IBM VisualAge® for Java™, Version 3.5

Integrated Development Environment

Note!
Before using this information and the product it supports, be sure to read the general information under **Notices**.

Edition notice

This edition applies to Version 3.5 of IBM VisualAge for Java and to all subsequent releases and modifications until otherwise indicated in new editions.

© Copyright International Business Machines Corporation 1997, 2000. All rights reserved.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

© Copyright IBM Corp. 1997, 2000

Chapter 1. The VisualAge for Java IDE

What's new in the VisualAge for Java IDE

Several usability and customization features have been added to the IDE for this release.

Solutions
A solution is a container that holds a group of related projects. Grouping projects into solutions allows you to import or export the projects in a single operation. Solutions are created in the new Solutions browser in the Repository Explorer.

Inner classes
Inner classes and their methods now appear in the general hierarchy views in the workbench browsers. Inner classes are indented from their owning class. You can edit and save inner classes and their methods from the browsers.

Inner Classes and their methods are still part of the Class definition. As such, they do not have an edition history.

Additional formatting options
Several new options have been added to the formatting page of the Options notebook. These include:

- Insert new line before opening brace
- Insert new line in control statement
- Keep else if on the same line
- Maximum line length
- Compact assignment
- Clear all blank lines (clear all/preserve one)
- Indentation is represented by tab/space
- Amount of spaces representing a tabulation

Class Source View
Source view is the ability for the user to view a complete class, including all of its methods. In this view, the user will be able to see and edit the class definition and all of the methods of a class at a single time. If this file was imported from a file, this view will retain the sorting that was present in the file.

Interface hierarchy view
You can see a hierarchy of interfaces from the Interface browser, similar to the hierarchy view that is available for classes

Project Resource Management
Project resources are any files other than .class files that belong to a project, for example, HTML, XML, or gifs. In past releases, project resource files were not associated with a specific version of a project. In VisualAge for Java, version 3.5, version control has been implemented for project resources. The resource files are stored in the ide/project_resources/ProjectA directory until the project is versioned. When the project is versioned, the project resources in the ProjectA directory are moved to the ide/repository/repositoryname.pr/ProjectA/timestamp directory. When you open a new edition of a project, the resources will move into the working directory.

© Copyright IBM Corp. 1997, 2000

Project resources can be managed from the new Resources page in the Workbench browser and from the Projects page in the Repository Explorer.

F1 Help improvements
To open the help browser for context sensitive help on a type or keyword, simply highlight the text and press F1.

Improved searching and filtering

- You can now search the workbench and the results of a comparison using a working set. A working set is a group of program elements. Working sets can be defined from the search dialog boxes.
- In the class browser you can choose whether or not inner classes and classes based on modifiers are displayed.
- You can select a working set of program elements in the All Problems page.
- You can choose to remove the warnings and list only problems in the All Problems page.

Sort lines alphabetically
By selecting lines and pushing Alt + F8, you can put the lines in alphabetical order.

Automatic generation of Serial UID
There is a new menu item that will generate the Serial UID for serializable objects.

Fix or Migrate
The Fix/Migrate tool repairs broken references to program elements, including projects, packages, classes, and interfaces. Select a program element then **Selected > Reorganize > Fix/Migrate** to launch the tool.

Extensions Support
An extension is a group of packages that implement an API that extends the Java platform. Extension classes, like classes in the platform's core API, can be found and loaded by the Java virtual machine even though they are not in the class path. You can specify that a project has an extension in the project's properties dialog.

Java2 Samples
All of the IDE samples provided have been migrated to work on the Java 2 platform.

Overview of the VisualAge for Java IDE

VisualAge for Java is an integrated, visual environment that supports the complete cycle of Java program development. You can create Java applets, which run in web browsers, and standalone Java applications.

With VisualAge for Java, you can do the following tasks:

- build Java programs interactively
- run Java programs
- run fragments of Java code before you include them in classes
- debug Java programs, changing them as you run the code
- manage multiple editions of Java code
- import Java source and binary code from the file system
- export Java source and binary code to the file system

- build, modify, and use beans

RELATED CONCEPTS
What's new in the VisualAge for Java IDE
Incremental compilation
Unresolved problems
Development without files
Projects and other program elements
Workspace

Development without files

As a programmer, you are probably familiar with developing programs in a file-based environment. You write code in source files and compile and link them into executable binary files. As you build the executable code, you must manage files and file dependencies. This is an error-prone process that takes time away from your primary task of programming.

By providing a development environment without files, VisualAge for Java performs the tasks of managing and compiling code. Browsers let you view and edit classes and methods individually. VisualAge for Java compiles Java source code for you when you save it.

From within the IDE, you can import code from the file system into VisualAge for Java. You can also export code to the file system if you want to work outside VisualAge for Java.

Editing is integrated
The Workbench and browsers reflect an object model rather than a file-based model. The object hierarchy of program elements (projects, packages, classes, interfaces, and methods) provides a structure for the code. The Workbench and browsers have a source pane with full editing capabilities that let you modify and save source code.

Compilation occurs when you save source code
After you modify the source for a class or method, you save it, and VisualAge for Java compiles the new code for you. VisualAge for Java keeps track of both the source and its corresponding bytecodes. However, you never see .java or .class files in the IDE.

VisualAge for Java compiles source code incrementally. That is, it compiles only those parts of the source code that you change (and other code that is directly dependent on it) and therefore significantly reduces the overall compilation time.

RELATED CONCEPTS
Unresolved problems
Projects and other program elements
Incremental compilation
Workspace
Repository
Editions and versioning

RELATED TASKS
Importing files from the file system
Compiling code

Saving changes to code
Experimenting with code fragments
Exporting and publishing code

Incremental compilation

The VisualAge for Java IDE automatically compiles Java source code into Java bytecode. When source code is imported into the workspace (from .java files) or added from the repository, it is compiled and analyzed with respect to the existing contents of the workspace. Any errors are flagged and listed in the Problems page of program element browsers and the Workbench.

When you import Java bytecode classes (.class files), or add them from the repository, they are similarly analyzed with respect to the existing contents of the workspace, and errors are similarly flagged.

Other changes, such as deleting, moving, copying, or renaming program elements, also initiate a compilation of affected program elements, to flag any new problems.

