
  - Integrated compilation and execution of MIDlets and MIDlet suites

MIDlet is a Java class or group of classes that comprise the basic unit of execution in a Java ME Embedded application. A Java ME Embedded application, or MIDlet, is deployed as a MIDlet suite.

A MIDlet suite is a combination of a JAR file that collects the MIDlets and the associated files necessary to create the Java ME Embedded application, and a Java Application Descriptor (JAD) file that identifies the application to target devices. A MIDlet suite can be built, executed, and deployed to mobile devices.

A MIDlet suite includes:

- \* A Java Application Descriptor (JAD) file. This file contains a predefined set of attributes (denoted by names that begin with `MIDlet-`) that allows Application Management Software (AMS) to identify, retrieve, and install the MIDlets.
- \* A Java Archive (JAR) file. This contains:

Java classes for each MIDlet in the suite.

Java classes shared between MIDlets.

Resource files used by the MIDlets (for example, image files).

A manifest file describing the JAR contents and specifying attributes used by AMS to identify and install the MIDlet suite.

- Support for MEEP 8 MIDlet development.

For the Java ME platform, the Micro Edition Embedded Profile (MEEP) defines a standard set of Java APIs that, together with the Connected Limited Device Configuration (CLDC), provides a complete Java ME application runtime environment targeted at mobile information devices, such as cellular phones and wireless personal organizers.

More information about the MEEP profile is available at

<http://docs.oracle.com/javame/config/cldc/opt-pkgs/api/meep/api/index.html>.

- Automatic generation of JAD and JAR files.

The Java Application Descriptor File (.jad) contains header information for a MIDlet suite, such as the company that developed the application, application name, and size. When an application is downloaded to a device, the descriptor is downloaded and read before the JAR file that contains the Java ME Embedded application. This enables the device to confirm that space exists for the application, and confirms that the full application should be downloaded.

- Integrated source-level debugging of MIDlets.

- Code completion against Java ME APIs.

- Emulator Platform Support.

An emulator simulates how your application will operate on a specific type of mobile device such as standard and touch screen mobile phones. Emulator platforms include specific device examples, or skins, that simulate the operations of a specific device.

- Includes the Java ME SDK

- Integration with third party emulators and SDKs through the Unified Emulator Interface support.

Unified Emulator Interface (UEI) is a common set of standards specifying command-line access to emulator functionality. The UEI allows IDE manufacturers to write to a single interface and, with little or no effort, be able to support emulators from many different companies.

- Ability to create project configurations to test your application against multiple devices.

- Integrated wizards and templates that enable you to:

- Create new projects, MIDlets, and other Java ME files.

- Create client/server code for connecting a MIDlet to Web services.

- Support for the ProGuard obfuscator

- Support for Push Registry, which facilitates the signing of MIDlet suites. This support includes an API Permissions tool to set permissions on sensitive APIs in the MIDlet Suite.

- A variety of deployment technologies.

- Support for Over-The-Air (OTA) download testing. This support simulates the complete lifecycle of an application, starting from the Over-The-Air (OTA) Provisioning from a web site, installing and verifying, running, updating, and removing your application without involving an external web server.

### 22.1.1 About Java ME Embedded and CLDC Applications

A Java<sup>TM</sup> 2, Micro Edition platform (Java ME<sup>TM</sup> Platform), Mobile Edition Embedded Profile (MEEP) application is an application targeted for mobile devices such as mobile phones, PDAs, and two-way pagers. Such applications conform to both the Connected, Limited Device Configuration (CLDC) and MEEP.

A Java ME Embedded application, or "MIDlet Suite," typically consists of:

- A Java Application Descriptor file. This file contains a predefined set of attributes (denoted by names that begin with MIDlet-) that allows Application Management Software to identify, retrieve, and install the MIDlets.
- A Java Archive file that contains:
  - Java classes for each MIDlet in the suite.
  - Java classes shared between MIDlets.
  - Resource files used by the MIDlets (for example, image files).
  - A manifest file describing the JAR contents and specifying attributes used by the AMS to identify and install the MIDlet suite.

In the IDE, MIDlet Suites are managed as Java ME Embedded projects. Java ME Embedded projects are executed through an emulator platform, which simulates the execution of the application on a specific mobile device. Using project configurations and preprocessor code blocks, you can define conditions for multiple devices in a single set of code, enabling you test and produce MIDlet suites for each device.

## 22.2 Understanding Java Platform Manager

The Java Platform Manager is a tool for registering different versions of the Java Development Kit (JDK) and other Java tools that your programs depend on. The dialog box lists all of your registered JDKs and SDKs in the left pane. It also lists the JDK that the IDE is running on as the Default Platform.

You can open the dialog box by choosing **Tools > Java Platforms**.

Using the Java Platform Manager, you can:

- Add or remove emulator and SDK platforms for mobile and PDA devices.
- View the registered platform classpath.
- Add or remove platform source files.
- Add or remove java documentation sources
- Use tools and extensions provided by the emulator

---

**Note:** Though NetBeans supports Java Micro Edition (ME) Connected Limited Device Configuration (CLDC) development, the IDE does not have a CLDC platform included in the installer that allows you to create applications for CLDC devices. For more information, see <http://wiki.netbeans.org/JavaMESDKSupport>.

---

## 22.3 Creating Java ME Projects

A project is a group of source files and the settings with which you build, run, and debug those source files. In the IDE, all Java development has to take place within a project. Within a Java ME Embedded project, you can create multiple project configurations that customize your MIDlet for different types of mobile devices.

When a project is created, the IDE:

- Creates a source tree you can examine in the Projects or Files views.
- Sets the emulator platform for the project.
- Sets the project run and compile-time classpaths.
- Creates a build script that contains actions for running, compiling, debugging, and building Javadoc.

The Java ME Embedded project structure is similar to that of standard NetBeans projects. There are differences, however, in the `build` and `dist` folders to accommodate the MEEP build process.

### Logical View

The logical view of the project, shown in the Projects window, provides a hierarchy of sources and resources which reflects the type of project and its contents.

The Source Packages node encapsulates all the Java packages of the project. Right-clicking on the Source Package node and choosing New lets you add new file templates to your application.

### Files View

When a project is created, the following folders are created under `build` (shown in the Files window):

- `compiled`. Contains all compiled classes.
- `obfuscated`. Holds the obfuscated versions of the class files.

### 22.3.1 How to Create a Java ME Embedded Application from a Template

The IDE manages applications in projects. To create a new application, you must first create and configure a new project. Use the New Project wizard or the project properties to create multiple configurations that enable you to simultaneously build and deploy your application to a range of platforms and devices.

With project templates, the IDE controls all aspects of how your application is built, run, and debugged. You set a project's source directory, classpath, emulator platform, and other project settings when creating the project and in the Project Properties dialog box.

NetBeans IDE comes with the following Java ME Embedded project template:

- **Embedded Application.** A new Java ME Embedded application in a standard IDE project.

#### To create a new project:

1. Choose File > New Project.
2. Select the template for your project.

3. Specify the name of the project and where to create it. The IDE creates the project folder in the specified location. Click **Next**.
4. Specify the emulator platform for the project. The emulator platform determines the execution environment for your project.
5. Click **Finish** to complete the wizard.

#### 22.3.1.1 Creating Embedded Application

You open the New Java ME Embedded Application wizard by choosing **File > New Project** and selecting the Java ME Embedded Application project template from the Java ME Embedded category.

The IDE creates a new Java ME Embedded application in a standard IDE project. You can also generate a MIDlet in the project.

#### 22.3.2 How to Add a MIDlet to a Project Configuration

##### To add a MIDlet to a Project Configuration:

1. Open the MIDlets properties page by choosing **File > <active project name> Properties**.
2. In the Project Properties dialog, select **Application Descriptor > MIDlets**.

---

**Note:** If you are setting the MIDlet properties for a project configuration other than the default configuration, you must uncheck the Use Values from "DefaultConfiguration" checkbox at the top of the page.

---

3. Click the **Add** button. This opens the Add MIDlet dialog. The dialog lists the MIDlets available in the project.
4. Use the **Add MIDlet** dialog to enter a name, then select a MIDlet class from the dropdown menu. You can also choose an icon for the MIDlet from the MIDlet icon dropdown menu.
5. Click **OK** to close the dialog. The MIDlet is listed in the table.

#### 22.3.3 How to Add Push Registry Entries

The push registry allows MIDlets to be launched in response to an incoming network connection. When a MIDlet registers for push notifications, the device software is instructed to listen for incoming network connections and launch the MIDlet, if appropriate.

A MIDlet can register for push connections at runtime using static methods in `javax.microedition.io.pushregistry`, which would require the user to run the MIDlet manually when the device is turned on.

A more common approach is to register at installation using properties contained in the application descriptor (JAD) file. This approach allows the AMS to ensure that the MIDlet can be registered, and to cause the installation to fail if there are registration conflicts (for example, if another MIDlet is already registered for the same socket connection).

Each key designation is unique and of the form MIDlet-Push-<n>, where <n> begins at 1 and is incrementally increased with each registration. A MIDlet suite can have multiple push registrations.

---

**Note:** Push registry configuration is only available for projects whose profile version is set to MDP 2.0.

---

#### To add Push Registry Entries:

1. Choose **File > <active project name> properties**. This opens the **Project Properties** dialog.
2. Select **Push Registry** from the left pane.
3. In the **Push Registry** properties page, click **Add** to add a new MIDlet, and provide the following information in the **Add Push Registry Entries** dialog:
  - **Class Name.** The MIDlet's class name.
  - **Sender IP.** A valid sender that can launch the associated MIDlet. If the value is the wildcard (\*), connections from any source are accepted. If datagram or socket connections are used, the value of Allowed Sender can be a numeric IP address.
  - **Connection String.** A connection string that identifies the connection protocol and port number.

A push registration key is automatically generated and shown as an attribute in the MIDlet suite's Java Application Descriptor (JAD) file. To view the JAD file, choose the Application Descriptor Property settings from the left pane.

---

**Note:** To make use of the Push Registry, you must also have permission to access the Push Registry API, `javax.microedition.io.PushRegistry`. API permissions are handled in the API Permissions page of the Project Properties.

---

Each key designation is unique and of the form MIDlet-Push-<n>, where <n> begins at 1 and is incrementally increased with each registration. A MIDlet suite can have multiple push registrations.

For more information about using the push registry, see <http://www.oracle.com/technetwork/systems/index-156665.html>.

#### 22.3.4 Adding API Permissions

If your MIDlet suite needs to access certain protected APIs (for example, network connections), the MEEP 8 security architecture requires that the connection grant permission for access. Required permissions are listed in the Java Application Descriptor (JAD) file with the attribute `MIDlet-Permissions`.

You can also define optional permissions to limit access to sensitive data or functionality with the attribute `MIDlet-Permissions-Opt`.

During MIDlet suite installation, the MEEP compares the permissions requested with the permissions in the destination protection domain. If the required permissions can not be granted, the application will not be installed.

During execution, if a required permission is denied, an exception is returned. If an optional permission is denied, the application might continue to function, although its functionality would be limited.

You can set the permission requests from the API Permissions page of the Project Properties.

#### To Add API Permissions:

1. Choose **File > <active project name> properties**. This opens the **Project Properties** dialog.
2. Select API Permissions from the left pane.
3. Click the **Add Button**.
4. Choose an API from the dropdown list or enter an API name in the combo box. Permissions have the same naming structure as a Java class. For instance, `javax.microedition.io.Connector.http` is the permission for the HTTP protocol.
5. Click **OK**.
6. In the API Permissions table, check the **Required** box if you want installation to fail in the event that permission cannot be granted.

For more information about API Permissions, see

<http://docs.oracle.com/javame/config/cldc/opt-pkgs/api/meep/api/index.html>.

## 22.4 Customizing MIDlet Properties

You can add, edit, or delete the attributes of a MIDlet suite for the active project configuration, and determine the order in which MIDlets are shown in a MIDlet suite.

#### To add a MIDlet to a suite:

1. Open the MIDlets properties page by choosing **File > <active project name> Properties**.
2. In the **Project Properties** dialog, select **Application Descriptor** and the **MIDlets** tab.
3. Click the **Add** button. This opens the **Add MIDlet** dialog. The dialog lists the MIDlets available in the project.
4. Use the **Add MIDlet** dialog to enter a name, then select a MIDlet class from the dropdown menu. You can also choose an icon for the MIDlet from the MIDlet icon dropdown menu.
5. Click **OK** to close the dialog. The MIDlet is listed in the table.

---

**Note:** Any class added to the project that extends MIDlet (directly or indirectly) will be automatically be added to the Application Descriptor.

---

#### To edit a MIDlet in the suite:

1. Open the MIDlets properties page by choosing **File > <active project name> Properties**.
2. Under Categories, select **Application Descriptor**. Select the **MIDlets** tab.
3. Select a midlet and click the **Edit** button. This opens the **Edit MIDlet** dialog.

4. Use the **Edit MIDlet** dialog to edit the name, MIDlet class, and MIDlet icon.

**To change the order in which a MIDlet is shown:**

Select an attribute and click the **Move Up** or **Move Down** button.

#### 22.4.1 How to Customize the Application Descriptor (JAD) Attributes

You can choose which attributes are included or excluded in your project configuration by choosing **File > <active project name> properties**. In the **Project Properties** dialog, select **Attributes**.

The **Application Descriptor** page enables you to add, edit, or remove the list of attributes that allow application management software to identify, retrieve, and install the MIDlets of a MIDlet suite.

**To add, edit, or remove an attribute:**

1. Click the **Add** button. The **Add Attributes** dialog opens.
2. Choose an attribute from the combo box. Then add a value for the attribute. Note that the list of available attributes is dependent upon the MEEP version supported by the project configuration, as set in the Platform properties. You can also add user-defined attributes by entering the attribute in the combo box.

---

**Note:** To avoid errors in verification, make sure all required attributes have a defined value. Also, do not begin user-defined attribute keys with `MIDlet-` or `MicroEdition-`.

---

#### 22.4.2 How to Customize MEEP Javadoc Settings

For each of your projects, you can produce a set of Javadoc HTML pages that describe the project's classes, inner classes, interfaces, constructors, methods, and fields. The Javadoc is constructed from the structure of your code and the Javadoc comments embedded in your code.

You open the **Documenting** page by choosing **File > Project Properties (project name)**. In the **Project Properties** dialog, select **Build > Documenting**.

**To set the javadoc generation properties:**

1. Choose **File > Project Properties (project name)**.
2. In the **Project Properties** dialog, select **Build > Documenting**.

For more information on generating Javadoc documentation, see  
<http://docs.oracle.com/javase/7/docs/technotes/tools/windows/javadoc.html>.

### 22.5 Building Java ME Embedded Applications

The development life cycle of applications for mobile devices that support the Java ME, CLDC, and MEEP technologies includes building Java ME Embedded applications.

The IDE builds its project infrastructure directly on top of Apache Ant, the leading build tool for Java applications. The IDE stores all of the information about your project in an Ant script, a properties file, and a few XML configuration files. This means that you can build and run your projects outside of the IDE exactly the same as inside the IDE.

**To build a Java ME Embedded project:**

Choose **Build > Build Project**.

**To create JAR files for multiple devices at once:**

Right-click on the project and choose **Batch Build**.

**To view and customize the build options for a project configuration:**

1. Choose **File > Project Properties (project name)**.
2. Expand the **Build** node in the left pane and choose one of the following:

- **Compiling.** Customize javac options for generating debugging information, optimizing the compilation, and reporting the use of deprecated APIs. You can also set encoding and the level of debugging for the `//#debug` and `//#mdebug` preprocessor directives.
- **Signing.** Sign MIDlets and add or remove security keystores.
- **Obfuscating.** Set the obfuscation level for the application.
- **Documenting.** Manage the source code documentation for the application.

The active project configuration is listed in the **Configuration** drop down menu on the **Platform** panel of the **Project Properties** dialog.

### 22.5.1 How to Customize Compilation Options

You can set the build options in a project configuration by choosing **File > <active project name> properties**. In the **Project Properties** dialog, select **Build > Compiling**.

Click in the associated check box to enable the following options:

- **Generate Debugging Info.** If checked, the compiler generates line numbers and source files information. This is the `-g` option in `javac`. If unchecked, no debugging information is generated (`-g:none`).
- **Compile with Optimization.** If checked, the compiled application is optimized for execution. This is the `-O` option in `javac`. Optimizing can slow down compilation, produce larger class files, and make the program difficult to debug.
- **Report Uses of Deprecated APIs.** If checked, the compiler lists each use or override of a deprecated member or class. This is the `-deprecated` option in `javac`. If unchecked, the compiler shows only the names of source files that use or override deprecated members or classes.

You can also set the default encoding used by the preprocessor and compiler in the **encoding** field. The default value is the default encoding used by your JVM and IDE.

The **Debug Block Level** drop-down menu sets the debug level for the `//#debug` and `//#mdebug` preprocessor directives. If you enter a unique value for the level (for example, `debug_graphics`), the preprocessor allows only debug blocks that exactly match the debug block (for this example, `//#debug debug_graphics`). All other debug blocks are commented out.

If you choose a pre-defined level from the drop-down menu, the preprocessor comments out debug blocks on lower levels, as defined by the following hierarchy:

- `fatal` (most exclusive)
- `error`
- `warn`

- info
- debug (most inclusive)

For example, if you choose warn, all info and debug blocks will be commented out.

---

**Note:** If you are setting the properties for a project configuration other than the default configuration (DefaultConfiguration), you must uncheck the **Use Values from "DefaultConfiguration"** checkbox at the top of the page to customize these options.

---

## 22.5.2 How to Add Libraries and Resources to a Java ME Embedded Project

Adding a group of class files to a project's classpath tells the IDE which classes the project should have access to during compilation and execution. Use the **Libraries & Resources** properties page to add or remove additional libraries, projects, or JAR and ZIP files to a classpath for a project configuration.

---

**Note:** If you are setting the properties for a project configuration other than the default configuration (DefaultConfiguration), you must uncheck the **Use Values from "DefaultConfiguration"** checkbox at the top of the page to customize these options.

---

You can add the following item types to your project:

- **Project.** The JAR file produced by another project, as well as the associated source files and Javadoc documentation. Adding this item to a class path sets up a dependency between the current project and the selected JAR.
- **Library.** A collection of JAR files or folders with compiled classes, which can optionally have associated source files and Javadoc documentation.
- **JAR file.** A JAR file somewhere on your system. Adding a JAR file creates a library entry named by the JAR and containing the JAR.
- **Folder.** The root of a package or directory containing files. Adding a folder does not create a library entry.