When you make a change to the source code for a method, field, or class, the change and all affected code is compiled when you save the changes. If you introduce an error, the IDE will warn you and give you the option of fixing the problem immediately, or of adding the problem to the Problems page and fixing later. If you choose to fix it immediately, the IDE's code assist tool will suggest possible solutions, if it can determine them.

Compiled code is stored in the workspace, but not in the repository (except for those classes that were imported from bytecode files rather than source code files). If you delete a class from the workspace, it deletes the bytecode, while the source code is still stored in the repository. If you add it back to the workspace, it will be recompiled before you can work with it again.

RELATED CONCEPTS
Unresolved problems
Projects and other program elements
Workspace

RELATED TASKS
Importing files from the file system
Creating a class
Creating an interface
Adding classes and methods from the repository to the workspace
Copying or moving a program element
Deleting program elements from the workspace
Renaming a program element
Saving changes to code
Finding and fixing problems

Workspace

All activity in VisualAge for Java is organized around a workspace, which contains the Java programs that you are developing. The workspace also contains all the packages, classes, and interfaces that are found in the standard Java class libraries, and other libraries that your classes may need.

[ENTERPRISE] In the team development environment, each VisualAge for Java client has its own workspace.

The workspace differs from the repository in the following ways:

- Program elements must be added to the workspace before they can be modified. Program elements that are in the repository can only be browsed.
- The workspace contains bytecode. The repository contains source code and Visual Composition Editor information.
- You can have only one edition of any program element in the workspace at any time. For performance reasons, your workspace should only contain the program elements that you are currently working on. By contrast, the repository contains every edition of every program element that you have ever developed, unless you have compacted the repository.
- You use the Workbench window to view, manipulate, create, modify, and manage program elements that are in the workspace. You use the Repository Explorer window to view program elements that are in the repository, add them to the workspace, and purge them from the repository.
- Changes to the workspace are not saved until you select **Save Workspace** from the **File** pull-down menu, or until you exit the IDE. Changes to the source repository are saved immediately, every time you save changes to a method, class, or interface.

When you start the IDE, the workspace is connected to the repository. The first time that you connect, VisualAge for Java builds pointers to the source code in the repository for every program element that exists in the workspace.

[ENTERPRISE] When you change repositories, this set of pointers is recached. You can not browse program elements that do not reside in the repository to which you are currently connected. You can always run code that is in the workspace, but if the connection to the repository is broken - for example by a server failure - then you can not browse source code or save changes.

You can add program elements from the repository to the workspace, replace the edition that is in the workspace with a different edition from the repository, or delete program elements from the workspace. You can maintain different versions of the workspace, customized for different projects or releases. See the list of topics below, for links to related information.

RELATED CONCEPTS

Repository
Overview of the VisualAge for Java IDE

RELATED TASKS

Browsing the workspace
Adding classes and methods from the repository to the workspace
Adding projects and packages from the repository to the workspace
Deleting program elements from the workspace
Replacing editions in the workspace (reloading)
Saving the workspace
Importing files from the file system
Recovering the workspace
Reinstalling the workspace

RELATED REFERENCES
Important files to back up

Repository

In the VisualAge for Java IDE, the repository is a source control mechanism that allows you to track changes made to program elements. When you start the IDE, it connects to a repository. As you create and modify program elements in the workspace, your changes are automatically stored in the repository. You can undo changes by retrieving previous editions from the repository.

Unlike the workspace, the repository contains all editions of all program elements. When you remove program elements from the workspace, they remain in the repository. Over time, the repository will grow. You should periodically purge program elements that are no longer required, and then compact the repository to reduce its size.

[ENTERPRISE] In the team environment of VisualAge for Java, Enterprise Edition, all team members' editions are stored in a shared repository on a server.

The Repository Explorer is the visual interface to the repository. Here are some examples of tasks that you can perform from the Repository Explorer window:

- Browse editions of projects, packages, classes, interfaces, and methods
- Create solutions
- Compare different editions of program elements
- Add program elements to the workspace
- Manage your resource files
- Change to another repository

Resource files, such as images and HTML files, are stored in the file system, not in the repository. You can, however, open and manage your resource files in the Repository Explorer. For more information, see the list of related topics at the end of this file.

RELATED CONCEPTS
Workspace
Editions and versioning

RELATED TASKS
Creating a solution
Adding projects and packages from the repository to the workspace
Searching for a program element in the repository
Comparing editions of a program element
Purging program elements from the repository
Backing up the repository
Compacting a repository
Exporting to another repository
Importing from another repository
Including resource files in a project

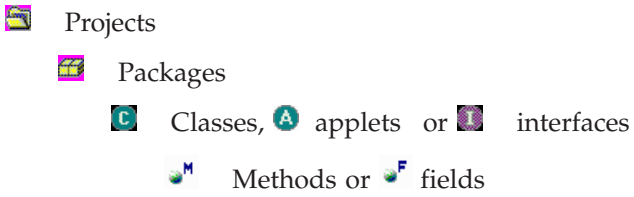
RELATED REFERENCES
Repository files

Projects and other program elements

The starting point for development work in the VisualAge for Java IDE is a *project*. Projects are units of organization used to group packages. They can be used, for example, to group packages from a certain provider, to group packages related to one application or customer, or to group frequently used classes that provide interrelated function. You can use them as best suits your development situation.

Packages and classes have the same meaning as in other implementations of the Java language. They are Java constructs. Projects contain packages, packages contain classes and interfaces, and classes contain methods. We refer to these constructs collectively as “program elements”.

In the IDE, the following symbols are used to represent the different program elements:



The Workbench organizes all the program elements that are in the workspace. From the Workbench, you can view, create, modify, and manage program elements. You can also open browsers and other windows that help you perform specialized tasks on program elements.

The Repository Explorer organizes all the program elements that are in the repository. With the Repository Explorer, you can view all editions of all projects, packages, classes, interfaces, methods, and resource files that are in the repository.

In the Repository Explorer you can organize your projects into solutions. A solution is a container that holds a group of projects. You can create solutions that contain related projects that you want to import or export as a group, for example a set of projects that you want to send to a particular customer.

The standard projects
The following projects are loaded into the workspace by default when you first install the IDE:

- IBM Java Implementation
- Java class libraries
- Sun class libraries

Other projects that are shipped with VisualAge for Java are stored in the repository, but not initially loaded into the workspace. You can add them as you need them.

[ENTERPRISE] Every project, package, class, or interface has an owner who is responsible for the quality of that program element and is authorized to release it. Each edition of a class or interface also has a developer, who is the only person who can version that program element.

RELATED CONCEPTS
Workspace
Repository
Editions and versioning

RELATED TASKS
Creating a solution
Creating a project
Creating a package
Creating a class
Creating an interface
Deleting program elements from the workspace
Printing program elements
Searching for a program element in the workspace
Searching for a program element in the repository
Versioning a program element
Adding projects and packages from the repository to the workspace

RELATED REFERENCES
VisualAge for Java IDE symbols

Resource files and directories

When you develop an application, you may use *resource files* that are not Java source files or bytecode. For example, you might create HTML files, image files, audio clips, or SQLJ source files.

You create resource files outside the development environment and store them in the project resources directory. You also copy, edit, move, and back up resource files outside the IDE. Within the IDE, you can view a project's resource files by selecting **Open to** > **Resources** from the project's pop-up menu. You can also delete, rename, or open resource files from the IDE.

Every time you create a project in the workspace, a *project resources directory* is automatically created for that project. VisualAge for Java looks for resource files in this directory when you run programs or applets in that project, and when you export your code.

Every project in the workspace has its own subdirectory in x:\IBMVJava\Ide\project_resources*project*, where x:\IBMVJava is the directory where VisualAge for Java is installed and *project* is the name of the project. This is where project resources are stored before a project is versioned. When a project is versioned, its resources are versioned as well. They are then stored in x:\IBMVJava\Ide\repository\ivj.dat.pr*project**DateStamp*. *DateStamp* is the time and date the project was versioned. When you purge a version of a project from the repository, its resources are deleted from the file system.

[ENTERPRISE] In a team development environment, when resources are versioned, they are stored in the directory where the shared repository resides. For more information, refer to the related information below.

RELATED CONCEPTS
Workspace

Repository

[ENTERPRISE] Version control for project resource files

RELATED TASKS

Including resource files in a project
Exporting code

Extensions

Extensions are packages of Java classes and associated native code that are stored in JAR files. Application developers can use extensions to extend the functionality of the core platform without having to add the extension API to the class path.

Extensions are stored in one or more JAR files. They are usually written in the Java programming language, but you can also implement them using platform-specific native code. Extensions may include properties, localization catalogs, images, serialized data, and other resources specific to the extension.

If you add an extension to a project, a directory called ext-resources (x:\IBMVJava\Ide\project_resources\ext-resources) is created for that project. You have to copy your extension files to this directory so they will be read by VisualAge for Java when you run your application.

RELATED CONCEPTS

Repository

RELATED TASKS

Adding an extension to a project
Including resource files in a project

Editions and versioning

Editions and versioning

In VisualAge for Java, whenever you work with any project, package, class, or resource file you are actually working with a specific *edition* of that program element. At any time, you can only have one edition of each program element in the workspace. To see which editions are in the workspace, click the Show Edition Names  button.

Resource files do not appear in the workspace. To work with the resource files for a particular project, from the project's pop-up menu select **Open to > Resources.**

[ENTERPRISE] You will usually work with open and versioned editions; occasionally, you may also create scratch editions of program elements to experiment with. You will periodically release editions of classes and packages that you have been working on, to provide a baseline for the team and to make your changes easily available to them. Editions, releasing, and ownership are all fundamental to managing application changes in the team environment. Editions are discussed below; releasing and ownership are discussed as separate topics.

Open editions

Open editions are works in progress. Before you can make changes to an existing project, package, or class, you must create an open edition of it. The Workspace

can only contain one edition of a program element. In the repository, however, you can have multiple open editions of the same program element, with each one implemented differently. For example, if you are adding features to an application that you have customized for different industries, you might have multiple open editions of a package with the same name stored in the repository.

Open editions appear in VisualAge for Java windows with a timestamp, in parentheses, showing when they were created. Here is an example:

```
PackageA (3/28/98 4:21:15 PM)
```

Versioned editions
Versioned editions are editions that can not be changed. You version your open editions for the following reasons:

- To keep a copy of a program element at some meaningful point, so you can return to it at a later date. In the case of packages and projects, versioning freezes a specific configuration of the contained program elements, which must also be versioned.
- **[ENTERPRISE]** To make your changed classes available to other team members who are browsing the repository.
- **[ENTERPRISE]** To release a class into its containing package, thereby updating the team baseline. Classes must be versioned before they can be released.

Versioned editions appear in VisualAge for Java windows with version names, as opposed to the timestamps that identify open editions. When you version an open edition of a program element, VisualAge for Java can automatically assign a name for you, or you can specify your own name. Here are some examples of versioned editions:

```
PackageA 1.6.1
PackageB VersionBRel2
PackageC JS - Fixed print problems for CustomerX
```

Versioning does not prevent you from ever changing a program element again. To make changes, create a new open edition of the program element. To revert to an earlier version, replace the edition in the workspace with a different edition from the repository, and create an open edition based on that.

You will probably version your classes frequently, whereas you may leave packages and projects open for extended periods of time.

[ENTERPRISE] In the team development environment, version control is achieved by means of releasing editions into a team baseline. Only program element owners can release. See the list of related topics at the end of this document for links to more information on ownership, baselines, or releasing.

[ENTERPRISE] Scratch editions
Scratch editions are editions that no other users of the shared repository can see. Scratch editions appear in VisualAge for Java windows with < > around the edition name:

```
PackageA <1.0>
```

Scratch editions are discussed separately.

[ENTERPRISE] Undefined editions
You may see a class or interface whose edition name is “undefined”:

```
ClassA Undefined
```

This means that someone has created a class or interface, but has never versioned or released it. VisualAge for Java has reserved the new program element's name in the shared repository. Such editions are also marked with the undefined symbol.

[ENTERPRISE] Tools for managing your editions
VisualAge for Java provides two tools for working with editions in a team development environment:

- The Managing page of the Workbench window consolidates information about all the editions that are in the workspace, and is a convenient place to perform activities such as versioning and releasing.
- The Management Query tool helps you search for program elements in the workspace by edition status. Open it by selecting **Management Query** from the **Workspace** menu.

You can also view edition details, such as status and ownership, by selecting **Properties** from a program element's pop-up menu.