### To add a project, file, folder or library to the project's classpath:

1. Choose **File > <active project name> properties**. This opens the Project Properties dialog.
2. Expand **Build > Libraries & Resources** in the left pane.
3. Click on the appropriate button to open a file chooser for that item type. Clicking the **Add Library** button opens the Library Manager, where you can add or edit library classes.

---

**Note:** If a project has multiple distribution JAR files, all of the JARs appear in the Project Jar Files window. You will need to select the correct version of the JAR file to include with the project configuration.

---

### 22.5.3 How to Customize MEEP Build Filter Settings

Filtering settings determine the files and packages that are included in a JAR file for a specific project configuration.

#### To set the Filtering options:

1. Choose File > <active project name> properties.
2. In the Project Properties dialog, expand Build > Filtering.
3. Using the Filtering Properties dialog, you can set the following options:
  - **Use Default Ant Excludes.** If checked, files that are defined by Ant to be excluded by default (version control files and folders) are excluded from the JAR file.
  - **Exclude Test and Test Packages.** If checked, tests and test packages are excluded from the JAR file.

#### To include or exclude an item from the build:

All items are included by default. Expand the nodes and uncheck the boxes next to the files or packages you want to exclude.

---

**Note:** If you are setting the properties for a project configuration other than the default configuration (DefaultConfiguration), you must uncheck the **Use Values from "DefaultConfiguration"** checkbox at the top of the page to customize these options.

---

### 22.5.4 How to Obfuscate a MIDlet Suite

Performance and security are primary concerns for any application. Because the Java language was designed to be compiled into a platform-independent bytecode format, much of the information contained in the source code remains in the bytecode, making it easy to decompile or reverse-engineer a Java application. This format can also contribute to larger file sizes, and with the limited resources of mobile devices, every byte saved in a Java ME Embedded application counts towards mobility and performance.

Obfuscators use a variety of methods to transform compiled code. Code obfuscation techniques include stripping object code of its symbol tables, or altering the names of variables and identifiers. Obfuscated programs are not only more difficult to decompile or reverse-engineer, they are also frequently smaller in size, improving performance and mobility as well as security.

You can add additional obfuscation parameters in the Additional Obfuscation Settings window. The default obfuscator included with the IDE is ProGuard.

You can find more details about command parameters for this obfuscator at <http://proguard.sourceforge.net/>.

Obfuscators make it more difficult for your application to be decompiled or re-engineered. Depending on the methods used, obfuscation can also reduce the size of the compiled MIDlet suite, reducing the size of the application and improving performance.

#### To obfuscate a MIDlet suite:

1. Choose File > <active project name> Properties.

2. In the **Project Properties** dialog, select **Build > Obfuscating**.
3. Use the Obfuscation lever to set the level of obfuscation for the suite. The **Level Description** window describes the impact each level has on the MIDlet.
4. You can add additional obfuscation parameters in the **Additional Obfuscation Settings** window.

---

**Note:** If you are setting the properties for a project configuration other than the default configuration (Default Configuration), you must uncheck the **Use Values from "DefaultConfiguration"** checkbox at the top of the page to customize these options.

---

### 22.5.5 How to Customize the JAR and JAD Files in a Build

You can modify the names of the JAR and JAD files, and specify whether the JAR file is compressed.

**To modify the JAR and JAD files:**

1. Choose **File > <active project name> Properties**.
2. In the **Project Properties** dialog, select **Build > Creating JAR**.
3. The Creating JAR page lets you set the following options:
  - **JAD File Name.** Name of the JAD file created by the project sources. The file name must have a .jad extension.
  - **JAR File Name.** Name of the JAR file created by the project sources. The file name must have a .jar extension.
  - **Compress JAR.** If checked, the JAR file is compressed.

---

**Note:** If you are setting the properties for a project configuration other than the default configuration (Default Configuration), you must uncheck the **Use Values from "DefaultConfiguration"** checkbox at the top of the page to customize these options.

---

## 22.6 Working with Security and MIDlet Signing

Sigining a MIDlet suite allows MEEP devices to verify the integrity and origins of your MIDlet suite. MEEP devices use signing information to check an application's source and validity before allowing it to access protected APIs. To sign a MIDlet suite, you create a *key pair*:

- A private key that is used to create a digital signature, or certificate.
- A public key that can be used by anyone to verify the authenticity of the signature.

Each certificate has a designated *security domain*. Once your MIDlet suite has been verified, it can access any of the protected APIs permitted by its security domain.

You can sign a MIDlet suite by:

- using the default keystore (with the security domains described above).
- importing a key pair from an existing keystore.
- creating a new key pair.

You then have to export the certificate and set its security domain in each emulator. For more information about Java ME security features, see <http://www.oracle.com/technetwork/systems/index-156466.html>.

### 22.6.1 How to Set Security through MIDlet Signing

Siging a MIDlet suite allows MEEP devices to verify the integrity and origins of your MIDlet suite. To sign a MIDlet suite, you use the Signing Properties page to assign or create a *key pair*:

- A private key that is used to create a digital signature, or certificate.
  - A public key that can be used by anyone to verify the authenticity of the signature.
- You can then export the key pair to the emulator platform you are using for testing.

**To sign a MIDlet suite:**

1. Choose File > <active project name> Properties. In the Project Properties dialog, select Build > Signing.
  2. Check the **Sign Distribution** checkbox to enable signing.
  3. Choose a keystore from the **Keystore** dropdown menu.
- 
- Note:** To create a new keystore or key pair, click the **Open Security Manager** button.
- 
4. Choose an alias from the **Alias keystore** dropdown menu. The menu lists the available aliases for the selected keystore.
  5. If necessary, click the unlock buttons to make the keystore and alias available for use.
  6. Click the **Export Key into Java ME SDK/Platform/Emulator** button to export a key into the emulator platform you intend to use to test the MIDlet.

### 22.6.2 How to Add or Creating a Keystore

Siging a MIDlet suite allows MEEP devices to verify the integrity and origins of your MIDlet suite. To sign a MIDlet suite, you assign or create a *key pair*.

- A private key that is used to create a digital signature, or certificate.
  - A public key that can be used by anyone to verify the authenticity of the signature.
- Key pairs are stored in keystore (.ks) files.

**To create a new keystore:**

1. Choose File > <active project name> Properties. In the Project Properties dialog, select Build > Signing.
2. Click the **Open Keystores Manager** button. This opens the **Keystores Manager** dialog.
3. Click the **Add Keystore** button.
4. Choose the **Create a New Keystore** radio button and enter the following information:
  - **Keystore Name.** A legal filename with the extension .ks

- **Keystore Folder.** Folder in your system where the keystore file will be saved.
  - **Password.** The password that can be used to unlock the keystore.
5. Click **OK** to close the dialog. The new keystore is displayed in the **Keystores** window of the Keystores Manager.

**To add an existing keystore:**

1. Choose **File > <active project name> Properties**. In the **Project Properties** dialog, select **Build > Signing**.
2. Click the **Open Keystores Manager** button. This opens the **Keystores Manager** dialog.
3. Click the **Add Keystore** button. This opens the **Add Keystore** dialog.
4. Choose the **Add Existing Keystore** radio button.
5. Click the browse button to navigate to the location of the keystore file you want to add.
6. Click **OK** to close the dialog. The new keystore is displayed in the **Keystores** window of the Keystores Manager.

For more information about using keystores to sign your MIDlet suites, see <http://www.oracle.com/technetwork/java/nbmp40-security-136005.html>.

### 22.6.3 How to Create a New Key Pair

Signing a MIDlet suite allows MEEP devices to verify the integrity and origins of your MIDlet suite. To sign a MIDlet suite, you use the **Signing Properties** page to assign or create a key pair:

- A private key that is used to create a digital signature, or certificate.
  - A public key that can be used by anyone to verify the authenticity of the signature.
- You can then export the key pair to the emulator platform you are using for testing.

**To create a new key pair:**

1. Choose **File > <active project name> Properties**. In the **Project Properties** dialog, select **Build > Signing**.
2. Check the **Sign Distribution** checkbox to enable signing.
3. Choose a keystore from the **Keystore** drop-down menu. The Security certificate details of the selected keystore:
  - a. If no keystore is available, or you want to create a new keystore, click the **Open Keystores Manager** button. This opens the **Keystores Manager** dialog.
  - b. Choose a keystore from the left window of the dialog, or click **Add Keystore** to create a new keystore. Then close the Keystores Manager.
4. Choose an alias from the Alias keystore drop-down menu. The menu lists the available aliases for the selected keystore. You can create new aliases in the **Keystores Manager** dialog.
5. If necessary, click the unlock buttons to make the keystore and alias available for use.
6. Click the **Export Key into Java ME SDK/Platform/Emulator** button to export a key into the emulator platform you intend to use to test the MIDlet.

For more information about key pairs and keystores, see  
<http://docs.oracle.com/javame/dev-tools/jme-sdk-3.0-win/html-helpset/z40001fb51614.html>.

#### 22.6.4 How to Export a Key to an Emulator Platform

To test a signed MIDlet in an emulator device, you must export the key and its corresponding certificate to the emulator platform.

**To export a key:**

1. Choose **File > <active project name> Properties**.
2. In the Project Properties dialog, select **Build > Signing**.
3. Check the **Sign Distribution** checkbox.
4. Select a Keystore and alias (a key pair).
5. Click the **Export Key into Java ME SDK/Platform/Emulator** button.
6. In the **Export Key** dialog:
  - a. Select the emulator platform,
  - b. Select the security domain,
  - c. Click the **Export** button.

### 22.7 Working with Java ME Embedded Project Configurations

One of the most difficult aspects of developing applications for mobile devices is device fragmentation. The situation arises when vendor-specific, proprietary, or optional APIs are added to mobile devices to expand device functionality. Mobile devices often differ in a variety of attributes, such as screen size or color depth, and many support proprietary or optional APIs. These differences can require special code or project settings. To solve this problem, the IDE supports *project configurations*.

A project configuration enables you to define an execution environment for each device you want to run your application on. With project configurations and preprocessing, you can write a single application and create, customize, and deploy a separate distribution JAR for multiple devices.

You should create one configuration for each distribution JAR you plan to build for your project. For example, if you are planning to support three different screen sizes using two sets of vendor specific APIs, you should create six configurations.

Properties that are defined in a project configuration include:

- emulator platform
- specific emulator device
- supported version of the Connected Limited Device Configuration (CLDC)
- supported version of the Micro Edition Embedded Platform (MEEP)
- supported optional APIs
- API permissions
- push registry entries
- obfuscation level
- signing

- javadoc generation
- deployment options

## 22.7.1 How to Work with Java ME Embedded Project Configurations

The following steps outline the basic process of working with Java ME Embedded Project configurations:

1. Create a project using the best template for your programming needs.
2. Create Additional Configurations:

In the **Project Properties** window, select the **Platform** category. Then click **New**.

3. Customize the Configuration.

Right-click the project node and choose **Properties**. Then select the **Platform** category and select the configuration you want to customize in the **Configuration** drop-down list. You can use the same settings as the default configuration, or you can add MIDlets, libraries, and other resource files, define push registry entries and API permissions, and create settings for obfuscation and security. All changes you make to the default configuration, other than to the Run options, propagate to all other configurations automatically. Any changes you make to a configuration other than the default configuration apply only to that non-default configuration.

4. Run Configurations.
  - a. Right-click the project node and chose **Run**.
  - b. Activate a different configuration by selecting the project node and choosing **Run > Set Project Configuration** from the main menu. Then choose a different configuration and run the project again. You can also change the configuration of the selected project with the Configuration combo box in the main toolbar.

## 22.7.2 How to Customize Java ME Embedded Project Configurations

A project configuration defines the execution environment for a Java ME Embedded application that emulates a specific mobile device. You can define multiple project configurations for an application, then customize the properties of each configuration to match the specific devices for which you are programming. This enables you to test and debug your application for each device. You can then use project configurations to deploy a separate distribution JAR for one or all of your configurations.

You customize project configurations using the **Project Properties** dialog.

After a project containing multiple configurations is built, the build folder contains the build directory for DefaultConfiguration. Sub-directories in the dist folder contain the build and distribution files for each configuration in the project.

### To configure settings for a Java ME Embedded project:

1. Choose **File > Project Properties (project name)**.
2. Select the category in the left pane and modify the properties in the right pane.

#### 22.7.2.1 Customizing Project Configurations

Properties that can be viewed or customized in a project configuration include:

- emulator platform
- specific emulator device

- supported version of the Connected Limited Device Configuration (CLDC)
- supported version of the Micro Edition Embedded Platform (MEEP)
- supported optional packages and APIs
- attributes listed in the JAD file
- MIDlets included in the JAR
- push registry entries
- API permissions
- build properties
- libraries and resource files bundled in the project
- file filtering
- obfuscation level
- JAD and JAR file names
- signing
- javadoc generation
- deployment methods

**To create a new configuration for a project:**

1. Open the **Run** category of the **Project Properties** window by right-clicking the project node in the **Projects** window and choosing **Set Configuration > Customize**.
2. Click **New** to open the **Create New Configuration** dialog box.
3. Type a name for the new configuration.
4. Click **OK** to close the **Create New Configuration** dialog box.
5. Select the **Platform** category in the left pane of the **Project Properties** window and modify properties of the new configuration.
6. Click **OK**.

### 22.7.2.2 Customizing Platform Properties

You can set the Platform properties of a project configuration by choosing **File > Project Properties (project name)**. In the **Project Properties** dialog, select **Platform**.

The **Platform** project properties pane enables you to choose an emulator platform that simulates a mobile device to which the application will be deployed. The properties shown are specific to the active project configuration, shown in the **Project Configuration** drop down menu of the dialog.

**To change the active emulator platform:**

Choose an installed emulator from the **Java ME Platform** drop down menu.

**To add or remove an emulator platform:**

Choose **Tools > Java Platforms** from the Main Toolbar, or click the **Manage Platforms** button in the **Platform Properties** dialog. This opens the **Java Platform Manager**, where you can add or remove the platform.

**To set the version of CLDC or MEEP supported:**

Choose the appropriate radio button for **Configuration** and **Profile**.

---

**Note:** The configuration and profile versions supported depend upon the platform emulator and device selected.

---

**To choose the optional packages and APIs supported by the emulator platform:**

Check the appropriate check boxes in the **Optional Packages** pane of the **Project Platform Properties** dialog. The dialog lists all the packages provided by the selected emulator platform.

**22.7.2.3 Customizing the Application Descriptor (JAD) Attributes**

You can choose which attributes are included or excluded in your project configuration by choosing **File > Project Properties (project name)**. In the **Project Properties** dialog, select the **Application Descriptor** category.

The **Attributes** tab enables you to add, edit, or remove the list of attributes that allow application management software to identify, retrieve, and install the MIDlets of a MIDlet suite or LIBlets.

**To add, edit, or remove an attribute:**

1. Click the **Add** button. The **Add Attribute** dialog opens.
2. Choose an attribute from the combo box. Then add a value for the attribute.

---

**Note:** To avoid errors in verification, make sure all required attributes have a defined value. Also, do not begin user-defined attribute keys with MIDlet- or MicroEdition-

---

**22.7.2.4 Adding MIDlets to a Project Configuration****To add an existing MIDlet:**

1. Right-click on a project and choose **Properties**.
2. Select **Application Descriptor** from the **Categories** window and select the **MIDlets** tab in the right pane.
3. Click the **Add** button.
4. Fill in the **Add MIDlet** dialog and click **OK**. The MIDlet is automatically added to the Application Descriptor (JAD) file.

**To add a new MIDlet:**

1. Select a project and choose **File > New File (Ctrl+N)**.
2. Select **Java ME Embedded** under **Categories**, then select the **MIDlet** file template under **File Types**.
3. Enter the appropriate information in the **New MIDlet** wizard.
4. Click **Finish**.

The MIDlet is automatically added to the JAD file.

### 22.7.2.5 Customizing Compilation Options

You can set the build options in a project configuration by choosing **File > Project Properties (project name)**. In the **Project Properties** dialog, select **Build > Compiling**.

For more information on compiler options, see:

<http://docs.oracle.com/javase/7/docs/technotes/tools/windows/javac.html>

Click in the associated check box to enable the following options:

- **Generate Debugging Info.** If checked, the compiler generates line numbers and source files information. This is the `-g` option in `javac`. If unchecked, no debugging information is generated (`-g:none`).
- **Report Uses of Deprecated APIs.** If checked, the compiler lists each use or override of a deprecated member or class. This is the `-deprecated` option in `javac`. If unchecked, the compiler shows only the names of source files that use or override deprecated members or classes.
- **Track Java Dependencies.** If checked, the latest version of any project dependencies is used by the IDE every time you build or run your project.
- **Enable Annotation Processing.** If checked, annotation processing is enabled when your project is built.

### 22.7.2.6 Adding Libraries and Resources to a Java ME Embedded Project

Adding a group of class files to a project's classpath tells the IDE which classes the project should have access to during compilation and execution. Use the **Libraries** properties page to add or remove additional libraries, projects, or JAR and ZIP files to a classpath for a project configuration.

You can add the following item types to your project:

- **Project.** The JAR file produced by another project, as well as the associated source files and Javadoc documentation. Adding this item to a class path sets up a dependency between the current project and the selected JAR.
- **Library.** A collection of JAR files or folders with compiled classes, which can optionally have associated source files and Javadoc documentation.
- **JAR file.** A JAR file somewhere on your system. Adding a JAR file creates a library entry named by the JAR and containing the JAR.
- **Folder.** The root of a package or directory containing files. Adding a folder does not create a library entry.

#### To add a project, file, folder or library to the project's classpath:

1. Choose **File > Project Properties (project name)**.

This opens the **Project Properties** dialog.

2. Select **Libraries** in the left pane.

3. Click the appropriate button (for example, **Add Project**) to open a file chooser for that item type.

Clicking the **Add Library** button opens the **Add Library** dialog, where you can add or edit library classes.

## 22.8 Running Java ME Embedded Applications

When you run a Java ME Embedded application, the IDE compiles and builds the application according to the settings of the current. As the MIDlet executes, the output is shown in the device emulator. You can run any individual project that contains an executable class.

### To run a project:

Choose **Run > Run Project** or right-click any project in the Projects window and choose **Run**.

You can also choose to execute the MIDlet using Over-the-Air provisioning which is the process of deploying an application from a website on to a mobile device. If you choose Execute Through OTA in the Java ME Embedded Project Run Property sheet, the IDE simulates the OTA process so you can test and demonstrate the full provisioning process of the MIDlet suites from the server to a device. Additional information on OTA Application Provisioning is available at <http://www.oracle.com/technetwork/systems/index-156355.html>.

### 22.8.1 How to Customize Java ME Embedded Project Run Options

You can choose one of two different ways to execute a Java ME Embedded program:

- **Regular Execution.** Runs the project on an emulator device. In this mode, you can also set a security domain for the application, to test its security.
- **Execute Through OTA.** Runs the project using Over-The-Air provisioning, which simulates the process of deploying an application from a server to a mobile device.

### To customize the run options:

1. Choose **File > <active project name> Properties**. In the Project Properties dialog, select **Running**.
2. Click a radio button to choose between the two run options.

The MEEP specification includes the Over The Air User Initiated Provisioning Specification, which describes how MIDlet suites can be transferred over-the-air (OTA) to a device.

### 22.8.2 How to Run a MIDlet Suite with OTA Provisioning

Running your MIDlet suite with Over-The-Air (OTA) provisioning emulates the process that takes place when a device downloads and installs your suite from a server. If your suite is signed, the emulator gives it access to protected APIs according to the security domain of the suite's certificate. Running a suite with OTA provisioning also lets you test any inbound connections from the server as defined in the suite's push registry.

### To enable OTA provisioning for an emulator:

1. Choose **File > <active project name> Properties**.
2. In the **Project Properties** dialog, select **Running**.
3. Choose the **Execute Through OTA** radio button.

The IDE enables OTA provisioning for the emulator and runs the MIDlet suite with the specified preferences.

---

**Note:** If you are setting the properties for a project configuration other than the default configuration (DefaultConfiguration), you must uncheck the **Use Values from "DefaultConfiguration"** checkbox at the top of the page to customize these options.

---

### 22.8.3 How to Run Headless Builds

Headless builds for mobility projects are handled under the same principles as standard MEEP projects. If the project has already been opened in the IDE on a target computer and no reference problems exist, then any Ant target can be invoked from within the project directory.

**To invoke the Ant target:**

Type `ant jar run` at the command line.

## 22.9 Using Java ME Emulator Platforms

An emulator platform simulates the execution of an application on one or more target devices. For example, the Java ME SDK enables you to run applications on several example devices, or "skins" included with the emulator. An emulator allows you to understand the user experience for an application on a particular device, and to test the portability of the application across different devices.

In the IDE, you specify the emulator platform as part of the overall project configuration. Many emulator platforms are packaged as software developer kits (SDKs) from manufacturers of mobile devices.

Some of the differences between emulators include their support for:

- Versions of MEEP and CLDC
- Devices, or "skins"
- Optional packages and APIs
- Monitoring and testing tools

Note that having an emulator does not completely free you from testing on your target devices. An emulator can only approximate a device's user interface, functionality, and performance. For example, an emulator might not simulate processing speed, so an application may run faster or slower on a target device than it does on an emulator.

### 22.9.1 How to Add MEEP Emulator Platforms

Adding an emulator platform to the IDE enables you to create project configuration and use that platform for compilation, execution and debugging within the IDE. Through the Java Platform Manager, you can add both custom and UEI-compliant emulator platforms.

Unified Emulator Interface (UEI) is a common set of standards specifying command-line access to emulator functionality. The UEI allows IDE manufacturers to write to a single interface and, with little or no effort, be able to support emulators from many different companies.

**To add a UEI-compliant MEEP emulator platform:**

1. Choose **Tools > Java Platforms**.
2. Click the **Add Platform** button.

**3. Choose Java ME CLDC Platform Emulator.**

Click **Next**.

The wizard searches your system and compiles a list of all available Java ME platforms. The wizard indicates which platforms:

- have already been installed.
- are available for installation. Indicated by a box with a check in it.
- cannot be installed. Indicated in red. You might be able to install these platforms as custom (non-UEI) platforms.

If the platform you want to install is not listed in the page, click on the **Find More Java ME Platforms** and navigate to the directory where the platform is installed.

**4. Put a check in the checkboxes of the platforms you want install.**

Click **Next**.

The chosen platforms are installed.

**To add a non-UEI-compliant emulator platform:**

1. Choose **Tools > Java Platform Manager**.
2. Click the **Add Platform** button.
3. Choose **Custom Java ME CLDC Platform Emulator**. Click **Next**.
4. In the **General Information** page, specify the following information:
  - The directory where the platform is located.
  - Syntax for execution and debugger commands
5. In the **Bootstrap Libraries** page, add any additional libraries used by the emulator.
6. In the **Sources & JavaDoc** page, add any additional source files and Javadoc used by the emulator. Click **Finish** to add the emulator.

## 22.9.2 How to Customize Java ME Platform Properties

You can set the Platform properties of a project configuration by choosing **File > Project Properties (project name)**. In the **Project Properties** dialog, select **Platform**.

The **Platform** project properties panel enables you to choose an emulator platform that simulates a mobile device to which the application will be deployed. The properties shown are specific to the active project configuration, shown in the **Configuration** drop down menu of the **Platform** panel.

**To change the active emulator platform:**

- Choose an installed emulator from the **Java ME Platform** drop down menu.

**To add or remove an emulator platform:**

- Choose **Tools > Java Platform Manager** from the main toolbar, or click the **Manage Platforms** button in the **Platform Properties** dialog.

This opens the **Java Platform Manager**, where you can add or remove the platform.

**To set the version of CLDC/MEEP supported:**

- Choose the appropriate radio button for **Configuration** and **Profile**.

---

**Note:** The configuration and profile versions supported depend upon the platform emulator and device selected.

---

**To choose the optional packages and APIs supported by the emulator platform:**

- Check the appropriate check boxes in the **Optional Packages** pane of the **Project Platform Properties** dialog. The dialog lists all the packages provided by the selected emulator platform.

### 22.9.3 How to Use the Java ME SDK

The Java Micro Edition SDK is a collection of tools and device emulators that support the development of Java applications that run on devices compliant with the Micro Edition Embedded Profile (MEEP) and Connected Limited Device Configuration (CLDC). These include a wide range of mobile phones, embedded systems, Blu-ray consoles and other devices.

NetBeans IDE does not come bundled with the Java ME SDK.

Java ME SDK is supported only on Windows OS.

For more information about the Java ME SDK, see

<http://www.oracle.com/technetwork/java/javame/javamobile/download/overview/index.html>.

For more information about Java ME SDK support in NetBeans IDE, see  
<http://wiki.netbeans.org/JavaMESDKSupport>.



# Working with Web and Application Servers

This chapter describes how to use the NetBeans IDE with Glassfish, Oracle WebLogic Server, JBoss, and Tomcat application servers. It describes how to configure web browsers, and covers developing Java web applications on Oracle Cloud and Amazon Elastic Beanstalk. It also explains how to use the HTTP Server-Side Monitor to diagnose problems.

This chapter contains the following sections:

- [About Working with Web and Application Servers](#)
- [Working with Web Browsers](#)
- [Working with Glassfish Application Servers](#)
- [Working with Oracle WebLogic Application Servers](#)
- [Working with JBoss Application Servers](#)
- [Working with Tomcat Web Servers](#)
- [Working with Web Applications on the Cloud](#)
- [Working with the HTTP Server-Side Monitor](#)

## 23.1 About Working with Web and Application Servers

The NetBeans IDE can be configured to use either the integral web browser, or an alternative web browser. It can pass arguments so that you can fully test your applications.

The IDE supports working with Glassfish, Oracle WebLogic, JBoss, and Tomcat application servers, and it support deployment to Oracle Cloud and Amazon Elastic Beanstalk.

You can diagnose problems using the HTTP Server-Side Monitor.

### 23.1.1 Web Browsers

The IDE uses a web browser to display web pages and run web applications. By default, it uses the default system browser but you can change this to use another browser. You can also configure the web browser to pass arguments to the executable when the browser is run.

For more information, see [Section 23.2, "Working with Web Browsers."](#)

### 23.1.2 Application Servers

A server is a computer that serves up applications. Any computer can be turned into a server if you install server software and connect the computer to the Internet.

A Java EE application server, also known as a Java EE container, is any server that is fully compliant with the J2EE or Java EE platforms. Unlike a regular web server, an application server can run full enterprise applications, EJB modules, and web services. It can also manage transactions of persistent entity objects and communicate with databases.

The IDE supports the following Java EE application servers:

- **GlassFish Server.** The IDE provides the fullest support for the GlassFish application server. You can stop and start the server, deploy applications to it, and view the deployed modules directly in the Services window. There is a visual editor for server-specific deployment descriptors and the IDE updates the descriptors automatically as you write your code. The IDE also automatically configures server resources like connection pools and JavaMail resources for this application server.
- **JBoss Application Server, Oracle WebLogic Server and IBM WebSphere Application Server.** The IDE provides basic support for these application servers. You can start, stop, and debug the servers and deploy applications to them.

The IDE also supports the Tomcat Web Server and its libraries for the following activities:

- Deploying web applications, JSP pages, and servlets.
- Deploying web services and web service clients.

**Tip:** The Tomcat Web Server is not a Java EE container. Therefore, you cannot deploy a web service client that uses JSR-109 stubs to this server. When you create the web service client make sure that you select IDE-generated static stub in the Client Type drop-down. For more information about JAX-WS Web service clients, see [Section 19.5, "Creating JAX-WS Web Service Clients."](#)

Before you can deploy an enterprise application, web application, JSP, servlet, or EJB module, the server to which you are going to deploy needs to be registered with the IDE. For more information, see the appropriate section for your server:

- Glassfish, [Section 23.3.1, "How to Register a Server Instance."](#)
- Oracle WebLogic, [Section 23.4.1, "How to Register a Server Instance."](#)
- JBoss, [Section 23.5.1, "How to Register a Server Instance."](#)
- Tomcat, [Section 23.6.1, "How to Register a Server Instance."](#)

When a server is registered with the IDE, its libraries are available for production, or deployment, or both. Servers that are installed during the IDE installation process are automatically registered in the IDE. You must register all other servers yourself.

**Tip:** You need to register a Java EE application server before you can begin developing enterprise applications.

When a server is registered with the IDE, you can see its node in the Services window under the Servers node. When you create a project in the New Project wizard, you can select the server to which you want to deploy your application. After you create the

application, you can change the server by right-clicking the project, choosing **Properties**, clicking **Run**, and selecting a different server.

### 23.1.3 Understanding Connection Pools

To store, organize, and retrieve data, most applications use relational databases. Java EE applications access relational databases through the JDBC API.

When accessing a database, the application calls on the following resources:

- **JDBC resource.** A JDBC resource (data source) provides applications with a means of connecting to a database. Typically, the administrator creates a JDBC resource for each database accessed by the applications deployed in a domain. (However, more than one JDBC resource can be created for a database.) Each JDBC resource has a unique JNDI name.
- **JDBC Connection pool.** A JDBC connection pool is a group of reusable connections that the application server maintains for a particular database. When an application closes a connection, the connection is returned to the pool. Connection pooling reduces the transaction time of connecting to a database by means of sharing connection objects providing access to a database source, thus avoiding a new physical connection being created every time a connection is requested.

At runtime, this is what happens when an application connects to a database:

1. **Lookup JNDI name of JDBC resource.** To connect to a database, the application looks up the JNDI name of the JDBC resource (data source) associated with the database. The JNDI API enables the application to locate the JDBC resource.
2. **Locate JDBC connection pool.** The JDBC resource specifies which connection pool to use. The pool defines connection attributes such as the database name (URL), user name, and password.
3. **Retrieve connection from connection pool.** The application server retrieves a physical connection from the connection pool that corresponds to the database. Now that the application is connected to the database, the application can read, modify, and add data to the database. Applications access the database by making calls to the JDBC API. The JDBC driver translates the application's JDBC calls into the protocol of the database server.
4. **Close connection.** When it is finished accessing the database, the application closes the connection. The application server returns the connection to the connection pool. Once it is back in the pool, the connection is available for the next application.

**Tip:** Each resource has a unique JNDI name which specifies its name and location. Because all resource JNDI names are in the `java:comp/env` subcontext, the JNDI name of a JDBC resource is expected in the `java:comp/env/jdbc` subcontext. For example, for a database name `MyDatabase`, specify `jdbc/MyDatabase`.

For information about using a connection pool, see [Section 16.3.6, "How to Access a Connection Pool from a Java Class."](#)

### 23.1.4 Common Application Server Tasks

This section contains information about tasks that are common to all application servers.

### 23.1.4.1 Setting Context Paths

When you execute a web application, the server uses a context path setting to derive the path to the web application. For example, if the context path is /directives, then you can access a file named index.html under the web application's document base (root directory) using the URL `http://host:port/directives/index.html`.

You should set a context path for a web application if you plan to execute different web applications on the same server. Otherwise, files with the same name will overwrite files from other web applications. When you create a web application from the New Project wizard, the default context path is derived from the name of the project.

#### To set a web application's context path:

1. In the Projects window, right-click the project node and choose **Properties** from the pop-up menu.
2. In the Project Properties dialog box, click **Run** and enter the Context Path. The path must begin with a forward slash (/), such as /directives.

**Tip:** When you change the Context Path in the Project Properties dialog box, the IDE updates the Tomcat Web Server's context descriptor (META-INF/context.xml) or the GlassFish application server's deployment descriptor (WEB-INF/glassfish-web.xml) to match.

When the server receives the HTTP request, the server selects the web application to be processed by matching the longest possible prefix of the Request URI against all of the defined context paths. A host can have an unlimited number of unique context paths.

### 23.1.4.2 Starting and Stopping Servers

You can start and stop a local server from the pop-up menu for the server in the Services window. Alternately, the server tab in the Output window has the buttons for controlling the server, shown in [Table 23–1](#).

**Table 23–1 Buttons Used to Control a Server**

Icon	Description
	Start the server
	Start the server in Debug Mode
	Restart the server
	Stop the server
	Refresh the server status

**Tip:** You cannot start and stop a remote installation of a server from within the IDE.

#### To start or stop a local server from the Services window:

1. Open the Services window by choosing **Window > Services** (Ctrl+5) from the main menu.

2. Expand the Servers node to display the registered server instances.
3. Right-click the server name and choose an action from the pop-up menu.

When you start a server, start up information is displayed in the server tab in the Output window.

**To start or stop a local server from the Output window:**

1. Choose **Window > Output** (Ctrl+4) from the main menu to open the Output window.
2. In the Output window, click the tab for the server.
3. Use the buttons in the left of the server tab to Start, Start in Debug Mode, Restart, or Stop.

If the Output window does not contain a tab for your server, you can start the server from the Services window. The server tab appears when the server is started.

#### 23.1.4.3 Removing Applications from Servers

When you no longer need to run an application that was added to the server using the IDE, you can remove the application's context from the server in the Services window.

By default, applications and modules are automatically registered, deployed and made available on the server when you run a project, and are automatically redeployed when they have been modified in the IDE. Once deployed, applications remain on the server until they are removed.

Deployed applications are visible in the Services window in the server instance node under a node for applications. When you undeploy an application, the archive file (ear, war, jar) is removed from the server and is no longer visible in the Services window.

**To remove an application from a server:**

1. Open the Services window by choosing **Window > Services** (Ctrl+5) from the main menu.
2. Expand the Servers node and ensure the server is running.  
If the server is not running, start the server by right-clicking the server node in the Services window and choosing **Start**. Alternately, click the **Start Server** button (▶) in the server tab in the Output window.
3. Expand the node of the server instance. If the server is not running, no subnodes are visible.
4. Expand the node containing the application you want to remove.
5. Right-click the application's context node and choose **Undeploy**.

#### 23.1.5 Developing on the Cloud

You can develop Java web applications locally on the IDE and deploy the final application to the cloud. The IDE supports deployment to:

- Oracle Cloud (requires Oracle Cloud plugin)
- Amazon Elastic Beanstalk

For more information, see [Section 23.7, "Working with Web Applications on the Cloud."](#)

### 23.1.6 HTTP Server-Side Monitor

The IDE provides the HTTP Server-Side Monitor to help diagnose problems with data flow from JSP page and servlet execution on the web server. The HTTP Server-Side Monitor gathers data about HTTP requests that are processed by the servlet engine.

For more information, see [Section 23.8, "Working with the HTTP Server-Side Monitor."](#)

## 23.2 Working with Web Browsers

The IDE uses a web browser to display web pages and run web applications. By default, it uses the default system browser but you can change this to use another browser. You can also configure the web browser to pass arguments to the executable when the browser is run.

### 23.2.1 How to Configure a Web Browser

You can:

- Change the browser which runs from the IDE.
- Add or remove browser instances from the list of those available in the IDE.
- Specify arguments to be passed to the executable when the browser is invoked.

#### **Configuring a web browser:**

1. Choose **Tools > Options** from the main menu.
2. Click the General category in the Options window.
3. Click **Edit** next to the Web Browser dropdown list to open the Web Browsers Manager.
4. Select the browser that you want to configure.
5. Enter the full path to the browser executable in the Process field. You can also click **Browse** to locate the executable on the local system.
6. Enter any arguments in the Arguments field.
7. Click **Close**.

### 23.2.2 How to Change the Default Web Browser

The IDE uses the specified default web browser to display web pages and run web applications. In the Options window you can choose any one of the supported browsers, such as the supported versions of Firefox and Internet Explorer. In addition, you can also configure the IDE to use other browser types.

#### **To change the web browser:**

1. Choose **Tools > Options** from the main menu and click the General category in the Options window.
2. Select a browser from the Web Browser dropdown list.

The default browser opens when you choose **Window > Web > Web Browser** in the main menu.

**Tip:** If you select a web browser and the web browser fails to open, confirm that the correct path to the browser executable is set. To modify the path to the executable or to add a browser to the Web Browser dropdown list, click the Edit button that is next to the dropdown list.

### Troubleshooting

When you set the IDE to use a proxy, your Internet browser may attempt to route all files through the proxy, including local files. This results in a File Not Found error when you try to open an HTML file or applet in the IDE's external browser. To resolve this problem, add localhost and your machine name to the list of bypassed hosts in your browser's proxy settings.

**Tip:** If the browser does not reflect changes in a JSP file or HTML file, this might be due to the way the browser caches pages. Check the browser's settings to ensure that the browser is set to reload pages each time you access them.

### 23.2.3 How to Open a Web Browser from the IDE

By default, the IDE opens your system's default web browser. If necessary, you can specify a different browser in the Options window. For more information, see [Section 2.2, "Working with the Options Window."](#)

#### To launch the web browser:

- Choose **Window > Web > Web Browser** from the main menu.

### Troubleshooting

If the IDE cannot find the operating system's default web browser, choose a different web browser. For more information, see [Section 23.2.2, "How to Change the Default Web Browser."](#)

### 23.2.4 How to Set a Proxy

If you want to use the Update Center or access the Internet through the IDE, but your system is behind a firewall or you use a proxy server, you need to configure the IDE with your proxy settings.