RELATED CONCEPTS
Projects and other program elements
Workspace
Repository
Version control for resource files

RELATED TASKS
Creating an open edition
Versioning a program element
Replacing editions in the workspace (reloading)

Version control for project resource files

VisualAge for Java, Enterprise Edition, Version 3.5 includes two new functions for resource files - versioning and releasing. *Resource files* are files that are not Java source files or bytecode. For example, HTML files, image files, audio clips, or SQLJ source files.

Versioning project resource files
Any project resource files contained in the project owner's local project_resources directory are automatically versioned when the owner versions the project. Since only the owner can version the project, all other developers on the team must provide a copy of their project resource files to the project owner if they want them versioned. Team members can do this by individually releasing project resource files, which the project owner can then access on the server.

Releasing project resource files
When a project owner versions a project, all the project resource files in the project owner's local project_resources directory are automatically released, updating the project baseline. Therefore, the project owner should make sure that the project resource files are correct when they version the project.

Project resource files can also be released individually by the members of the development team. When released individually, they must be released into open editions of the project. A team member may choose to explicitly release project

resource files while the team is working on an open edition, so that if anyone loads or reloads the project, they will get the latest changes.

If you are working with an open edition of a project and release a resource file, then modify it and re-release it, your previous edition of the file will automatically be replaced.

Team members can add, delete, rename, or replace resource files in their workspace but may only perform these same operations on released resources if they are the project owner or the owner of the affected resources. For example, a team member can create a local copy of a resource, but they cannot change the released edition of it, unless assigned ownership of it.

For a team member to own a released edition of a resource, the project owner must release the resource and then assign ownership to it.

You can assign ownership in the Resources page of the project browser or the Workbench.

Resource files do not appear in the Repository Explorer until they have been released.

Refer to the list of related tasks below for more information on performing these tasks.

Released resources in the file system
When resources are released, they are are stored in a directory in the same location as the shared repository. The name of the directory is the name of the repository with the suffix '.pr'. For example, if your repository is called `ivj.dat`, your directory would be called `ivj.dat.pr`.

Note: Never delete files directly from the file system. You should always work in VisualAge for Java to purge, copy or back up resource files.

For all platforms: Use caution when you are using clients and servers with different file systems where one file system is case-sensitive and the other is not. For example, if you have UNIX® clients and a PC server, and you have a resource called TEST.TXT and a resource called test.txt and you release them both, one will overwrite the other as they both cannot exist on the PC filesystem. Conversely, if you have a PC client and a UNIX server and you have a project with the resources TEST.TXT and test.txt, when you load those resources, one will overwrite the other as they both cannot exist on the PC file system.

Loading a project with project resource files into your workspace
When users load an edition of a project into their workspace, they can view and edit all the resources associated with a project.

If you replace an edition of a project in your workspace without releasing the resources, any changes you make to them will be lost.

When users load an edition of a project into their workspace, all of the resources associated with it are copied into the local project_resources directory for the project (any old resources in this directory will be deleted first). When a project is deleted from the workspace, the project resource files are deleted from the local project_resources directory. If you load an edition of the project into the workspace

that was created in Version 2.0 or 3.0x of VisualAge for Java and does not have any project resource files associated with it, the contents of the the project_resources directory will remain untouched.

RELATED CONCEPTS

Resource files and directories
Version control for Java program elements
Baselines, releasing and reloading

RELATED TASKS

Releasing a program element or resource file
Sharing resource files
Creating resource folders
Adding resources from the file system
Replacing a resource file with the released version
Changing the owner of a resource file or folder

Scratch editions

Scratch editions are private. Unlike open editions, you can not version them to make them available to other developers on the team. You can create scratch editions of projects or packages. You can not create a scratch edition of a class, but you can create open editions of classes contained in scratch editions of packages.

You *can not release* into a scratch edition of a package or a project.

You might create a scratch edition for any of the following purposes:

- To learn how someone else's code works. After you finish experimenting, delete the scratch edition from your workspace or replace it with another edition from the repository.
- To test a change that you think the owner should make. If you think your change is good, talk to the class owner about integrating it into an edition that can be versioned and released.
- To start development on a class when the containing package has been versioned and the package owner is not available to create an open edition for you. You can make your changes, test and debug them, and version your class edition, but you can not release it until the package owner creates an open edition of the package for you.

If you have configured your VisualAge for Java options to show edition names, your scratch editions will be designated with < > around the program element's version name:

```
PackageA <1.0>
PackageB 1.2
```

In the example above, PackageA is a scratch edition that was created from a versioned edition called 1.0. PackageB is not a scratch edition; it is a versioned edition.

You do not explicitly create scratch editions. VisualAge for Java automatically creates scratch editions from versioned editions, under the following circumstances:

- If you modify a class contained in a versioned edition of a package, and then save your changes. A scratch edition of the package is created in the workspace.

- If you replace the edition of a class in a versioned package with another edition of that class, a scratch edition of the package is created in the workspace.
- If you create a new open edition of a package, in a project that has been versioned, a scratch edition of the project is created in the workspace.

RELATED CONCEPTS

Editions and versioning
Baselines, releasing, and reloading
Ownership and team roles - overview
Package groups

RELATED TASKS

Creating a scratch edition
Creating an open edition
Versioning a program element
Releasing a program element
Replacing editions in the workspace (reloading)

Baselines, releasing, and reloading

Every edition of a containing program element has a specific configuration of editions of the program elements that it contains. For example, MyPackage 1.1 may contain MyClass 1.0 and YourClass 1.3, whereas MyPackage 1.2 may contain Myclass 1.1 and YourClass 1.3.

As long as a project or package is an open edition, its configuration of package or class editions can be changed. Program element owners change project or package configurations by *releasing* different package or class editions into them, or by deleting editions. Once a project or package is versioned, that particular configuration of editions is frozen.

In the VisualAge for Java team development environment, these configurations are called project or package *baselines*. Baselines determine which editions a team developer has available to work with, after adding a project or package from the shared repository or after replacing an edition in the workspace with another edition (*reloading*). Baselines allow developers to get a common view of the current state of the application, and to catch inconsistencies early.