#### To use an HTTP proxy:

1. From the main menu, choose **Tools > Options** from the main menu and click **General** in the left pane of the Options window.
2. Select **HTTP Proxy** as the proxy type.
3. Type the proxy host name and the proxy port number.
4. Click **OK**.

#### To use a SOCKS proxy:

To run the IDE using a SOCKS proxy, you must pass the SOCKS proxy host and proxy port parameters to the JVM software when you start the IDE. On Microsoft Windows machines, use the `IDE-HOME/etc/netbeans.conf` file to pass the parameters. On UNIX and Linux machines, you can write a wrapper shell script.

**On a Microsoft Windows machine:**

1. In the IDE's installation directory, expand the etc directory.
2. If an netbeans.conf file is not there, create one.
3. Open netbeans.conf in a text editor and type:

```
-J-DsocksProxyHost=SOCKS-SERVER -J-DsocksProxyPort=1080
```

For SOCKS-SERVER, use the host name of your SOCKS proxy.

4. Save netbeans.conf and close it.
5. Restart the IDE.

**On a UNIX or Linux machine:**

1. Change directory to your *IDE-installation/bin* directory.
2. Create a new shell script file.
3. Open the new shell file in a text editor and type:

```
runide.sh -J-DsocksProxyHost=SOCKS-SERVER -J-DsocksProxyPort=1080
```

For SOCKS-SERVER, use the host name of your SOCKS proxy.

4. Save the shell script and close it
5. Restart the IDE with the new shell script.

## 23.3 Working with Glassfish Application Servers

The GlassFish application server is a Java EE compatible application server that supports JavaServer Pages (JSP), Java Servlet, and Enterprise JavaBeans (EJB) component-based applications.

Use this server and its libraries for the following activities:

- Deploying enterprise applications and enterprise beans.
- Deploying web services and web service clients that use JSR-109 or JSR-101 stubs.
- Deploying web applications, JSP pages, and servlets.

Libraries included with the GlassFish application server can be used to compile web services and clients, verify that your application implements the Java EE specification correctly, create a new web application with JavaServer Faces (JSF) support, or add JSF support to an existing web application.

**Tip:** To develop applications on the Java EE platform, you need to register an instance of the GlassFish Server.

After the GlassFish application server is registered with the IDE, the server instance is visible in the Servers node in the Services window. When the server is running, you can expand the server instance node to see the deployed applications and modules and modify the properties of resources on the server from within the IDE.

You can do the following directly through the server instance node:

- Undeploy applications
- Enable and disable applications
- Identify which applications are currently running

- Configure JDBC resources and other resource properties
- Access the GlassFish update center

When the GlassFish application server is running, you can expand the server instance node in the Services window to see the server resources and applications deployed to the server instance. For more information, see [Section 23.3.2, "How to View the Server in the IDE."](#).

### How to work with the GlassFish Server:

#### Task 1 Start the server

1. Register the GlassFish application server with the IDE. For more information, see [Section 23.3.1, "How to Register a Server Instance."](#)
2. In the Services window, right-click the server instance and select **Start**.

#### Task 2 Create and deploy enterprise applications

1. Create the project for the enterprise application. For more information, see [Section 14.1, "About Developing Enterprise Applications."](#)
2. Create the projects for any EJB modules. For more information, see [Section 16.2, "Creating an EJB Module Project."](#)
3. Create the projects for any web modules. For more information, see [Section 12.2, "Creating Web Application Projects."](#)
4. Add the modules to the enterprise application. For more information, see [Section 15.2.3, "How to add a module to an enterprise application."](#)
5. Deploy the enterprise application. For more information, see [Section 12.11, "Deploying a Web Application."](#)
6. Remove the application from the GlassFish application server. For more information, see [Section 23.1.4.3, "Removing Applications from Servers."](#)

#### Task 3 Configure the server

1. Add your database drivers. For more information, see [Section 24.3.1, "How to Add a JDBC Driver."](#)
2. Create a connection to a database. For more information, see [Section 24.3, "Setting up a Database Connection."](#)
3. Set up a connection pool for the database. For more information, see [Section 23.3.7, "How to Set up a Connection Pool."](#)
4. Register the connection pool. For more information, see [Section 23.3.11, "How to Register and Delete Resources."](#)
5. Create a JDBC resource for the connection pool. For more information, see [Section 23.3.8, "How to Set up a JDBC Resource."](#)
6. Register the JDBC resource. For more information, see [Section 23.3.11, "How to Register and Delete Resources."](#)

#### 23.3.1 How to Register a Server Instance

To begin developing enterprise applications you must first register an instance of the GlassFish Server with the IDE.

If you deploy your applications to a remote instance of the application server, those libraries are available at runtime. However, during development, to compile your project you need a local instance of this server and libraries.

You can register multiple domain instances and Domain Administration Server (DAS) instances of the GlassFish application server with the IDE, but you must first specify the local folder that contains the local installation of the server. You only need to specify the location of the local installation once.

#### To register an instance of the GlassFish Server:

1. Choose **Tools > Servers**, and in the Servers manager, click **Add Server** to open the Add Server wizard.

Alternatively, right-click the Servers node in the Services window and choose **Add Server**.

The Add Server wizard starts.

2. In the Choose Server pane:

- Select your application server from the list.
- Enter a name for the server instance. This name is used to identify the server in the IDE.
- Click **Next**.

3. In the Server Location pane, use **Browse** to navigate to and select the installation directory of your application server.

**Next** is only enabled when you have selected the correct installation directory.

4. Select the type of domain you are registering. If you are registering the local default domain, the available domains are listed in the drop-down list. If you are registering a remote domain, you need to specify the host and port used to communicate with the remote domain.

To define a local server instance:

- From the Local instances drop-down menu, select a server instance. When you select a server instance, the IDE updates **Domain path**, **Host**, and **Port**.
- Enter the **Username** and **Password**. If you are registering an instance of the server that was installed with the IDE, the default username is `admin`. There is no password.

5. Click **Finish** to complete the Add Server Instance wizard.

6. If you started the Add Server Instance wizard from the Server Manager, click **Close** in the Server Manager dialog box. The IDE displays a server node for your application server in the Services window under the Servers node.

After a server instance is registered with the IDE, a node for the server instance is visible in the Services window under the Servers node. You can perform the following tasks by right-clicking the server instance node and choosing a task from the pop-up menu:

- Start and stop a server instance
- Remove an instance of the server
- Open the server's admin console in a browser
- View the server log file

- View the properties of the server
- Contact the GlassFish Update Center to check for available software and server updates

**To remove an instance of the server:**

1. Choose **Tools > Servers**.
2. In the Servers manager, select the server name in the left pane.
3. Click **Remove**.
4. Click **Close**.

**Tip:** Alternatively, in the Services window right-click the server instance you want to remove and choose Remove from the pop-up menu.

### 23.3.2 How to View the Server in the IDE

In the Services window, each registered instance of the GlassFish application server is represented by a node under the Servers node in the Services window. The subnodes of the GlassFish instance node represent the resources and applications that have been deployed to the server.

Table 23–2 shows you how to use the nodes in the Services window to explore and work with the server instance. To delete a resource, locate and right-click the resource under the appropriate node and choose **Delete Resource**.

**Table 23–2 Nodes in the Services Window**

Icon	Description
	<b>GlassFish Application Server</b>  This node is visible under the Servers node () in the Services window and represents an instance of the GlassFish application server.  There is a corresponding server instance node for each instance registered with the IDE. For more information, see <a href="#">Section 23.3.1, "How to Register a Server Instance."</a>
	<b>Applications</b>  If the server instance is running (), you can expand this node to see the resources for the server instance. You can right-click the server instance node to stop, start and restart the server or remove the server instance. For more information, see <a href="#">Section 23.1.4.2, "Starting and Stopping Servers."</a>
	<b>Resources</b>  Expand this node to see the resources registered with the server.
	<b>JDBC resources</b>  Expand this node to see the JDBC resources and Connection Pools registered with the server.

**Table 23–2 (Cont.) Nodes in the Services Window**

Icon	Description
	<b>JavaMail Session</b> Expand this node to see the JavaMail Session resources registered with the server.
	<b>Connectors</b> Expand this node to see the Connector resources registered with the server.

### 23.3.3 How to Start and Stop the Server

You can quickly start or stop the server. For more information, see [Section 23.1.4.2, "Starting and Stopping Servers."](#)

**To start and stop the server:**

- In the Services window, right-click the GlassFish application server node and choose **Start / Stop Server**.  
Alternatively, click the **Start the Server** button in the server log tab in the Output window.

### 23.3.4 How to View the Server Log File

You can view the log files of a registered instance of the GlassFish application server directly in the IDE. Log files describe the events that occur on the server and can be helpful when troubleshooting your application.

**To view the log file:**

1. Expand the Servers node in the Services window of the IDE.
2. Right-click the server instance node and choose **View Server Log**.

When you choose **View Server Log**, the server log is displayed in the Output window of the IDE.

### 23.3.5 How to Access the Server Admin Console

The instance node for the GlassFish application server in the IDE Services window provides a subset of the functions found in the GlassFish Admin Console. You can manage server resources through the instance node in the IDE or by using the GlassFish Admin Console.

**To access the GlassFish Admin Console:**

1. Make sure the GlassFish application server is running. (Denoted by the running overlay icon (  ) next to the server's node in the Services window.)
2. Right-click the server instance node and choose **View Admin Console**.

The login screen for the Admin Console launches in your browser. After you log in, you can see that the options on the left are similar to those in the Services window in the IDE.

**Tip:** If you are prompted for a username and password, the default username is **admin** and there is no password.

When you manage resources directly through the Admin Console, the resources are not directly associated with your project.

### 23.3.6 How to Modify Server Properties

You can set and modify some of the properties of the GlassFish Server by modifying the properties panel for the server in the Servers manager. You can open the properties panel from the Services window.

#### To modify the server properties:

1. Ensure that the GlassFish instance is running.
2. In the Services window, right-click the GlassFish instance under the Servers node and choose Properties to open the Servers manager.

The GlassFish panel in the Servers manager contains a Common tab and a Java tab. In the Common tab you can modify properties related to the domain, username and deployment options.

3. Click the Java tab to edit the Java platform and debug settings for the server.  
You can start the server in debug mode from the Services window.
4. Specify the JDK platform that the server runs on.
5. Select one of the following debug settings.
  - **Socket** (default). Select this to use the default transport. When selected the address value refers to the Java Platform Debugger Architecture (JPDA) port number. The default address setting is 9009. You can modify the address value to change the JPDA port.

**Shared Memory:** On Windows, you can select this setting to use shared memory transport and then modifying the address property as necessary.

For more on JPDA connection settings, see  
<http://docs.oracle.com/javase/6/docs/technotes/guides/jpda/conninv.html>.

6. Enable or disable the use of IDE proxy settings. This setting is enabled by default. Click Close.

For more about Java Platform Debugger Architecture (JPDA), see  
<http://www.oracle.com/technetwork/java/javase/overview/index.html>.

### 23.3.7 How to Set up a Connection Pool

A JDBC connection pool is a group of reusable connections that the application server maintains for a particular database. Applications requesting a connection to a database obtain that connection from the pool. When an application closes a connection, the connection is returned to the pool.

Connection pooling reduces the transaction time of connecting to a database because connection objects providing access to a database source are shared, thus avoiding the creation of a new physical connection every time a connection is requested.

A connection pool is required if you want to create a JDBC resource. You can create connection pools for a project using the New File wizard.

#### To create a connection pool for your project:

1. Choose File > New (Ctrl+N).

2. Choose a project from the combo box.

**Tip:** Though the resource can be saved in either the EJB module or the Web module, Java EE patterns recommend saving the resource in the EJB module to allow greater flexibility and reusability.

3. Select **GlassFish** in the Categories pane, **JDBC Connection Pool** in the File Types pane, and click **Next**.
4. (required) Give the connection pool a name.
5. Select to either use an existing database connection or create a new connection by choosing a configuration from the list.
6. Select the XA checkbox if the connection is an XA transaction.
7. Click **Next**.
8. The Datasource classname is determined by the connection you selected.
9. Enter a description for the pool.
10. Enter any properties for the connection. You may need to enter the URL for the database and user name and password to connect.
11. Click **Next** if you want to modify the default properties for the connection. You can change these properties later by editing the `glassfish-resources.xml` file.
12. Click **Finish**.

Once you have created a connection pool in your project, the resource needs to be registered with the server to make it available to all applications and resources. The resource is automatically registered when the project is deployed.

### 23.3.8 How to Set up a JDBC Resource

A JDBC resource (data source) provides applications with a means of connecting to a database through a connection pool. Typically, there is at least one JDBC resource for each database accessed by an application deployed in a domain. It is possible to have more than one JDBC resource for a database.

**Tip:** A JDBC resource connects to a JDBC connection pool. If you have not already created the JDBC connection pool, you can create it in the wizard while you are creating the resource.

When you use **Enterprise Resources > Use Database** to generate lookup code for a database, the JDBC data sources are generated automatically.

#### To create a JDBC resource:

1. Ensure the GlassFish application server is running.
2. Go to **File > New (Ctrl+N)**.
3. In the Project drop-down list, choose either your EJB module or Web module.
4. Select **GlassFish** in the Categories pane, **JDBC Resource** in the File Types pane, and click **Next**.
5. Select an existing connection pool or create a new one.
6. (required) Enter a JNDI name for the resource.
7. Enable or disable the resource by choosing **True** or **False**.

8. Enter a description and click **Next**.
9. Enter any properties for the resource.
10. Click **Finish**.

**To prepare to use a data source:**

1. Ensure the GlassFish application server is running.
2. Create a web application and select the GlassFish Application Server as the target server.
3. Access the data source in, for example, a JSP page.

The resource is automatically registered when the project is deployed. When you register the resource, the resource is available to all applications and resources.

### 23.3.9 How to Set up a JMS Resource

The JMS API allows loosely coupled, reliable, asynchronous interactions among Java EE components and legacy systems capable of messaging. You can add new behavior to a Java EE application that has existing business events by adding a new message-driven bean to operate on specific business events. Java EE components that use the JMS API within EJB or web containers must have no more than one JMS session per JMS connection. For more information, see [Section 16.3.4, "How to Send JMS Messages."](#)

The JMS API uses two kinds of administered resource objects:

- **Connection Factories** (Connector Resource). These objects are used to create a connection to the JMS messaging system, and encapsulate connection parameters.
- **Destination Resources** (Admin Object Resource). These objects are specified as the targets and sources of messages. When creating a destination resource, you create two objects:
  - A physical destination
  - A destination resource that refers to the physical destination using the JNDI name

A JMS application normally uses at least one connection factory and at least one destination. The order in which the resources are created does not matter.

**To create a JMS resource:**

1. Go to **File > New (Ctrl+N)**.
2. In the Project drop-down list, choose either your EJB module or Web module.
3. Select **GlassFish** in the Categories pane, **JMS Resource** in the File Types pane, and click **Next**.
4. (required) Enter a JNDI name for the resource. JMS administered objects are usually placed within the `jms` naming subcontext (for example, `jms/MyMessageSource`).
5. Enable or disable the resource by choosing true or false.
6. Enter a description.
7. To create a destination resource, choose one of the following admin object resource types:
  - `javax.jms.Queue` - used for point-to-point communication

- javax.jms.Topic - used for publish-subscribe communication

To create a connection factory, choose one of the following connector resource types:

- javax.jms.QueueConnectionFactory - used for point-to-point communication
- javax.jms.TopicConnectionFactory - used for publish-subscribe communication
- javax.jms.ConnectionFactory - used for point-to-point communication

**Tip:** With the introduction of the JMS 1.1 specification, it is recommended that you use javax.jms.ConnectionFactory if you do not need to support existing code.

8. Click **Next**.
9. Enter properties for the resource, if any.
10. Click **Finish**.

### 23.3.10 How to Set Up a JavaMail Session

The JavaMail API is a set of abstract APIs that model a mail system. The API provides a platform-independent and protocol-independent framework to build mail and messaging applications. The JavaMail API provides facilities for reading and sending email. Service providers implement particular protocols.

The JavaMail API is implemented as a Java platform optional package and is also available as part of the Java EE platform.

#### To create a JavaMail Session resource:

1. Go to **File > New (Ctrl+N)**.
2. In the Project combo box, choose either your EJBModule or WebModule.
3. Select **GlassFish** in the Categories pane, **JavaMail Session** in the File Types pane, and click **Next**.
4. (required) Enter a JNDI name for the resource.
5. (required) In the mail host field, enter the DNS name of the default mail server.
6. (required) In the default user field, enter the user name to use when connecting to the mail server.
7. (required) In the **Default Return Address** field, enter the email address of the default user, in the form `username@host.domain`.
8. Enable or disable the resource by choosing `true` or `false`.
9. Enter a description.
10. In the Advanced area, change the field values only if the server's mail provider has been reconfigured to use a nondefault store or transport protocol. **Select True for Debug Enabled** for extra debugging output.
11. Click **Next**.
12. Enter any properties for the resource.
13. Click **Finish**.

### 23.3.11 How to Register and Delete Resources

Once you have created a resource object, the resource needs to be registered with the server. Once registered, the object is available to all the applications and resources.

Resources are automatically registered with the server when the project is deployed.

When you create a resource object for a project with the New File wizard, the resource name and properties are added to the file `glassfish-resources.xml`. You can edit `glassfish-resources.xml` in the Source Editor to modify resource properties.

To open `glassfish-resource.xml` in the Source Editor, double-click the file in one of the following places:

- In the Projects window, under the Server Resources node
- In the Files window, in the setup directory

---

**Note:** Alternatively, you can create resources in the Admin Console of the server. After you create a resource on the server, the resource will appear in the Services window.

---

After the object is registered with the server it is visible in the Services window under the Servers node under the node corresponding to the type of resource. You can modify some of the properties of the resource object by opening the properties window for the object.

**To delete a resource from the server:**

1. Locate the object you wish to delete in the Services window.
2. Right-click on the object and select **Delete Resource**.

When you delete a resource from the server it is no longer visible in the Services window.

---

**Notes:** ■If you create a resource in your project, removing the resource from the server does not delete it from your project. The resource is still defined in `glassfish-resources.xml` until you modify the file and remove it. If you do not remove the resource definition in `glassfish-resources.xml`, the resource will be created again when you deploy your project.