A class owner can update a package baseline by releasing a class into the package. A package or project owner can update a project baseline by releasing a package into a project. Classes must be versioned before they can be released. Packages, on the other hand, may be released while they are still open editions. Releasing one or more open packages into a project has the effect of establishing a dynamic, or rolling, baseline for the project. As long as the project contains an open edition of a package, the *project's* configuration of classes is immediately updated every time a class is released into the contained package. The benefit of a rolling baseline is that team members can resynchronize in one step, by reloading the project instead of reloading individual packages or classes.

Since releasing affects every team member who reloads a baseline, changes should be tested before they are released. Classes may be versioned every day, but they should only be released when they are stable.

Periodically, project and package owners will preserve a baseline by versioning the project or package. At that point, all contained packages and classes must also be versioned. The result is a frozen configuration to which the team can return, if necessary.

For more information on baselines, see the team development scenarios that are listed as related topics, below.

RELATED CONCEPTS

Team development - overview
Ownership and team roles - overview
Editions and versioning
Team development scenarios - overview
Team development scenario - single package, multiple developers
Team development scenario - multiple packages, multiple developers
Sample life cycle of an application

RELATED TASKS

Building a team baseline
Releasing a program element
Finding unreleased editions in the workspace
Replacing editions in the workspace (reloading)
Managing editions of program elements
Adding projects and packages from the repository to the workspace

Unresolved problems

VisualAge for Java compiles source code incrementally. If you save source code that has a compiler error (also called an unresolved problem), VisualAge for Java displays a warning message. As a result, you see errors immediately.

You can cancel the save operation. In some cases, you can also save the source with the error. However, you cannot save the source in the following cases:

- a class or an interface has a Java syntax error
- a method has a Java syntax error inside its signature

When you fix an unresolved problem, VisualAge for Java fixes the problem throughout the workspace. This means that many unresolved problems disappear for you as you write code.

Even though a class contains unresolved problems, you may be able to run it. The debugger may open and suspend the program during execution.

Unresolved problems occur for many reasons, for example:

- the source code includes a Java syntax error
- the source code refers to a field, method, class, interface or package that is not declared (this could be as a result of deleting or renaming a method, field, class, interface, or package)
- the source code refers to a class or an interface that is not visible
- you import code and do not have all the packages on which the code depends

If you save a class that contains an unresolved problem, the class is marked with  .

If you save a method that has an unresolved problem, the method is marked with  . If the class of the incorrect method has no unresolved problem of its own, it is marked with  .

VisualAge for Java maintains a list of all the classes and methods that have unresolved problems and updates it when you save new or modified code. You can view the entire list from the **All Problems** page of the Workbench, or you view the list of problems in a single project by selecting **Problems** from any projects pop-up menu. In the **All Problems** or **Problems** page, or anywhere where you can select the class or method and see its source (except in a view of the repository) you can modify the source to fix the problem.

RELATED CONCEPTS
Development without files
Incremental compilation

RELATED TASKS
Creating a class
Creating a method
Creating an applet or application
Finding and fixing problems
Importing files from the file system
Saving changes to code
Debugging during the development cycle with the IDE debugger

The Scrapbook

The Scrapbook is a window that helps you organize, develop, and test ideas for your Java programs. In the Scrapbook, you can experiment with Java code fragments without specifying a containing class. The Scrapbook can have several pages. Contents of the pages may be saved to files, but are not saved in the repository.

The compilation context
The *compilation context* is the class you choose to contain the Java code fragment when you compile and run it. When you evaluate code in the Scrapbook, it is treated as though it were part of the compilation context, so the code inherits any imports from the selected class. It can also refer to protected or private fields and methods in the class and nonpublic classes in the same package.

Each Scrapbook page has its own compilation context, and does not interfere with other pages, so you can do things such as test the client and server parts of a program. The default compilation context is java.lang.Object.

Other uses for the scrapbook
The Scrapbook can open, read, and save to text-based files, including .java files, from the file system. It supports a variety of formats and origins, including NT, OS/2®, UNIX-based, Solaris, and Macintosh. You can run Java code that is in any of these types of files.

You can import Java code from these files into a class, interface, or method in the workspace.

The Scrapbook is also useful as a simple text editor for viewing and editing text-based files within the file system, and for making notes to yourself.

RELATED CONCEPTS
Development without files

RELATED TASKS

Organizing your ideas in the scrapbook
Experimenting with code fragments

Create Servlet SmartGuide

The Create Servlet SmartGuide allows you to create servlets and related Web resources files (HTML and JSP pages). Together, they make up a Web application. With the Create Servlet SmartGuide, you can import Java beans, and then generate HTML, JSP pages, and servlet configuration files from the beans. Please note that the Create Servlet SmartGuide supports JSP 1.0.

You can use the WebSphere™ Test Environment to test the generated servlets.

For details on the JSP/Servlet Development Environment and WebSphere Test Environment, see the online documentation. Both of these features are available with the VisualAge for Java, Professional and Enterprise Editions.

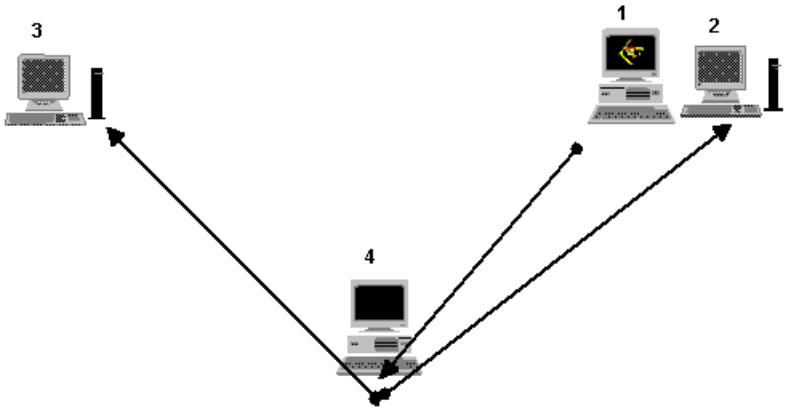
RELATED TASKS

Creating servlets using the Create Servlet SmartGuide

VisualAge for Java for the Network Station™

VisualAge for Java provides a productive environment for developing applications for the IBM Network Station. The VisualAge for Java IDE lets you develop, debug and prototype your code before deploying it to a Network Station. Deploying the code to the Network Station is as easy as exporting the compiled files to a shared directory on the Network Station Manager and configuring the application.

The following figure shows the suggested configuration for application developers, testers and users of the system.