- The JNDI name for a resource must be unique. To change the JNDI name, open `glassfish-resources.xml` in the Source Editor and modify the name.
- 

### 23.3.12 How to Manage Users

You use the Admin Console of the GlassFish application server to add and delete users, change passwords and perform other user and group management tasks. You can access the Admin Console from the Services window. For more, see [Section 23.3.5, "How to Access the Server Admin Console."](#)

**To add a user:**

1. Open the Admin Console of the GlassFish application server.

2. In the tree in the left pane of the Admin Console, select **Configuration > Security > Realms > admin-realm**.
3. In the Edit Realm page, click **Manage Users**.  
Current user IDs are displayed in a table.
4. Click **New** and enter the new User ID and Password in the respective fields. Confirm the password by entering the same password again in **Confirm Password**.
5. Click **OK** to create the new user. If you made an error entering the password, a prompt appears telling you to reenter the password.
6. To continue managing users, click **Close** to return to the File Users page.

**To delete a user:**

1. Open the Admin Console of the GlassFish application server.
2. In the tree in the left pane of the Admin Console, select **Configuration > Security > Realms > admin-realm**.
3. In the Edit Realm page of the Admin Console, click **Manage Users** to display a table showing the Current user IDs.
4. In the User ID column, select the checkbox of the user you want to delete and click **Delete**.  
The user immediately disappears from the table of User IDs.
5. To continue managing users, click **Close** to return to the File Users page.

**To change a user's password:**

1. Open the Admin Console of the GlassFish application server.
2. In the tree in the left pane of the Admin Console, select **Configuration > Security > Realms > admin-realm**.
3. In the Edit Realm page of the Admin Console, click **Manage Users** to display a table showing the Current user IDs.
4. In the User ID column, select a user name, such as `admin`.  
The File Users page opens.
5. Enter the new password in **Password**. Confirm it by entering the password again in **Confirm Password**.
6. Click **Save** to have your password entries checked and saved. If you made an error, a prompt appears telling you to enter the password again.
7. To continue managing users, click **Close** to return to the File Users page.

### 23.3.13 How to Configure Security Roles

If you want to create secure areas of a web application, you need to configure the security roles by modifying the project's deployment descriptors. When configuring the security roles for your web application, you define your security roles in `web.xml`.

**To add a role:**

1. In the Security Roles section, click **Add**.

2. In the Add Security Role dialog box, enter the name and a description for the role and click **OK**.

**To edit a role:**

1. In the Security Roles section, select the role you want to edit and click **Edit**.
2. In the Edit Security Role dialog box, make your changes and click **OK**.

**To remove a role:**

- In the Security Roles section, select the role you want to delete and click **Remove**.

If the target server for your application is the GlassFish server, you need to edit `glassfish-web.xml` to map the security roles to the users and groups defined on the server. You map security roles by adding a principal or group to a security role. A security role can have more than one principal or group. You can use the IDE to help you edit `glassfish-web.xml` to map security roles.

**To map security roles:**

1. In the Projects window of the IDE, double-click `glassfish-web.xml` located in the Configuration Files directory of your web application project.
2. Click the Security tab in the visual editor.
3. Click Add Security Role Mapping to create a new security role.

**Tip:** The security roles are determined by the security roles defined in `web.xml`. If `web.xml` already defines a security role, the role is listed in `glassfish-web.xml`. For more on defining security roles in `web.xml`, see [Section 12.10.2, "How to Edit Deployment Descriptors."](#)

4. Expand the security role node to view the properties of the security role.
5. Click **Add Principal** or **Add Group** to open the Add Principal or Add Group dialog box.
6. In the dialog box, enter the name of the principal or group to add to the selected security role. The name of the principal or group must match a name specified on the GlassFish server.

For more details, see the following GlassFish Server documentation: "Managing Administrative Security".

### 23.3.14 How to Download and Install Server Updates

You can download and install add-ons and updates to the GlassFish server. You can also inspect installed components from the GlassFish Update Tool.

**To install server updates:**

1. In the Services window, right-click the GlassFish application server node and choose **View Domain Update Center**.
2. If the GlassFish Server 4.0 Update Center is not already installed you are prompted to install it. You can follow the installation process in the Output window.
3. In the GlassFish Update Tool, select the desired update from the available software and server updates.
4. Click **Install**.

## 23.4 Working with Oracle WebLogic Application Servers

Oracle WebLogic Server is a scalable, enterprise-ready Java Platform, Enterprise Edition (Java EE) application server. The Oracle WebLogic Server infrastructure supports the deployment of many types of distributed applications and is an ideal foundation for building applications.

### 23.4.1 How to Register a Server Instance

Before you can start developing and deploying applications to an application server, you must register your application server with the IDE.

---

**Note:** Oracle WebLogic Server must be installed locally on your machine even if a remote server is going to be used. A version of the locally installed WebLogic Server must be identical to the version of the remotely installed WebLogic Server. You can download Oracle WebLogic Server at

<http://www.oracle.com/technetwork/middleware/weblogic/downloads/index.html>.

---

#### To register an instance of an Oracle WebLogic Server:

1. Choose **Tools > Servers**, and in the Server Manager, click **Add Server**.

Alternatively, right-click the Servers node in the Services window and choose **Add Server** from the pop-up menu.

The Add Server Instance wizard starts.

2. In the Choose Server pane, do the following:
  - a. Select your application server from the list.
  - b. In **Name**, type a name for the server instance. This name is used to identify the server in the IDE.
  - c. Click **Next**.
3. In the Server Location pane, use **Browse** to navigate to and select the installation directory of your application server.

**Next** is only enabled when you have selected the correct installation directory.

4. Select **Local Domain** if you use WebLogic Server installed on your local machine or **Remote Domain** for a server running on a remote computer.
5. Click **Next**.
6. To define a local server instance, do the following:
  - a. From the Local instances drop-down menu, select a server instance. When you select a server instance, the IDE updates **Domain path**, **Host**, and **Port**.
  - b. Fill in **Username** and **Password**. The default username/password is `weblogic/weblogic1`.
7. To define a remote server instance, do the following:
  - a. In **Hostname**, type an IP address of the WebLogic Server instance.
  - b. Specify the port number in **Admin Port** and select the **SSL/TSL** option if data encryption is needed and provided by the server (the server has to be configured to listen for SSL).

- c. Select the **Server debug mode enabled** option if the remote WebLogic server instance runs in the debug mode and specify the port number where debugging is enabled in **Debug Port**.
  - d. Fill in **Username** and **Password**. The default username/password is weblogic/weblogic1.
8. Click **Finish**.
9. If you started the Add Server Instance wizard from the Server Manager, click **Close** in the Server Manager dialog box. The IDE displays a server node for your application server in the Services window under the Servers node.

### 23.4.2 How to View the Server in the IDE

In the Services window, each registered instance of the Oracle WebLogic application server is represented by a node under the Servers node in the Services window. The subnodes of the Oracle WebLogic instance node represent the resources and applications that have been deployed to the server.

**Table 23–3** shows you how to use the nodes in the Services window to explore and work with the server instance. To delete a resource, locate and right-click the resource under the appropriate node and choose **Delete Resource**.

**Table 23–3 Nodes in the Services Window**

Icon	Description
	<b>Oracle WebLogic Application Server</b>
	This node is visible under the Servers node ( ) in the Services window and represents an instance of the GlassFish application server. There is a corresponding server instance node for each instance registered with the IDE. For more information, see <a href="#">Section 23.4.1, "How to Register a Server Instance."</a>
	If the server instance is running ( ), you can expand this node to see the resources for the server instance. You can right-click the server instance node to stop, start and restart the server or remove the server instance. For more information, see <a href="#">Section 23.1.4.2, "Starting and Stopping Servers."</a>
	<b>Applications</b> Expand this node to see the modules and application contexts deployed to the server. You can disable, enable and remove an application context by locating and right-clicking the context in the Applications node.
	<b>Resources</b> Expand this node to see the resources registered with the server.
	<b>JDBC resources</b> Expand this node to see the JDBC resources and Connection Pools registered with the server.
	<b>JavaMail Session</b> Expand this node to see the JavaMail Session resources registered with the server.
	<b>Connectors</b> Expand this node to see the Connector resources registered with the server.

### 23.4.3 How to Start and Stop the Server

You can quickly start or stop the server in the Services window or from the Output window. For more information, see [Section 23.1.4.2, "Starting and Stopping Servers."](#)

**To start and stop the server:**

- In the Services window, right-click the Oracle WebLogic application server node and choose **Start / Stop Server**.

Alternatively, click the **Start the Server** button in the server log tab in the Output window.

## 23.5 Working with JBoss Application Servers

The JBoss Application Server supports JavaServer Pages (JSP), Java Servlets and Enterprise JavaBeans (EJB) component-based applications.

In the IDE, you can use this server for the following activities:

- Deploying enterprise applications and enterprise beans
- Deploying web services and web service clients
- Deploying web applications, JSP pages, and servlets

Once the JBoss Application Server is registered with the IDE, the server instance is visible in the Servers node in the Services window. For more information about what the icons represent, see [Section 23.1.4.2, "Starting and Stopping Servers."](#)

### 23.5.1 How to Register a Server Instance

Before you can start developing and deploying applications to an application server, you must register your application server with the IDE.

**To register an instance of a JBoss Application Server:**

1. Choose Tools > Servers, and in the Servers manager, click Add Server.

Alternatively, right-click the Servers node in the Services window and choose Add Server from the pop-up menu.

The Add Server Instance wizard starts.

2. In the Choose Server pane, do the following:

- a. Select your application server from the list.
- b. In the Name field, type a name for the server instance. This name is used to identify the server in the IDE.
- c. Click **Next**.

3. In the Server Location pane, use **Browse** to navigate to and select the installation directory of your application server.

**Next** is only enabled when you have selected the correct installation directory.

4. Click **Next**.

5. In the Instance Properties pane, from the Domain drop-down menu, select the JBoss Application Server domain you want to use for this server instance. The IDE automatically updates the Domain path field.

6. Click **Finish**.

7. If you started the Add Server Instance wizard from the Server Manager, click **Close** in the Server Manager dialog box. The IDE displays a server node for your application server in the Services window under the Servers node.

### 23.5.2 How to View the Server in the IDE

In the Services window, each registered instance of the JBoss application server is represented by a node under the Servers node in the Services window. The subnodes of the JBoss instance node represent the resources and applications that have been deployed to the server.

**Table 23–4** shows you how to use the nodes in the Services window to explore and work with the server instance. To delete a resource, locate and right-click the resource under the appropriate node and choose **Delete Resource**.

**Table 23–4 Nodes in the Services Window**

Icon	Description
	<p><b>JBoss Application Server</b></p> <p>This node is visible under the Servers node (  ) in the Services window and represents an instance of the JBoss application server.</p> <p>There is a corresponding server instance node for each instance registered with the IDE. For more information, see <a href="#">Section 23.5.1, "How to Register a Server Instance."</a></p>
	<p><b>Applications</b></p> <p>Expand this node to see the modules and application contexts deployed to the server. You can disable, enable and remove an application context by locating and right-clicking the context in the Applications node.</p>
	<p><b>Resources</b></p> <p>Expand this node to see the resources registered with the server.</p>
	<p><b>JDBC resources</b></p> <p>Expand this node to see the JDBC resources and Connection Pools registered with the server.</p>
	<p><b>JavaMail Session</b></p> <p>Expand this node to see the JavaMail Session resources registered with the server.</p>
	<p><b>Connectors</b></p> <p>Expand this node to see the Connector resources registered with the server.</p>

### 23.5.3 How to Start and Stop the Server

You can quickly start or stop the server in the Services window or from the Output window. For more information, see [Section 23.1.4.2, "Starting and Stopping Servers."](#)

#### To start and stop the server:

- In the Services window, right-click the GlassFish application server node and choose **Start / Stop Server**.

Alternatively, click the **Start the Server** button in the server log tab in the Output window.

### 23.5.4 How to Set up a Connection Pool

A JDBC connection pool is a group of reusable connections that a web server or application server maintains for a particular database. Applications requesting a connection to a database obtain that connection from the pool. When an application closes a connection, the connection is returned to the pool. Connection pool properties may vary with different database vendors. Some common properties are the URL for the database name, user name, and password.

The first step in creating a database connection pool on the JBoss Application Server is to create JDBC resource (also called a data source). A JDBC resource provides applications with a connection to a database. Typically, there is at least one JDBC resource for each database accessed by an application deployed in a domain. It is possible to have more than one JDBC resource for a database. You can create a JDBC resource manually in a file tailored to your database server, provided in the JBoss installation directory.

#### To set up a data source on the JBoss Application Server:

1. Open the Favorites window (Ctrl+3).
2. Right-click in the window, choose **Add to Favorites**, and browse to the JBoss installation directory's docs/examples/jca folder.
3. Open the -ds.xml file of your choice in the editor. For example, if MySQL is your database server, double-click mysql-ds.xml.
4. Define the data source. For example, for PointBase, the data source could be similar to the following:

```
<datasources>
  <local-tx-datasource>
    <jndi-name>MySqlDS</jndi-name>
    <connection-url>jdbc:mysql://mysql-hostname:3306/jbossdb</connection-url>
    <driver-class>com.mysql.jdbc.Driver</driver-class>
    <user-name>x</user-name>
    <password>y</password>

    <exception-sorter-class-name>org.jboss.resource.adapter.jdbc.vendor.MySQLExceptionSorter</exception-sorter-class-name>
    <metadata>
      <type-mapping>mySQL</type-mapping>
    </metadata>
  </local-tx-datasource>
</datasources>
```

5. Copy the file to your JBoss deployment directory. For example, if default is your domain, copy the file to server/default/deploy.

#### To reference a data source from a web application:

1. In the WEB-INF/jboss-web.xml file, add a resource reference. For example, for the data source above, the resource reference could be as follows:

```
<resource-ref>
  <res-ref-name>MySqlDS</res-ref-name>
  <jndi-name>MySqlDS</jndi-name>
</resource-ref>
```

2. In the WEB-INF/web.xml file, add a resource reference. For example, for the data source above, the resource reference could be as follows:

```

<resource-ref>
    <res-ref-name>MySqlDS</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
    <res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>

```

**To prepare to use a data source on the JBoss Application Server:**

1. Register the JBoss Application Server. For more information, see [Section 23.5.1, "How to Register a Server Instance."](#)
2. Set the JBoss port number in its server.xml file. By default, the port number is 8080. If you are using the default domain, the server.xml file is found here:  
`\server\default\deploy\jbossweb.sar\server.xml`
3. Create a web application and select JBoss Application Server as the target server. For more information, see [Section 23.1.4.2, "Starting and Stopping Servers."](#)
4. Access the data source in, for example, a JSP page.

### 23.5.5 How to Access Application Server Utilities

The JBoss Application Server instance node in the IDE provides access to the JBoss Management Console, JMX Console, and server log. You can manage server resources, such as deployed applications, through these utilities.

**To access the JBoss Application Server's utilities:**

1. In the Services window, expand the Servers node.
2. Start the server by right-clicking the JBoss Application Server instance node and choosing **Start**.
3. After the server starts, right-click the node and choose **View Admin Console**, **View JMX Console**, or **View Server Log**.

## 23.6 Working with Tomcat Web Servers

The IDE provides an IDE module that handles integration with Tomcat Web Servers. A Tomcat Web Server is pre-configured to work with the plugin. The Tomcat Web Server that is bundled with the IDE implements the JavaServer Pages 2.2 and Servlet 3.0 specifications and includes many features that are useful in the deployment of web applications.

Instead of using the bundled Tomcat Web Server, you can register a supported Tomcat Web Server with the IDE and then deploy to it. The IDE supports Tomcat 8, Tomcat 7 and TomEE.

To see the Tomcat Web Servers that are registered with the IDE, expand the **Servers** node in the Services window.

For more information, see the following web sites:

- Extensive information about the Tomcat Web Server internals is available at <http://tomcat.apache.org/>.
- The JavaServer Pages Specification is available at <http://www.oracle.com/technetwork/java/javaee/jsp/index.html>.

- The JavaServer Servlet Specification is available at  
<http://www.oracle.com/technetwork/java/javasee/servlet/index.html>.

When you download and install a version of the IDE that contains Java EE technologies, you can choose to install and register the Tomcat Web Server during the installation process.

The configuration files for the bundled Tomcat Web Server are in the Tomcat base (Catalina base) directory, located under your IDE user directory. The Tomcat home (Catalina home) directory is under the IDE installation directory. Note that this setup differs from a typical Tomcat installation, where the base directory and home directory coincide. To see the exact locations of the home and base directories, right-click the server's node in the Services tab and choose Properties from the popup menu.

**Tip:** If the Tomcat base (Catalina base) folder does not exist, it is created the first time the server is started.

Shared libraries that are used by web applications must be added to the home directory. Do not add shared libraries to the base directory.

---

**WARNING:** Developers using a common IDE installation need to exercise caution when adding shared libraries to the home directory. Libraries stored in the home directory are shared by all web applications that are deployed on the server, so the potential for version conflicts exists.

---

On the UNIX platform, the IDE is sometimes installed in a read-only directory. If this is the case, you cannot add shared libraries to the bundled Tomcat installation. Ask the person responsible for the IDE installation to make writable all directories under the Tomcat home directory. Alternatively, you could add another Tomcat Web Server and configure it to meet your needs.

### 23.6.1 How to Register a Server Instance

When you download and install a version of the IDE that includes Java EE technologies, you can choose to install and register the Tomcat Web Server during the installation process. You can also configure the IDE to deploy web applications to other installations of supported versions of the Tomcat Web Server. These Tomcat Web Server installations must first be registered. Unsupported versions of the Tomcat Web Server cannot be registered with the IDE.

#### To register an instance of the Apache Tomcat web server:

1. Make sure the server is not running. If the server is running and there is a port number conflict with other installed servers, the IDE will not be able to resolve the conflict.
2. Choose **Tools > Servers**.  
Alternatively, right-click the Servers node in the Services window and choose **Add Server**.
3. Click **Add Server** to display the Add Server Instance wizard.
4. Select the version of the Tomcat web server that you want to register. Click **Next**.
5. Specify the server installation location and login properties in the Add Server Instance wizard. Click **Finish**.

When you click **Finish**, the IDE adds a node for the server instance under the Servers node in the Services window. You can place the cursor over the server's node to display a tooltip that contains the server's port number.

**To set the username and password for Tomcat Web Server:**

1. In the Services window, right-click the Servers node and choose Add Server. The Add Server Instance dialog box opens.
2. Select the applicable server and click **Next**.
3. Browse for the Catalina Home directory. You can also choose to use a private configuration folder.
4. Enter the username and password you want to use for the server instance. If the user does not exist, click the checkbox to create the user.
5. Click **Finish** to add the server instance. The new username and password is added to the `tomcat-users.xml` file.