The following are the key points of this application development scenario:

1. Developers have application development machines that support VisualAge for Java.
2. Developers have Network Stations to test their developed code in the Network Station environment.
3. Each tester or user of the system has a Network Station.

4. There is a single Network Station Manager that manages all of the network stations and has a shared directory that is exported through its web server.

RELATED TASKS

Deploying applets to the Network Station
Deploying applications to the Network Station

Choosing the right debugger for your program

VisualAge for Java comes with two debuggers:

- The IDE debugger, which is integrated in the IDE
- The Distributed Debugger, which must be installed as a separate component from the VisualAge for Java, Enterprise Edition installation CD

The IDE debugger
Use the IDE debugger to debug applets and applications running in the IDE, and code fragments in the Scrapbook. While the IDE debugger supports debugging external classes that are loaded at run-time, It will not debug classes running on other vendors' virtual machines or those running remotely.

The IDE debugger can work with multiple running programs concurrently, which can be useful, for example, when you are debugging client and server portions of an application. With this debugger, you can interchange different editions of methods and classes while the programs are running. The IDE debugger lets you set regular or conditional breakpoints, and it supports the debugging of inner classes.

The Distributed Debugger
This debugger debugs Java programs that you have exported to the file system. It can debug Java bytecode or Java programs that have been compiled with the AS/400® Feature, or with the High-Performance Compiler for Java for OS/390®. The Distributed Debugger is not installed when you install VisualAge for Java. You must install the debugger from the product CD by running setup and selecting **Install the Distributed Debugger** from the installation screen.

The Distributed Debugger offers the following advantages:

- **Remote debugging** While running a Java program on Windows®, OS/2, AIX®, OS/400®, or OS/390, you can debug it in the appropriate external debugger on a Windows or OS/2 front end. In other words, while running the program on one machine, you can debug it on another machine.
- **Debugging optimized code** The Distributed Debugger can debug Java programs that have been compiled into platform-specific executables or DLLs. The VisualAge for Java Accelerator for OS/390 and the AS/400 Feature all optimize and compile Java source or bytecode for a specific platform. The optimized executable code can be debugged by the debugger for that platform. This type of debugging is especially useful when the optimized program exhibits behavioral differences from the interpreted Java bytecode program.

Note: The Java source, bytecode, and debug information for the program must be available to the Distributed Debugger. Accelerators that optimize Java programs can generate debug information. The IDE can export debug information when it exports classes to the file system.

See the related links to information on the debugger for information on platform support, prerequisites to debugging, and debugger tasks.

RELATED CONCEPTS
The IDE debugger

RELATED TASKS
Debugging during the development cycle with the IDE debugger
Exporting for debugging with the Distributed Debugger

The IDE debugger

The IDE debugger is integrated with the VisualAge for Java integrated development environment. Use the IDE debugger to debug applets and applications running in the IDE. In the debugger, you can view running threads, suspend them, and inspect their visible variable values.

You can open the debugger manually or you can have it open automatically by setting breakpoints or specifying caught exceptions. It will also open automatically if an uncaught exception is thrown.

When a program is running, the debugger lists running threads. Suspend a thread to view a list of methods that represent the current stack state. Select one of these methods to inspect the visible variables in that stack frame.

While a thread is suspended you can do a number of things:

- View and modify visible variable values.
- Edit most methods' source code.
- Step into, over, or to the return statement of a method.
- Drop to a selected stack frame.
- Replace most methods with another edition.
- Evaluate expressions.
- Resume running the thread.
- Terminate the thread.

The debugger can work with multiple running programs concurrently, which can be useful, for example, when you are debugging client and server portions of an application. In the Debugger window, threads are grouped by the program that started them, for easy manipulation.

The IDE debugger lets you set regular or conditional breakpoints, and it supports debugging inner classes. Also, it can optionally generate a class loading and initialization trace.

RELATED CONCEPTS
Choosing the right debugger for your program

RELATED TASKS
Debugging during the development cycle
Opening the debugger manually
Suspending, resuming, and terminating threads
Setting breakpoints in source code
Configuring and setting conditions on breakpoints
Setting breakpoints in external classes
Selecting exceptions for the debugger to catch
Clearing and disabling breakpoints

Inspecting and modifying variable values
Stepping through methods
Modifying code while debugging
Evaluating expressions in the debugger
Generating the class trace
Setting debugger options

Chapter 2. Navigating the IDE

Browsing the workspace

The VisualAge for Java IDE lets you view all the Java programming information that you work with. It provides many ways of looking at the same information, with a variety of focuses and views.

The most effective way to learn how to view code and other information in the IDE is to experiment. The environment is complex and powerful, and if you explore, you will find it has a large degree of flexibility.

The descriptions of the browsers in the online help usually assume that the layout settings are set to the defaults that the product is shipped with. For example, most discussions of panes assume you have selected horizontal orientation (in the **Window** menu). Other settings that affect the appearance and behavior of browsers can be changed in the Options dialog.

The Workbench
The Workbench is the main window into the workspace. You organize and perform your work from the Workbench. It gives you a view of all the program elements that are in the workspace and their unresolved problems.

From the Workbench, you can open other windows and browsers that help you with your tasks.

Browsers
Browsers are specialized windows that help you with programming tasks. A browser gives you a focused view of an individual program element that is in the workspace and lets you work at a granularity that is finer than in the Workbench. The tasks that you perform in a particular browser are similar to those in the Workbench, except that they are focused on one program element (and, usually, the program elements that it contains). You can open a browser on a project, package, class, interface, or method.

Working in a browser focused on a particular program element has the following advantages over working in the Workbench:

- It filters out other program elements in the workspace, so that you can see more clearly the ones in which you are interested.
- You can examine multiple editions of the program elements.
- You can see a hierarchical tree or graph view of a specific class.
- You can see project resources.
- You can use the Visual Composition Editor from a class browser.
- You can get BeanInfo information from a class browser.
- You can see inner classes in the Class browser

To open a browser on a program element, select the element (in the Workbench or another browser) and select **Open** from the pop-up menu. To open the browser to a particular page, select the element, select **Open To** from the pop-up menu, and select the page from the list presented.

© Copyright IBM Corp. 1997, 2000

To get help with each page, pane, toolbar button, and menu in the workbench and browsers, go to the browser page in question and press F1.