**To get the username and password from your user directory:**

1. Select **Help > About** from the main menu.
2. Make a note of the path to the `Userdir` folder.
3. In your file system, go to your `Userdir` and then to the `apache-tomcat-8.0.x_base\conf` folder.
4. Open the `tomcat-users.xml` file and make a note of the password that is defined for the "ide" username.

---

**Note:** The username and password are not saved when you enter them in this dialog box. The username and password are only saved if you define them in the Tomcat Web Server's Properties dialog box. To open the Properties dialog box, right-click the Tomcat Web Server's instance node in the Services window and select Properties.

---

**To remove an instance of the Tomcat Web Server:**

1. Choose **Tools > Servers** from the main menu.
2. In the Servers manager, select the Tomcat Web Server from the list of servers in the left pane.
3. Click **Remove Server**.
4. Click **Close**.

### 23.6.2 How to View the Server in the IDE

The Servers node in the Services window and its subnodes represent the Tomcat Web Server, its instances, and the contexts that have been deployed by an instance to the server. If you right-click a node and display its popup menu, you see a list of menu items that enable you to work with this node.

**To view the Tomcat Web Server in the IDE:**

1. Choose **Window > Services** (Ctrl+5) from the main menu. The Services window appears.
2. Expand the Servers node to view the Tomcat Web Server nodes.

- The **Servers** node shows all the servers that are registered in the IDE, such as the bundled Tomcat Web Server.
- Each physical installation of a Tomcat Web Server is represented by a **Tomcat Server** instance node.
- The **Web Applications** node lists the web application contexts that have been deployed to the server. It appears under the server's instance node in the Services window. The IDE adds a web application's context node the first time the web application is deployed from the IDE.

In the Projects and Files windows, the Tomcat **context descriptor node** (  ) reflects the context.xml file that contains the web application's context element. The Tomcat context descriptor node appears under the web application's META-INF node. When Tomcat is set as the target server, the IDE creates the META-INF directory and the Tomcat context descriptor when you create, open, or import a web application, unless one already exists.

The context.xml file is used for setting a web application context path and for advanced web application configuration. For simple web applications, you do not need to edit the context.xml file because you use the WEB-INF property sheet to set the context path and the deployment descriptor (web.xml) to configure the web application. The IDE maintains the context element's path attribute automatically when you edit the Context Path in the WEB-INF properties sheet. If you update the context path in one place, the IDE updates the value in the other place.

You can use the context.xml file to make the following settings:

- Servlet context log files
- JNDI resources and resource parameters (JDBC data source)
- Web application context parameters and environment entries

The context.xml file represents the server.xml context element. It is recommended that the context information is specified in a standalone context.xml file in the META-INF directory. For more information about the context.xml configuration, see: <http://tomcat.apache.org/tomcat-7.0-doc/config/context.html>.

#### To open the Tomcat Web Application Manager

1. In the Services window, right-click the **Tomcat Web Server** node and choose **Start**.
2. Expand the **Web Applications** node and right-click the node for the /manager context and choose **Open in Browser**.
3. Log in using a valid username and password.

#### To undeploy an application

1. In the Services window, expand the Tomcat Web Server instance and the **Web Applications** node in the Services window.
2. Expand the node which contains the type of application to undeploy.
3. Right-click the application and choose **Undeploy** from the contextual menu.

### 23.6.3 How to Start and Stop the Server

The server starts automatically when you deploy a web application. For more information, see [Section 23.1.4.2, "Starting and Stopping Servers."](#)

**To start the server manually:**

1. In the Services window, expand the Servers node.
2. Right-click the Tomcat Web Server node and choose **Start** or **Restart**.

**To stop the server:**

1. In the Services window, expand the Servers node.
2. Right-click the Tomcat Web Server node and choose **Stop**.

**23.6.4 How to Configure the Server Properties**

You can configure properties and customize the server.

The Tomcat node in the Services window represents a physical installation of a Tomcat Web Server. When you choose the install Tomcat with the IDE, the installer automatically adds an instance for the bundled Tomcat Web Server. You can register additional Tomcat Web Servers using the Add New Server dialog box. See [Section 23.6.1, "How to Register a Server Instance."](#)

If you right-click the Tomcat node and display the contextual menu, you see a list of menu items that enable you to work with this node. The following list describes some of these menu items:

- **Start.** Starts the server.
- **Start in Debug Mode.** Starts the server in debug mode.
- **Start in Profile Mode.** Starts the server in profile mode.
- **Restart.** Stops and restarts the server.
- **Stop.** Stops the server.
- **Refresh.** Shows the status (started or stopped) of the server.
- **Remove.** Removes the server from the Servers node.
- **Edit server.xml.** Opens the server.xml file in the Source Editor. It is recommended to edit this file using the Tomcat Manager. Only advanced users should edit this file in the Source Editor.
- **View Server Output.** Displays the server log file as defined in the server.xml file. This item is enabled when the server is running. If no logger is defined in the context.xml file, logging for that context is performed in this shared context log.
- **Properties.** Opens the Tomcat Web Server instance's Properties dialog box.

**To customize the server:**

1. In the Services window, expand the Servers node to view the Tomcat Web Server nodes.
2. Right-click a Tomcat Web Server's instance and select **Properties** from the popup menu.

The Servers manager opens and displays the following information:

**Connection Tab**

- **Catalina Home.** Specifies the location of the server installation.
- **Catalina Base.** Specifies the base directory for the Tomcat Web Server. You can set this property when you add a Tomcat Web Server to the Server Manager. If you set a base directory, then that is where the Tomcat configuration files reside. One

reason for specifying a separate base directory is to allow more than one user to use the same server. If no base directory is specified, then the configuration files reside in the home directory.

- **Credentials of an existing user in the "manager" role.**
  - **Username.** Specifies the user name that the IDE uses to log into the server's manager application. The user must be associated with the manager role. The first time the IDE starts the Tomcat Web Server, such as through the Start/Stop menu action or by executing a web component from the IDE, the IDE adds an admin user with a randomly-generated password to the `tomcat-base-path/conf/tomcat-users.xml` file. (Right-click the Tomcat Web Server instance node in the Services window and select **Properties**. In the Properties dialog box, the Base Directory property points to the `base-dir` directory.) The admin user entry in the `tomcat-users.xml` file looks similar to the following:

```
<user username="ide" password="woiehh" roles="manager"/>
```
  - **Password.** Specifies the user's password. See the explanation for the Username property above for details.
- **Server Port.** Specifies the number of the TCP/IP port that the server listens on. The URL for a web application that is deployed on the server is derived from the host's URL and the server port, such as `http://localhost:8086`.
- **Shutdown Port.** Specifies the port number on which the server waits for a shutdown command.
- **Enable HTTP Monitor.** If selected, this checkbox enables the HTTP Monitor for web applications executing on the Tomcat Web Server. When enabling the monitor, you must restart the server in order for the change to take affect. The HTTP Monitor is useful for debugging web applications. If you are using the server for production purposes only, you may want to clear the checkbox to reduce its impact on the server's performance. When you clear the checkbox, the IDE removes the HTTP monitor's declaration from the server. However the monitor's libraries remain deployed to the server. When you disable the HTTP Monitor, you must restart the server for the change to take affect.

### Startup Tab

- **Use Custom Catalina Script.** When selected the server is started using a custom a Catalina script. Click **Browse** to specify the location of the custom script. This is deselected by default.
- **Force Stop.** Specifies whether to use the force shutdown (shutdown is followed by the kill command). This is unchecked by default. This function is disabled on Windows.
- **Debugger Transport.** Enables you to select either shared memory or socket based transport for debugging.
  - **Shared Memory Name.** When selected, the transport is set to `dt_shmem`. You can type a shared memory name or use the default name. This option is selected by default on Windows.
  - **Socket Port.** When selected the Java Platform Debugger Architecture (JPDA) transport is set to `dt_socket` and the JPDA address is set to the specified socket port number when the server is started in debug mode.

The Shared Memory transport option is only available on Windows. If you want to use the JPDA socket transport for debugging on Windows, select **Socket Port** and specify the JPDA port number.

#### **Deployment Tab**

- **Deployment Timeout(s).** Specifies the length of server timeouts.
- **Enable JDBC driver deployment.** Specifies whether to enable JDBC driver deployment. This is checked by default.

#### **Classes Tab**

- Lists all classes associated with the server instance.

#### **Sources Tab**

- Lists all sources associated with the server instance.

#### **Javadoc Tab**

- Lists all Javadoc associated with the server instance.

### **23.6.5 How to Edit the Configuration File**

The main configuration file for Tomcat is the `server.xml` file. It provides configuration information for Tomcat components and specifies deployment information for the server. For simple web applications, editing the `server.xml` file is not necessary because the IDE makes the necessary changes automatically. Most edits to the server configuration file can be achieved by editing the properties for the Tomcat Web Server node, or by using menu commands in the Projects window, Files window, and Services window.

Editing is usually necessary only for advanced configuration, such as setting up authentication based on JNDI or database lookups. For more information about configuring Tomcat, see

<http://tomcat.apache.org/tomcat-7.0-doc/config/index.html>.

The `server.xml` file can be in one of two places. Typically, you edit the `server.xml` file that is in the `tomcat-installation-dir/conf` directory. However, if the server node's property sheet shows a base directory path that is different from the home directory path, then you must edit the `server.xml` file that is in the `base-dir/conf` directory.

#### **To edit the `server.xml` file in the Source Editor:**

---

**WARNING:** You can edit the `server.xml` file that is in the `base-dir/conf` directory in the Source Editor. It is recommended that only advanced users edit the `server.xml` file in the Source Editor.

---

1. In the Services window, expand the Servers node and the Tomcat Servers node.
2. Stop the server by right-clicking the Tomcat instance node and choosing **Stop**.
3. Right-click the Tomcat instance node and choose **Edit server.xml** from the contextual menu to open the `server.xml` file in the Source Editor.

### 23.6.6 How to Authenticate the Server

It would not be safe to ship application servers and web servers with default settings that allowed anyone on the Internet to execute them on your server. Therefore, servers are shipped with the requirement that anyone who attempts to use them must authenticate themselves, using a username and password with the appropriate role associated with them.

For example, the Tomcat Web Server uses the Tomcat Manager to run web applications. To authenticate yourself when you run your web application, JSP file, or servlet, you need a username and password for a user with the "manager" role. This username and password are defined in the `tomcat-users.xml` file that is in your user directory.

**Tip:** You can only use the usernames and passwords that are defined in the IDE user directory's `tomcat-users.xml` file, and not those that are in the IDE installation directory's `tomcat-users.xml` file. Click **show** to display the password for a username created while registering the server with the IDE.

During installation, the IDE generates a user called `ide` for use with the bundled Tomcat Web Server. This user is assigned the "manager" role and is created in your user directory's `tomcat-users.xml` file.

#### To set the username and password for the Tomcat Manager:

1. In your system, go to Tomcat's base directory and then to its `\conf` subfolder.

**Tip:** If you don't know where Tomcat's base directory is, right-click the Tomcat Web Server instance node in the Services window and select **Properties**. In the Server Manager, the Catalina Base Directory points to the base directory.

The `tomcat-users.xml` file in Tomcat's base directory contains instructions for creating user roles. If necessary, define the necessary role, save the file and stop and restart the Tomcat instance in the Services window.

2. Open the `tomcat-users.xml` file and make a note of the password that is defined for the `ide` username.
3. Right-click the Tomcat Web Server instance node in the Services window and select **Properties**. In the Server manager, type the username and password.

**Tip:** If the username and password have not been specified, a dialog box appears when you run a web application, JSP file, or servlet. If you specify the username and password in this dialog box, they are not saved. The username and password are only saved if you define them in the Tomcat Web Server instance's Properties dialog box.

For details on using the security manager, see the following document.

<http://tomcat.apache.org/tomcat-7.0-doc/security-manager-howto.html>

### 23.6.7 How to Set up a Connection Pool

A JDBC connection pool is a group of reusable connections that a web server or application server maintains for a particular database. Applications requesting a

connection to a database obtain that connection from the pool. When an application closes a connection, the connection is returned to the pool. Connection pool properties may vary with different database vendors. Some common properties are the URL for the database name, user name, and password.

For detailed information on Tomcat's database connection pooling functionality, see <http://commons.apache.org/proper/commons-dbcpc/>.

The first step in creating a Tomcat database connection pool is to create JDBC resource (also called a data source). A JDBC resource provides applications with a connection to a database. Typically, there is at least one JDBC resource for each database accessed by an application deployed in a domain. It is possible to have more than one JDBC resource for a database. You can create a JDBC resource manually in your `server.xml`.

#### To add a JDBC resource manually in the `server.xml` file:

---

**WARNING:** Be aware that you hand-edit the `server.xml` file at your own risk; the IDE cannot repair a damaged `server.xml` file. You are strongly encouraged to create a backup version of your working `server.xml` file before beginning to edit by hand.

---

1. In the Services window, expand the Servers node and the Tomcat Servers node.
2. Stop the server by right-clicking the Tomcat instance node and choosing **Stop**.
3. Right-click the Tomcat instance node and choose **Edit server.xml** from the contextual menu to open the `server.xml` file in the Source Editor.
4. Make your changes.
5. Reference the JDBC resource from your web application as described below.

#### To reference a JDBC resource from a web application:

1. Expand the project node in the Projects window. Then expand the Web Pages node and the WEB-INF node. Double-click the `web.xml` node and use the Source Editor to add your resource reference to the `web.xml` file as follows:

```
<resource-ref>
  <description>Tomcat DBCP</description>
  <res-ref-name>jdbc/poolDB</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
</resource-ref>
```

2. Expand the META-INF node. Right-click the `context.xml` node, choose **Edit** from the contextual menu and use the Source Editor to add the following resource link between the `<context>` tags in the `context.xml` file:

```
<ResourceLink name="jdbc/poolDB" type="javax.sql.DataSource"
  global="jdbc/poolDB"/>
```

---

**Note:** Do not double-click the context.xml file. If you do so, the IDE opens the context.xml file in the Context Editor instead of the Source Editor. You cannot add a resource link in the Context Editor. As you cannot open the context.xml file in both the Source Editor and the Context Editor at the same time, the IDE disables **Edit** in the contextual menu if the context.xml file is opened in the Context Editor.

---

Your web application's META-INF/context.xml file should now look similar to the following:

```
<Context path="/Employees">
<ResourceLink name="jdbc/poolDB" type="javax.sql.DataSource"
    global="jdbc/poolDB"/>
<Logger className="org.apache.catalina.logger.FileLogger"
    prefix="Employees" suffix=".log" timestamp="true"/>
</Context>
```

3. Finally, use the JDBC resource in your web application.

## 23.7 Working with Web Applications on the Cloud

It is recommended to develop web applications locally and deploy the final application to the cloud, although it is possible to develop web applications directly on cloud-based servers. Developing applications locally has the following advantages:

- Local deployment takes seconds on a running server. Deployment to the Cloud can take minutes.
- Incremental deployment is only available locally.
- Debugging is only available locally.

---

**Note:** To locally develop applications that you will later deploy to the cloud, your local environment must mirror the environment on the cloud.

---

When your local environment is set up, develop web applications as usual. When the application is complete, deploy it to the cloud environment.

The following task list shows how to develop and deploy to the Cloud.

### Task 1 Create an account

- Create an account with the cloud provider.

### Task 2 Register your cloud account in the IDE

1. Open the Services window.
2. Right-click the Cloud node.
3. Select **Add Cloud**.
4. Enter your cloud account details.

For more information, see [Section 23.7.2, "How to Register a Cloud Account in the IDE."](#)

### Task 3 Develop your application locally

1. Set up your local environment (application and database server) to match the cloud environment.
2. Create the web application in the local environment, selecting a local server that matches the cloud environment.
3. Test, debug, and tweak your application using NetBeans IDE tools.

### Task 4 Deploy and run the application on the cloud

1. In the Projects window, right-click the application's node and select **Properties**.
2. Select the Run category in the project's Properties.
3. Select a cloud-based server from the Server drop-down list.(If you registered a cloud account with NetBeans IDE, the servers associated with that account are listed.)
4. In the Projects window, right-click the project's root node and select **Run**. The IDE builds the project and deploys it to the cloud, and the application's landing page opens in a browser window.

#### 23.7.1 About Web Applications on the Cloud

NetBeans IDE supports the development of Java web applications on cloud-based servers. The IDE currently supports the following clouds:

- Oracle Cloud (requires Oracle Cloud plugin)
- Amazon Elastic Beanstalk

Cloud accounts can be registered in the Services window under the Cloud node.

After you register a cloud account in the IDE, the server associated with that cloud account appears in the Services window under the Servers node. You can treat a server in the cloud like you would treat any other server registered in the IDE.

You cannot create or administer a cloud account from inside the IDE. You must go to the cloud provider's web page to create or administer an account.

For more details about working with cloud providers supported in the IDE, see:

- Running Web Applications on Oracle Cloud on the NetBeans site at <https://netbeans.org/kb/docs/web/oracle-cloud.html>.
- NetBeans wiki: AmazonBeanstalkSupport on the NetBeans site at <http://wiki.netbeans.org/AmazonBeanstalkSupport>.

#### 23.7.2 How to Register a Cloud Account in the IDE

To develop and deploy applications to cloud-based servers you need to register your cloud account with the IDE.

##### To register your cloud account:

1. If necessary, open the Services window (Ctrl+5).
2. Right-click the Cloud node in the Services window and select **Add Cloud** to open the Add Cloud Provider wizard.
3. Select a cloud provider and click **Next**.
4. Specify the details of your cloud account. Click **Next**.

When you click **Next** the IDE attempts to communicate with your cloud account. If communication is successful a list of the resources associated with your cloud account is displayed in the wizard.

**5. Click **Finish**.**

After registering a cloud account, the web/application servers associated with the account appear in the Services window under the Servers node.

### 23.7.3 How to Develop Cloud Applications Locally

You are recommended to develop web applications locally and deploy the final application to the cloud, although you can develop web applications directly on cloud-based servers. Developing applications locally has the following advantages:

- Local deployment takes seconds on a running server. Deployment to the Cloud can take minutes.
- Incremental deployment is only available locally.
- Debugging is only available locally.

To locally develop applications that you will later deploy to the cloud, your local environment must mirror the environment on the cloud.

When your local environment is set up, develop web applications as usual. When the application is complete, deploy it to the cloud environment.

For more details about working with cloud providers supported in the IDE, see:

- Running Web Applications on Oracle Cloud on the NetBeans site at <https://netbeans.org/kb/docs/web/oracle-cloud.html>.
- AmazonBeanstalkSupport on the NetBeans wiki at <http://wiki.netbeans.org/AmazonBeanstalkSupport>.

### 23.7.4 How to Deploy Web Applications to the Cloud

To deploy a web application to the cloud, set the application's server to a server on the cloud and run the application.

**To deploy your application:**

1. In the Projects window, right-click the application's node and select **Properties**.
2. Select the Run category in the project's Properties window.
3. Select the target server from the Server drop-down list and click **OK**.

The remote cloud server will be listed in the drop-down list if the cloud account is registered in the IDE.

Alternatively, you can select a cloud server when you create the web application and develop it entirely on the cloud. However this is not recommended. See [Section 23.7.3, "How to Develop Cloud Applications Locally."](#)

4. In the Projects window, right-click the project's root node and select **Run**.

When you click Run the IDE builds the project and deploys it to the cloud server. You can follow the progress of deployment in the IDE's Output window. The application's landing page opens in a browser window.

## 23.8 Working with the HTTP Server-Side Monitor

The IDE provides the HTTP Server-Side Monitor to help diagnose problems with data flow from JSP page and servlet execution on the web server. The HTTP Server-Side Monitor gathers data about HTTP requests that are processed by the servlet engine. For each HTTP request that is processed, the monitor records data about the incoming request and the data states maintained on the server.

You can analyze your HTTP requests, store request records for future sessions, and replay and edit previous HTTP requests. HTTP request records are stored until you exit the IDE, unless you explicitly save them.

Additional tools are the profiler and the debugger. For more information, see:

- [Chapter 9, "Testing and Profiling Java Application Projects"](#)
- [Chapter 10, "Running and Debugging Java Application Projects"](#)

The following task list shows how to work with the HTTP Server-Side Monitor.

### Task 1 Set up the HTTP Server-Side Monitor

1. Ensure that the HTTP Server-Side Monitor is configured correctly.
2. Ensure that the HTTP Server-Side Monitor is enabled for the web server.

By default, the HTTP Server-Side Monitor starts when you deploy to the Tomcat Web Server. For the Glassfish application server, go to the Services window (Ctrl+5), expand the Servers node, right-click the server's node, choose **Properties**, and select the **Enable HTTP Monitor** checkbox.

### Task 2 Analyze Session Data

1. Run the web application.
2. Use the HTTP Server-Side Monitor to analyze session data. For more information, see [Section 23.8.2, "How to Analyze Session Data."](#)

### Task 3 Optional

- Save session data. For more information, see [Section 23.8.3, "How to Save Session Data."](#)
- Edit and replay session data. For more information, see [Section 23.8.4, "How to Replay Session Data."](#)

### 23.8.1 How to Set up the HTTP Server-Side Monitor

The HTTP Server-Side Monitor is enabled and displayed by default when you deploy to the Tomcat Web Server. For the Glassfish application server, you must enable it manually. If the HTTP Server-Side Monitor does not display or if it does not show the session data for a running web application, ensure that it is enabled as described below.

#### To manually display the HTTP Server-Side Monitor:

- Choose Window > Debugging > HTTP Server-Side Monitor from the main menu.

#### To enable the HTTP Server-Side Monitor for servers that are started from the IDE:

1. Right-click your server's node in the Services window's Servers node.
2. Choose **Properties**.

3. Select **Enable HTTP Monitor**.
4. If the server is running, stop and restart it.

The HTTP Server-Side Monitor is now enabled and will appear when you deploy your application.

**To enable the HTTP Server-Side Monitor for servers that are started outside the IDE:**

1. Copy the following two files to your web application's WEB-INF/lib folder.
  - *IDE-install-directory/enterprise/modules/ext/org-netbeans-modules-web- httpmonitor.jar*
  - *IDE-install-directory/ide/modules/org-netbeans-modules-schema2beans.ja r*

The Glassfish application server has its own schema2beans.jar. Therefore, when you are using the Glassfish application server, only copy httpmonitor.jar to your web application's WEB-INF/lib folder.

2. Add the filter declaration that is appropriate for your servlet's version to the top of your web application's WEB-INF/web.xml file. Filters and filter mapping entries must come first in a deployment descriptor.

- Servlet 2.4 Filter Declaration:

```
<filter>
    <filter-name>HTTPMonitorFilter</filter-name>

    <filter-class>org.netbeans.modules.web.monitor.server.MonitorFilter</filter
    -class>
        <init-param>
            <param-name>netbeans.monitor.ide</param-name>
            <param-value>name-of-host-that-runs-IDE:port-of-internal-HTTP-server
            </param-value>
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>HTTPMonitorFilter</filter-name>
        <url-pattern>/*</url-pattern>
        <dispatcher>REQUEST</dispatcher>
        <dispatcher>FORWARD</dispatcher>
        <dispatcher>INCLUDE</dispatcher>
        <dispatcher>ERROR</dispatcher>
    </filter-mapping>
```

- Servlet 2.3 Filter Declaration:

```
<filter>
    <filter-name>HTTPMonitorFilter</filter-name>

    <filter-class>org.netbeans.modules.web.monitor.server.MonitorFilter</filter
    -class>
        <init-param>
            <param-name>netbeans.monitor.ide</param-name>
            <param-value>name-of-host-that-runs-IDE:port-of-internal-HTTP-server
            </param-value>
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>HTTPMonitorFilter</filter-name>
```

```
<url-pattern>/*</url-pattern>
</filter-mapping>
```

3. To monitor the data records from a web application using more than one IDE, add a second init-param entry as follows:

```
<init-param>
<param-name>netbeans.monitor.register</param-name>
<param-value>
other-IDE-host:other-HTTP-server-port,
another-IDE-host:another-HTTP-server-port</param-value>
</init-param>
```

**Tip:** When you deploy the web application to a production server, remember to remove the jar files from your WEB-INF/lib folder and remove the filter and filter mapping declarations from the web application's deployment descriptor.

### 23.8.2 How to Analyze Session Data

After you have setup the HTTP Server-Side Monitor, you can use it to debug your web application by observing data flow from your JSP page and servlet execution on the server. The HTTP Server-Side Monitor records data about each incoming request.

#### To analyze session data:

1. Run a web application.
2. Click the HTTP Server-Side Monitor.
3. Select any HTTP request in the tree view to view its information in the display panel.

The HTTP Server-Side Monitor consists of two panels. On the left is a tree view of HTTP request records. On the right is a display panel that presents the session data associated with the selected HTTP request records.

#### The tree view

In the tree view, the All Records category contains two subcategories: Current Records and Saved Records. Individual HTTP request records reside in either of these sub-categories. They are accompanied by the following icons:

- Get method
- Post method

Each method icon includes a badge based upon the value of the response status code:

Badge	Description
	1xx - Information
None	2xx or the value could not be determined. In the latter case, this normally means that the request has succeeded (200 - OK)
	3xx - Warning

Badge	Description
	400 - Error

The request's method type and status code are also displayed in the display panel's Request tab. For a full description of each of the status codes, see Section 10, "Status Code Definitions" of the Hypertext Transfer Protocol -- HTTP/1.1 available from the World Wide Web Consortium (W3C) at <http://www.w3.org/Protocols/rfc2616/rfc2616.html>.

Requests resulting in an internal dispatch cause nested nodes on servers that support this functionality. The forwarded or included requests are nested under the node corresponding to the main requests.

Entries in Current Records are available only during the current IDE session. Current HTTP request records persist across restarts of the server. They are cleared whenever you exit the IDE, or when you explicitly delete them. Entries in Saved Records persist until you delete them. Records in all categories can be sorted according to various criteria using the buttons above the tree view:

Icon	Description
	Reloads all the HTTP request records stored on the client.
	Sorts the HTTP request records by timestamp in descending order.
	Sorts the HTTP request records by timestamp in ascending order.
	Sorts the HTTP request records alphabetically.
	Shows or hides the time stamps in the list of HTTP request records.

**Tip:** If the HTTP Server-Side Monitor does not show the HTTP request records for a running web module, verify that the monitor is enabled for your web server by right-clicking your web server's node in the Services window's Servers node and choosing Properties from the contextual menu. The Enable HTTP Monitor checkbox should be selected.

### The display panel

When you select a HTTP request record in the tree view on the left, the session data corresponding to that HTTP request record appears in the display panel on the right.

### 23.8.3 How to Save Session Data

The HTTP request records that are in the tree view's Current Records node are available only during the current IDE session. Current HTTP request records persist

across restarts of the server. They are cleared whenever you exit the IDE, or when you explicitly delete them. Entries in the Saved Records node persist until you delete them.

You can save the HTTP request records that are in the Current Records node of the tree view. This action lets you view or replay the HTTP request in a later IDE session.

**To save session data:**

1. Right-click the HTTP request record to be saved. You can select multiple records at one time by using the Shift or Control keys.
2. Choose **Save** from the contextual menu.

The selected records are moved, not copied, to the Saved Records subcategory.

#### 23.8.4 How to Replay Session Data

You can replay HTTP requests that are associated with the current and saved HTTP request records. When you replay a request, the response appears in the web browser.

---

**Notes:** ■If the server or the web application is not running, an error message is displayed. If this happens, re-execute the application on a running target server.

- If the server or the web application is not running, an error message is displayed. If this happens, re-execute the application on a running target server.
  - If a servlet or JSP page reads data directly from the `ServletInputStream`, instead of using methods from the `getParameter` family, requests to that resource will not replay correctly.
- 

**To edit HTTP requests before replaying them:**

1. In the tree view on the left side of the HTTP Server-Side Monitor, right-click the HTTP request record that you want to edit and replay.
2. Choose **Edit and Replay** from the contextual menu. The Edit and Replay dialog box appears.

In the Edit and Replay dialog box, you can do the following:

**To edit requests:**

1. Click the Request tab.
2. Edit the request:
  - To edit the Request URI parameter or the Protocol parameter, click the ellipsis (...) button next to the parameter.
  - To change the request method, select the desired method from the combo box in the Method parameter. For instance, change GET to POST.
3. Click **Send HTTP Request**.

**To edit cookies:**

1. Click the Cookies tab.
2. Edit the cookie:

- To add a cookie, click **Add Cookie**. The Add Cookie dialog box appears. Type a name and value, and click **OK**.
- To edit a cookie, click the ellipsis (...) button next to the cookie.
- To delete a cookie, select the cookie and click **Delete Cookies**. Use the Shift or Control key to select multiple cookies for deletion.

3. Click **Send HTTP Request**.

---

**Note:** Not all servers support the editing of cookies. The server ignores the values that were saved when the record was recorded, and the server uses the cookie that the browser generates. If the record header does not include a session cookie, the browser does not send a cookie. Although you can edit session cookies, the changes do not take effect on most servers. The reason is because the session's cookie ID is read before the request enters the servlet container.

---

**To edit servers:**

1. Click the Server tab.
2. Edit the server. To change a parameter's value, such as the hostname of servlet engine or the port number of an HTTP service click the ellipsis (...) button next to the parameter.
3. Click **Send HTTP Request**.

**To edit headers:**

1. Click the Headers tab.
2. Edit the header.
  - To add a header, click **Add Header**. The Add Header dialog box appears. Type in a name and value and click **OK**.
  - To edit a header parameter such as Accept, Connection, Host, or User-Agent, click the ellipsis (...) button next to the parameter.
  - To delete a header, select it and click **Delete Headers**. Use the Shift or Control key to select multiple headers for deletion.
3. Click **Send HTTP Request**.

# 24

---

## Working and Connecting with Databases

This chapter describes how to create connections to Java DB databases and Oracle Database, and how to work with MySQL. It describes how to create and work with database objects including tables and views, and how to execute SQL statements.

This chapter contains the following sections:

- [About Working and Connecting with Databases](#)
- [Working with the Database Tools](#)
- [Setting up a Database Connection](#)

### 24.1 About Working and Connecting with Databases

The NetBeans IDE allows you to connect to databases and view their content, how to modify database objects, and how to view and change data.

It provides drivers for the Java DB, MySQL, Oracle, and PostgreSQL database servers so that you can connect to these databases very easily. You can also register any other JDBC driver with the IDE, so that you can explore any database that provides a JDBC driver.

### 24.2 Working with the Database Tools

When you open the Services window, you see the Databases node used to perform operations related to JDBC-compliant databases within the IDE.

Within the Databases node you can do the following:

- Connect to a database.
- View current database connections.
- Select or add a driver for your database.
- Enter SQL statements and see the results immediately.
- Run SQL scripts on a connected database.
- Migrate table schemas across databases from different vendors.
- Create, browse and edit database structures by running SQL statements or using a graphic view.

### 24.2.1 How to Browse Database Structures

Databases consist of tables, views and stored procedures, each of which represents a list of columns.

When a connection is established, you can expand the database connection node (  ) to see the subnodes representing tables, views, and procedures.

#### To browse a table:

1. In the Services window, expand the node for a connected database.
2. Right-click the Tables node to reveal the individual tables.
3. Expand the Tables node to reveal the individual tables.
4. Expand a table node (  ) to see the table's columns, an Indexes node and Foreign keys node.

#### Columns

This table lists Services window column icons and their descriptions.

Icon	Description
	Table
	Column
	Column (Primary Key)
	Column (Unique key)
	Index
	Foreign key
	Column in foreign key
	View

#### Indexes

Click on an index node to see the list of the columns contained by that index.

#### Foreign Keys

Click on a foreign key node to see the list of the columns contained by that foreign key. A foreign key contains columns that reference another table's columns (usually primary keys).

#### Views

Click the Views node to see the views on a given database. View columns are dimmed to indicate that you cannot change the physical characteristics of the columns in a view. However, you can change a view by dropping the view and subsequently recreating it.

### Procedures

Select a Procedures node to view stored procedure names and their in and out parameters.

## 24.2.2 How to Create Database Tables

When connected to a database, you can use the IDE to create tables in a database in the following ways:

- Use the Create Tables dialog by right-clicking the Tables node in the Services window and choosing Create Table. You can create a table in a graphic form.
- Use a table definition to re-create a table.
- Run an SQL script in the SQL Editor.

#### To create a new table using the Create Tables dialog:

1. In the Services window, expand the node for a connected database.
2. Right-click the Tables node and choose **Create Table**.
3. Enter the table name.

This name must follow conventions for quoted table names in your database. Case is usually significant and you can use spaces. However, it is recommended that table names be entered in upper case and without spaces.

4. Modify the table with **Add Column** and **Remove**.

Specify the column type (required) and column size. Column size is optional for some data types.

5. Specify additional properties for the column (Optional).
6. Click **OK** to create the table.

#### To grab a table definition:

1. In the Services window, expand the node for a connected database.
2. Expand the Tables node, right-click the table you want to re-create and choose **Grab Structure**.
3. In the Grab Table dialog, type a name and navigate to where you want to save the file and click **Save**.

When you click Save, a snapshot of the definition of the selected table is saved in the selected location. You can use the snapshot to re-create the table in the current database or in a different database.

#### To re-create a table from a table definition:

1. In the Services window, expand the node for a connected database.
2. Right-click the Tables node (or the node of any table under the Tables node) and choose **Recreate Table**.
3. Locate and select the table definition file you want to use to re-create a table.
4. Specify the table name in the dialog and click **OK** to re-create the table. If you want to modify the table definition command, click the Edit table script button and edit the table definition command.

**To recreate a table in a different database:**

- You can save the structure of a table from one database and recreate the table in a different database, even if the two databases have a different SQL syntax. You can thus create database-independent schemas. For example, you can develop a complete application with a database from one vendor and then for deployment, you can transfer the table structures to a different database by a different vendor. You do not need to edit the SQL.

### 24.2.3 How to Create Database Views

The Database Explorer enables you to create SQL views that are part of the database itself and are available to application programs and other clients of the database.

**To create views:**

1. In the Services window, expand the node for a connected database.
2. Right-click Views and choose **Create View**.
3. In the Create View dialog, type the name of the view in **View Name**.
4. In the text area, enter the SQL (SELECT) statement that will define the new view and click **OK**. The IDE adds a node for the new view (  ) under the Views node for the database.

Once the view is saved, you can see the data it contains by right-clicking the view in the Services window and choosing **View Data**. Choose **Execute Command** to run an SQL command on this view.

### 24.2.4 How to Add Columns to a Database

When connected to a database, the database tables can be modified directly through the IDE.

**To add a column:**

1. In the Services window, expand the node for a connected database (  ).
2. Expand the Tables node to display the list of tables.
3. Right-click the table you want to modify and choose **Add Column**.
4. Enter the values for the column in the dialog.
  - Enter the name of the column.
  - Provide the Type of the column by selecting a type from the Type drop-down menu.
  - Provide the size of the column in **Size**.
  - Provide a scale value for numeric or decimal data type.
  - Provide a default value for the column in **Default**.
  - Select constraints you want for column values (**Primary key**, and **Unique** or **Null**).
  - If you select **Primary key**, the column is included in the primary key and in most cases the database server will generate an index for it.
  - If you want to add the new column to an existing index, select **Index** and select an index from the drop-down menu.

- (Optional) Select **Check** to enter constraints to be checked while inserting or updating data in that column.
- 5. Click **OK**.

---

**Note:** The ability to add or copy columns depends on the database you use. Your database system might not allow you to alter tables containing data.

---

When you add a column to a table, the new column appears under the table node (  ) in the Services window. After creating a column, you may need to recapture the database schema to use the new column in your project.

#### 24.2.5 How to Delete Database Objects

When connected to a database, you can delete data objects from the database schema and remove the corresponding node from the folder.

**To remove a data object:**

1. In the Services window, expand the node for a connected database.
2. Locate and right-click on the object you wish to delete.
3. Choose **Delete** from the pop-up menu or press the Delete key on the keyboard to delete an object.

#### 24.2.6 How to View and Modify Data in a Database

When a database is connected, you can view and modify the data contained in a table in the built-in graphical representation of the SQL Editor.

**To view data in a table:**

1. In the Services window, expand the node for a connected database (  ).
2. Expand the Tables or Views node.
3. Right-click the node of the table or view and choose **View Data**.

When you choose View Data, an SQL query to select all the data from the table is automatically generated in the upper pane of the SQL Editor. A visual representation of the table and the data it contains is displayed in the lower pane of the SQL Editor.

You can use this GUI representation of the table to add, modify, and delete table data, for example:

- **Insert new records.** To add new records, click the Insert Records button (  ). This opens the Insert Records dialog window, in which you enter new records. You can click the Show SQL button in the Insert Records dialog window to view the SQL statement(s) that will be applied upon initiating the action. Click OK to add the new records to the database table.
- **Modify records.** To edit existing records, double-click a record in a table cell and modify its content. Modified entries are shown in green. Click the Commit Record button (  ) to commit changes to the actual database. Similarly, click the Cancel Edits button (  ) to cancel any edits you have made.
- **Delete selected records.** Select a row in the table and choose the Delete Selected Record button (  ). You choose multiple rows simultaneously by holding Ctrl.