Changing the default browsing style
By default, a new window is opened every time you select a program element to browse. You have the option of changing the browsing style so that when you open a browser on a program element, rather than opening a new window, the IDE simply opens the browser in the current window. Then you can move back and forth between "windows" by using the Back  and Forward  toolbar buttons, similar to most Internet browser interfaces.

To change the browsing style, select **Options** from the **Window** menu. On the General page, select the **Current browser** option for the **Open an item in** option.

RELATED CONCEPTS
Workspace
Development without files

RELATED TASKS
Moving between windows
Changing the IDE browsing style
Searching for a program element in the workspace
Setting IDE options

Changing the IDE browsing style

You can select one of the following options for browsing the workspace:

1. When you open an item, a new window opens for the program element browser, or
2. When you open an item, the browser opens in the current window, and you access previous items' browsers by using the **Back**  and **Forward**  toolbar buttons.

To select the method that you prefer:

1. Select **Options** from the **Window** menu.
2. Go to the General page.
3. If you want to open a new window each time you open an item (Option 1, above), select the **New browser** option for the **Open an item in** option.
4. If you want to open items in the current browser and use the **Back** and **Forward** buttons (Option 2), select the **Current browser** option for the **Open an item in** option.

Using the Back and Forward buttons
The Back and Forward buttons let you browse the workspace in a manner similar to using an Internet browser. The IDE keeps track of where you were, and you can return there by clicking the appropriate button.

Alternatively, right-click the **Back** or **Forward** button to view a pop-up list that shows the button's "memory." Select one of the items to return to it.

Keep in mind that certain changes to a program element (for example, deleting it) remove it from the list. When an item is deleted from the list, other items might also be deleted, which means you will have to navigate to them some other way.

RELATED TASKS
Browsing the workspace
Moving between windows

Moving between windows

There are two setup options for browsing the workspace:

1. When you open an item, a new window opens for the program element browser, and
2. When you open an item, the browser opens in the current window, and you access previous items' browsers by using the **Back** and **Forward** toolbar buttons.

To select the method you prefer:

1. Select **Options** from the **Window** menu.
2. Go to the General page.
3. If you want to open a new window each time you open an item (Option 1, above), select the **New browser** option for the **Open an item in** option.
4. If you want to open items in the current browser and use the **Back** and **Forward** buttons (Option 2), select the **Current browser** option for the **Open an item in** option.

Notes for browsing with Option 1
When you have opened several tools and browsers, your desktop may be getting crowded and difficult to manage. The IDE gives you several options for opening browsers you recently closed and for bringing into focus windows you already have open.

Opening a recently-closed browser
The **File** menu in any browser keeps a list of the most recently used browsers. Select the desired browser from this list to reopen it.

Finding an out-of-focus window
Each IDE window that is opened is registered with the operating system so that you can use the system menu to bring it into focus. For example, use the Taskbar or Alt+Tab in Windows.

Alternatively, select **Switch To** from the **Window** menu of any IDE window. The submenu lists all open IDE windows. Select the one you wish to bring into focus.

RELATED TASKS
Browsing the Workspace
Changing the IDE browsing style

Searching for program elements

Searching for a program element in the workspace

To find a program element in the workspace, use the Go To tool. The Go To tool finds and selects a program element within the current browser and page. It helps you select a particular program element in a long list of elements. The Go To tool can help you find declarations or references to a certain class, or to locate an element in your browser.

The Go To tool

If you have a browser open on, for example, a particular project, running a Go To search for a package that is not in the project will not find the package. Therefore, if you want to search the entire workspace for the program element, run the Go To search from the **Projects** page of the Workbench.

1. Start the Go To tool by selecting **Go To** from the **Selected** menu. In the submenu, select the program element type (for example, Package, Project, or Type) you want to go to. Only those program element types that the current page can display are available.
2. In the Go To dialog, all the available program elements in the page are listed. Select the one you want to go to from the list, or narrow down the list by typing in part of the name. Use the following wildcard characters to help with the search:

Wildcard Character	Represents
*	any string of characters
#	any single character

3. Click **OK** to go to the program element. Remember, you will remain in the same browser and page that you were in before, but the searched-for program element will be selected. If you want to open a browser for the element, double-click on it.

If you do not find the program element in the workspace, find it by using the Go To tool in the Repository Explorer, and then add it to the workspace.

RELATED CONCEPTS
Workspace
Repository

RELATED TASKS
Searching for a program element in the repository
Searching for declarations of and references to a program element
Searching for text in a source pane
Browsing the workspace

Searching the workspace by edition status, owner, or developer

You can use the management query tool to search the workspace using list of commonly-asked search questions or using different combinations of the following search criteria:

- Search by edition status (open, versioned, unreleased, scratch, or undefined)
- Search by kind of program element (type, package, or project)
- Search by scope (workspace or working set)
- Search by owner
- Search by developer

To open the Management Query window, select **Management Query** from the Workspace menu.

To use a pre-defined search question:

1. Open the list of search questions by expanding the search option drop-down list at the top of the window.
2. Select the question that you wish to have answered.
3. Click the **Start Query**  button. The results of your query will appear in the pane at side of the window.

To use the search tool:

1. Select **Custom** from the drop-down list of search options.
2. Use the Status, Scope, Owner, and Developer areas to specify your search criteria.

   To search by class developer, select **Type** from the Program element area.
3. Click the **Start Query**  button. The results of your query will appear in the pane at side of the window.

You can select items from the search results list and use their pop-up menus to version, release, or create new editions of them, or perhaps to remove them from the workspace. You can also edit the search results list and copy it to the system clipboard.

RELATED CONCEPTS

Editions and versioning
Ownership and team roles - overview
Workspace

RELATED TASKS

Managing editions of program elements
Finding unreleased editions in the workspace
Finding unversioned editions in the workspace
Viewing a program element's owner
Viewing a class or interface's developer
Searching for a program element in the workspace
Searching for a program element in the repository

Searching for a program element in the repository

To find a program element in the repository, use the Go To tool in the Repository Explorer. The Go To tool finds and selects a particular program element in a long list of elements.