- **Delete all records.** To delete all records in a table, click the Truncate Table button ().

If the displayed data needs to be resynchronized with the actual database, you can click the Refresh Records ( ) button.

For more on using the SQL Editor, see [Section 24.2.7, "How to Execute SQL Statements."](#)

**Tip:** Each column node has the same View Data item in its pop-up menu. This feature is similar to the feature for tables; however, when you view tables, the SQL statement for viewing data fetches only the one column rather than the entire set of columns that the IDE displays when you choose **View Data**.

## 24.2.7 How to Execute SQL Statements

You can use the SQL Editor to write and execute SQL statements and SQL scripts within the IDE. The SQL statement or script is executed on the database that is selected in the Connection drop-down list in the toolbar. If the database connection is closed, the IDE opens the connection to the database. In the Connection drop-down list, you can change the database on which you want to run the SQL statement.

You can use the SQL Editor to create, edit and execute any SQL files in your project. To open an SQL file, right-click the SQL file ( ) in the Projects or Files window and choose **Open**. You can also double-click the icon to open the file in the SQL Editor.

You can run SQL statements directly from an SQL file, without opening it.

### To run an SQL file:

1. Confirm that the target database server is running.
2. Open the Favorites window by choosing **Window > Favorites**.
3. Right-click inside the Favorites window and choose **Add to Favorites**. Add the directory that contains the SQL file you want to run.
4. Right-click the SQL file and choose **Run file**.
5. Select a database from the list of registered database connections. Click **OK**. The SQL script will be executed on the selected database connection.

### To create and execute a SQL statement or script:

1. Expand the Databases node in the Services window.
2. Right-click the node for the connected database and choose **Execute Command** from the pop-up menu to open the SQL Editor.
3. Make sure your current database connection is selected in the **Connection** list. To see the status of the listed connections, click the **Select Connection** in Explorer icon ( ) in the toolbar. When you click **Select Connection** in Explorer, the selected connection is highlighted in the Services window.
4. Enter a DDL or DML statement in the SQL Editor window.
5. Click **Run SQL** ( ) in the toolbar or right-click in the SQL Editor and choose **Run Statement** from the pop-up menu.

---

**Note:** When you execute SQL in the SQL Editor, normally the results tabs from the previous execution are closed. You can change this behavior so that the previous tab remains open. To do this, click Keep Prior Results ( ) in the tool bar. This setting is saved for all subsequent SQL executions, or until you toggle it off again.

---

When you execute an SQL statement or script, you see the following:

- The status of the SQL execution is displayed in the Output window. Any errors when executing the SQL statement or script are displayed in the Output window.
- The output of the statement appears in the results pane in the lower half of the SQL Editor. If the statement has no output, the number of affected rows is displayed in the Output window of the IDE.

---

**Notes:** ■ You can run selected text in the SQL Editor by right-clicking the selected text and choosing **Run Selection** from the pop-up menu.

- You can access the Execute Command pop-up menu item from any subnode of a connected database ( ) in the Services window.
  - To view the data in a table, right-click a table in the Services window and choose **View Data** from the pop-up menu.
  - Click SQL History ( ) in the toolbar to show the list of all SQL commands that you executed.
- 

#### 24.2.8 How to Obtain a Database Schema

The IDE allows you to capture and create a persistent database schema which can then be used in other projects.

To connect to a database to retrieve a schema, the database driver first needs to be added to the IDE. Database drivers can be added in the Services window. For more information, see [Section 24.3.1, "How to Add a JDBC Driver."](#)

**To capture a database schema:**

1. Make sure the database server is running.
2. Choose **File > New File** to open the New File wizard.
3. Select the Project where the schema should be saved.
4. Select **Persistence** in the Categories pane.
5. Select **Database Schema** in the File Types pane and click **Next**.
6. For **File Name**, provide a name for the schema.
7. Click **Browse** to specify the folder where to save the schema. The default location is **src** within the specified project. You can choose any other location for the schema file.
8. Click **Next**.
9. Select an existing database connection from the list or select **New Database Connection** to create a new connection. For more information, see [Section 24.3](#),

"Setting up a Database Connection."

**10. Click Next.**

You may be prompted by a Connect dialog to supply a username and password to connect to the database, if required. Enter the user name and password in the Basic setting tab and click **OK**.

When connecting to the database, the schema with the same name as the user name is set as default, but if a schema with that name does not exist, you are automatically switched to the Advanced Tab to select a schema. Choose the correct schema and click **OK**.

**11. Select the tables and views you want to capture in the Available Tables and Views list on the left, then click **Add** to add them to your list of Selected Tables and Views on the right. Alternatively, you can click **Add All** to specify all tables and views.**

If you specify a table that references a table that you have not chosen to capture, both tables are captured.

**12. Click **Finish** to complete capturing the schema.**

#### 24.2.9 How to Recapture a Schema from a Database

When you modify your database, such as by adding or deleting fields or changing relationships, you may need to recapture the database schema so that the changes are also available to your project.

**To recapture the database schema from the database:**

1. Make sure the database server is running.
2. Locate the database schema node (  ). The database schema can be found in the following locations:
  - in the Source Packages node in the Projects window
  - in the specified location within the project structure in the Files window
3. Right-click the database schema node and choose **Recapture Schema** from Database.
4. Enter the username and password, if necessary.
5. Click **OK**.

The Recapturing Database Schema progress box opens and displays the recapturing progress, including the table and view information it is capturing. After recapturing the database schema, you can:

- Explore the structure of the new database schema by expanding the schema in the Projects or Files windows. For more information, see [Section 24.2.1, "How to Browse Database Structures."](#)
- Explore the recaptured schema and view the data in the Services window. For more information, see [Section 24.2.6, "How to View and Modify Data in a Database."](#)

### 24.3 Setting up a Database Connection

A database connection is represented by a database connection node (  ) under the Databases node in the Services window. To connect to a database, you need to provide a valid JDBC driver for your database in the NetBeans IDE.

When you create a database connection, you supply the details needed to connect to a specific database. These details include the location of the database, which driver to use, and the username and password information.

**To create a new connection:**

1. Ensure that your database is running.
2. Right-click the Databases node and choose **New Connection**. Alternatively, you can initiate a connection directly from the list of registered drivers. If the database driver is already listed under the Drivers node, right-click this driver's instance node and choose **Connect Using**.
3. In the first step of the New Connection wizard, choose the driver for the database you are connecting to from the **Driver** drop-down list and confirm that the correct driver's file is provided in **Driver File**. Click **Next**.

You are passing the same steps when registering a new driver in the IDE. For more information, see [Section 24.3.1, "How to Add a JDBC Driver."](#)

4. In the Customize Connection dialog of the New Connection wizard, confirm that the correct driver is selected for your database.
5. Depending on what database you are connecting to, provide the connection details for this database, such as database location (hostname or IP address), port, username and password, database name, service name, etc. For more information, see:
  - [Section 24.3.2, "How to Connect to the Java DB Database"](#)
  - [Section 24.3.3, "How to Connect to Oracle Database"](#)
  - [Section 24.3.4, "How to Use MySQL Database with the IDE"](#)

To help you get started, the IDE fills the default details for some databases for you. If in doubt, refer to the documentation for the driver that you are using.

6. (Optional) Click **Remember Password** to have the IDE remember your password. If selected, you will not be prompted for the user name and password on subsequent connections to the database.

To set the Remember Password property to `False`, right-click the database connection node in the Services window and choose **Properties**. Modify the Remember Password property. Note that you can do this only when the connection you are modifying is in the disconnected state. When the connection is enabled, you cannot change this property.

7. Click **Test Connection** to test the connection to the database.

### 24.3.1 How to Add a JDBC Driver

To connect to a database from within the IDE, you must have an appropriate JDBC driver for this database and register this driver with the IDE. Registered drivers () are listed under Drivers under the Databases node in the Services window.

The drivers are of the following kind:

- JDBC drivers that are registered and bundled with the NetBeans IDE. These are drivers for MySQL and PostgreSQL. You can establish a connection to these databases right away, without downloading them separately.
- JDBC drivers that are registered with the IDE by default, but not bundled with the IDE. These are drivers for Oracle Database and Java DB. For example, the Java DB

drivers are located in the Java DB installation directory and NetBeans recognizes them automatically at their default location. As for the drivers for Oracle Database, you need to download them manually from the Oracle website.

If you need another JDBC driver that is different from those mentioned above, you need to manually add it to the IDE.

**Tips:** ■ Adding a new driver just creates a template for new connections. The IDE does not actually try to use that driver or check it to ensure it is correct until a connection is made using the driver.

- You can add the driver directly at the first step when setting up a connection. If you want just to add a driver, without connecting to the database, right-click the Drivers node and choose New Driver. For more information, see [Section 24.3.3, "How to Connect to Oracle Database"](#)

An uninstalled driver node ( ) is displayed in the Services window if a driver is registered with the IDE but the archive JAR file for the driver is not currently on the IDE's class path. In this case, you should remove the driver or modify the driver's properties to specify the correct location of the driver JAR file.

**To remove a driver:**

- Right-click the driver node in the Services window and choose **Delete**.

**To modify driver's properties:**

- Right-click the driver's node and choose **Customize**.

### 24.3.2 How to Connect to the Java DB Database

The Java DB database is a distribution of the open source Apache Derby database. Java DB is a fully transactional, secure, standards-based database server, written fully in the Java programming language, and fully supports SQL, JDBC API, and Java EE technology. The Java DB database server is bundled with the GlassFish application server.

If you install the GlassFish application server during the IDE installation, the Java DB database server is automatically registered in the IDE.

**To enable the Java DB database server in the IDE, do one of the following:**

- Register an instance of the GlassFish application server. This registers the installation of the Java DB database server that is packaged with the application server. The connection to the sample Java DB database is automatically displayed in the Services window. The default user name and password for the sample database is app. For more information, see [Section 23.3.1, "How to Register a Server Instance."](#)
- Register an existing Java DB or Apache Derby installation. If you already have one of these database servers installed on your computer, you can register the database with the IDE by doing the following:
  1. In the Services window, right-click the Java DB node and choose **Properties**.
  2. In the dialog, set the location of the Java DB installation directory.
  3. Set the folder where your Java DB databases are stored in the Database Location property and click **OK**.

**4. Start the Java DB server.**

Depending on your installation, you can start the server in the Services window by right-clicking the Java DB node and choosing **Start Server**. In some cases you might need to start the server from the command line prompt.

Once your Java DB database is registered with the IDE, the Java DB Database menu item appears under the Databases Node in the Services window. This menu item enables you to easily start and stop the Java DB database server and to create a new database.

**To create a new Java DB database:**

- 1.** In the Services window, right-click the Java DB node and choose **Create Database**.
- 2.** In the Create Java DB Database dialog, enter a name for the database, a user name, and a password.

By default, the IDE creates the database in the `.netbeans-derby` folder of your home directory. To change the default location, click **Properties** in the Create Java DB Database dialog, or in the Services window, right-click the Java DB node and choose **Properties**. Type the new database location in the appropriate field.

**To connect to an existing Java DB database:**

- 1.** In the Services window, right-click the Databases node and choose **New Connection**.
- 2.** In the Locate Driver step of the New Connection wizard, choose one of the following Java DB drivers from the drop-down menu:
  - **Java DB (Embedded).** Use this driver when you are sure your application will access the database from a single JVM. A typical example of this scenario is a single-user Swing JDBC application using a Java DB database as a data store.
  - **Java DB (Network).** Use this driver when you need to connect to the database from multiple JVMs. An example of this scenario is when you need to connect from the IDE and from a Java EE application on the application server. Using the network driver, you can also access the database from remote computers.
- 3.** In the Customize Connection dialog of the New Connection wizard, enter the database name to which you want to connect.
- 4.** Provide the database user name and password.
- 5.** (Optional) Click **Test Connection** to see if the connection can be established.
- 6.** Click **OK**. The IDE displays the connection to the database under the Databases node in the Services window.

---

**Notes:** ■Selecting **Remember Password** will store the password for the current IDE session. If left unselected, a dialog appears prompting you for the password every time you connect to the database.

- The URL for the sample database is `jdbc:derby://localhost:1527/sample`.
  - The default location for the sample database is in the `.netbeans-derby` directory in your home directory.
  - To change the default location for the Java DB database directory, in the Services window, right-click the Java DB node in the Services window and choose **Properties**. Enter the new location.
- 

### 24.3.3 How to Connect to Oracle Database

You can establish a connection to Oracle Database from the NetBeans IDE. For example, you can use Oracle Database 11g Express Edition (Oracle Database XE), a lightweight database that is free to develop, deploy, and distribute.

Before connecting to Oracle Database, ensure that you have downloaded the JDBC driver for the database. The recommended driver is `ojdbc6.jar`. For more information about Oracle JDBC drivers, see the JDBC page at the Oracle Technology Network (<http://www.oracle.com/technetwork/database/features/jdbc>).

**To connect to the Oracle Database XE from the IDE:**

1. In the Services window, right-click the Databases node and choose **New Connection**.
2. In the Locate Driver step of the New Connection wizard, choose one of the following drivers for Oracle Database:
  - **Oracle Thin.** Oracle's JDBC Thin driver uses Java sockets to connect directly to Oracle Database. Because it is written in Java, this driver is platform independent and can also run from a Web Browser (applets).
  - **Oracle OCI.** Oracle's JDBC OCI drivers use Oracle Call Interface (OCI) and native libraries to interact with an Oracle database. The required libraries are part of the Oracle Database Instant Client, which needs to be installed and configured on your system.
3. Click **Add** to specify the location of the `ojdbc6.jar` file. This file contains packages for both types of Oracle DB drivers. However, if you want to use the OCI driver, you also need to have Oracle Instant Client installed on your machine. Click **Next**.
4. In the Customize Connection dialog of the New Connection wizard, choose the driver name and how you want to specify the database: by a SID name (Oracle System ID), Service Name, or a TNS name.
5. Then, enter the hostname or IP address to which you want to connect. If you are connecting to a locally installed database instance, enter `localhost` or `127.0.0.1`.
6. Provide the port number. The default port is `1521`.
7. Depending on your choice of the driver, enter either the SID name, Service, or TNS name. For example, the default SID for Oracle Database XE is `XE`.
8. Provide the database user name and password.
9. (Optional) Click **Test Connection** to see if the connection can be established.

10. Click **Next** and on the Customize Connection page choose the schema you want to use.
11. (Optional) Click **Next** and change the connection name. The default name is similar to `jdbc:oracle:thin:@host:port:SID [connection on schema]`
12. Click **Finish**. The IDE displays the connection to the database under the Databases node in the Services window.

For more information, see Connecting to Oracle Database from NetBeans IDE available on the NetBeans site at  
<https://netbeans.org/kb/docs/ide/oracle-db.html>.

---

**Notes:** ■ SID, or System ID, is a unique name of an Oracle database instance. For example, the default SID for Oracle Database XE is XE.

You can find the SID in the `tnsnames.ora` file, which is a configuration file that specifies various database properties used to connect to the database. The default location of `tnsnames.ora` is `%ORACLE_HOME%\NETWORK\ADMIN` directory.

- You can also find the SID and Service Name values through the web-based database management interface. For example, go to Database Home Page for Oracle Database XE, choose **Administration > About Database > Parameters**. Find the values of the following properties: `db_name`, which is the database SID; `service_names` is the database Service Name. You must have database administrator rights to access this information.
  - The sample URL to the Oracle Database XE is  
`jdbc:oracle:thin:@//localhost:1521/XE`
- 

#### 24.3.4 How to Use MySQL Database with the IDE

The IDE comes with MySQL database support. In the Services window, a MySQL node appears under the Databases node when the IDE detects a running MySQL server on the local machine at the default port. The node is also registered automatically under the following conditions:

- If the IDE detects a recognized installation of MySQL. This is currently MAMP on Macintosh and XAMPP on Windows.
- If a MySQL connection is registered, the IDE auto-registers the MySQL node using the properties defined on the connection.
- If the user creates a new connection to MySQL, the IDE auto-registers the MySQL node at that time.

When you right-click the MySQL node, you see the following options:

- **Create Database** Brings up a dialog for creating a database, including sample databases.

---

**Note:** When using MySQL 5.0 with Windows, some foreign characters may not be used when creating a database.

---

- **Start** Starts the database server. If the user chooses this action and no start or stop command is set, then they are prompted to set the command.

- **Stop** Stops the database server. If the user chooses this action and no start or stop command is set, then they are prompted to set the command.
- **Connect** Connects the IDE to the MySQL database server.
- **Disconnect** Disconnects the IDE from the MySQL database server.
- **Delete** Removes the MySQL node. It also marks the node as removed, so that next time the user starts up the IDE will not auto-register the node again. The user will need to manually register the node using **Databases > Register MySQL Server**.
- **Refresh** Refreshes the list of databases based on what's currently on the server.
- **Run Administration Tool** Brings up the administration tool that is registered with the runtime. The provider either invokes the admin tool as a separate system process, or if it is a web-based tool, brings up the provided URL in the default browser. If the administer command has not been set, then when the user picks the action, they are prompted to set the path to the command.
- **Properties** Brings up a dialog that allows the user to configure important settings for the server. There are two tabs, one for basic settings and one for setting command paths to administrative commands (start, stop, administer)

When you expand the MySQL node, MySQL databases detected by the IDE appear under the node. By right-clicking a database, you can either connect to or delete the database.

If the IDE does not auto-detect and register the MySQL node, you can manually register it. In this case, the Databases node has a new subnode designed to register the server. When you right-click the Databases node, you see the following options:

- **New Connection** When you choose New Connection, the New Database Connection dialog appears.
- **Register MySQL Database** If the IDE does not auto-detect and register the MySQL node, the user can manually register it. There will be a new action under Databases to register the server
- **Enable Debug** Once debug mode is enabled, the IDE sends SQL statements to the Output window, where you can view the log of statements. Debug mode does not list all SQL statements, only those which you cannot see. For example, SQL queries for viewing data are not logged.
- **Disable Debug** This terminates the debugging function.

### 24.3.5 How to Enable Debug Mode

Enable debug mode to see SQL statements that are being sent to the database. By default, debug mode is disabled.

#### To enable debug mode:

1. Choose **Window > Services** from the main menu to open the Services window.
2. In the Services window, right-click the Databases node and choose **Enable Debug**.

Once debug mode is enabled, the IDE sends SQL statements to the Output window, where you can view the log of statements. Debug mode does not list all SQL statements, only those which you cannot see. For example, SQL queries for viewing data are not logged.