To find a program element within the current page of the Repository Explorer:

1. Open the Repository Explorer by selecting **Repository Explorer** from a **Window** menu.
2. Start the Go To tool by selecting **Go To** from the **Names** menu. In the submenu, select the program element type (for example, Project, Package, or Methods) you want to go to. Only those program element types that the current page of the explorer can display are available.
3. In the Go To dialog, all the program elements in the page are listed. Select the one you want to go to from the list, or narrow down the list by typing in part of the name. Use the following wildcard characters to help with the search:

Wildcard Character	Represents
*	any string of characters

Wildcard Character	Represents
#	any single character

4. Click **OK** to go to the program element. Remember, you will remain in the Repository Explorer page that you were in before, but the searched-for program element will be selected. If you want to open a shadow browser for the element, double-click on it.

If you want to add the found element to the workspace:

1. Select the element.
2. Select the desired edition.
3. Select **Add** from the edition's pop-up menu.

Note: You can add a class or method to the workspace only if its containing package is already added.

RELATED CONCEPTS

Workspace
Repository

RELATED TASKS

Searching for a program element in the workspace
Searching for declarations of and references to a program element
Searching for text in a source pane
Adding projects and packages from the repository to the workspace
Adding classes and methods from the repository to the workspace

Searching for declarations of and references to a program element

To find declarations of or references to any program element in code in the workspace, use the Search dialog. Launch the Search dialog box by clicking the Searchbutton  on the toolbar, or by selecting **Search** from the **Workspace** menu.

In the Search dialog box, specify the following criteria:

1. **Search string** - If a program element was selected in the browser before you launched the search, its name will be inserted into this field by default. Use the default name, or type over it to Indicate the name or part of the name of the element you want to find. Use the asterisk (*) wildcard character to represent any number of characters in the string.
2. **Element** - Indicate the type of program element you want to find. If you select **Type**, for example, it searches classes and interfaces with the name pattern given in the **Search string** field.
3. **Scope** - Indicate the scope of code searched. If you select **Hierarchy**, for example, it searches the superclasses and subclasses of the selected class. If you select **Working Set** option, you can limit the search to a defined group or projects, packages or classes.
4. **Usage** - Indicate whether you want to search for references or declarations, or both.

Click **Start** to start the search. The number of occurrences found are shown as the search runs. Click **Stop** to halt the search and pass any results on to the Search Results window. Click **Cancel** to cancel the search without reporting any found occurrences.

The results of the search, if any occurrences were found, are added to the Search Results window, which also lists the results of previous searches. The **Search Descriptions** pane lists all the searches run in this session. If you select one of the searches, the (number of) **matches** pane shows the list of elements in which the declarations or references are made for the selected search. You can rearrange this list by selecting the  or  button from the toolbar to list the methods by method name or type name, respectively. If you select one of the elements, the **Source** pane displays the program element's source code, with the first declaration or reference highlighted.

You can re-run the search from the Search Results window to update the list of occurrences as the workspace changes.

RELATED CONCEPTS
Workspace
Repository
Projects and other program elements

RELATED TASKS
Searching for a program element in the workspace
Searching for a program element in the repository
Searching for text in a source pane

Searching for text in a source pane

To find all occurrences of a string in a **Source** pane (or **Comment** pane, for projects and packages):

1. Go to the pane in which you want to search for the string. You may select text from the pane to be the search string.
2. To launch the Find/Replace dialog, type **Ctrl+F**, or select **Find/Replace** from the pane's pop-up menu.
3. In the Find/Replace dialog, type the search string in the **Find what** field (if it has not been filled in already).
4. If you want to replace occurrences of the search string with another string, enter a **Replace with** string.
5. Select **Down** or **Up** to indicate the direction of the search from the cursor's current position in the pane.
6. Select the **Match case** check box if you want the found string to match the case given in the **Find what** field.
7. Select the **Wrap search** check box if you want the search to continue at the beginning of the pane once it has reached the end (or vice versa, for backward searches).
8. Click the appropriate buttons at the bottom of the pane to run the search. Click **Close** when you are done.

RELATED TASKS
Searching for a program element in the workspace
Searching for a program element in the repository
Searching for declarations of and references to a program element
Browsing the workspace

Accessing context-sensitive API help

To view the declaration of a given class, interface or class member:

1. Select it in a Source pane.
2. Select **Open on Selection** from the text's pop-up menu, or press F1.

A browser will open on the type associated with the selected text. If you selected a method, the method will be selected in the Members pane.

To access API reference help for a package, type, or member, select it in a Source pane and select **View Reference Help** from the text's pop-up menu. If the selected text is documented in the VisualAge for Java online information, your Web browser will open to the reference information for it. The online information contains the Java 2 platform (J2SDK) API documentation, and public API documentation for IBM tools and builders shipped with VisualAge for Java and installed on your machine.

RELATED TASKS
Accessing tools and Enterprise Access Builders
Browsing the workspace

Accessing tools and Enterprise Access Builders

VisualAge for Java Enterprise edition comes with several tools and enterprise access builders (also called features). If you did a custom installation, you may have opted not to install some of them. If you need a component you did not install, you will have to run the install program again and update your installation.

Adding features
To add a tool, builder, or other feature to your workspace:

1. Open the Quick Start window by selecting **Quick Start** from the **File** menu.
2. Select **Features** on the left side. Select **Add Feature** on the right side. Click **OK**.
3. Select the feature(s) you want to add, then click **OK**.

The Java code for the features you selected will be added to the workspace.

Accessing tools and builders
If a feature is a tool or builder with a user interface, you can access it through the **Tools** menu item in the **Selected** menu.

RELATED TASKS
Using the Quick Start window
Adding a feature to your workspace

Printing from the IDE

Printing program elements

The contents of any project, package, type, or method are printable from the IDE. To output the desired program elements to a printer:

1. Select the program element you want to print. You can select more than one, provided they are all the same kind of element (for example, you could select

more than one package or more than one method). When an element is printed, all of the elements it contains are also printed.

2. Select **Document** > **Print** from the element's pop-up menu. If the element is a method, the pop-up menu option is **Print**.
3. In the Print dialog box, select the options you want to use for printing the element. The following options are available:

Program element	**Available options (Select an option to print it.)**
Projects	• Comment • Package Listing • Include Packages
Packages	• Comment • Type Hierarchy • Type Listing • Include Types
Types (Classes and Interfaces)	• Hierarchy • Definition • Include Methods
Methods	• Entire Method • Declaration Only

   Selecting certain options disables others (for example, if you clear the **Contents of Packages** check box, the options for packages, classes, and methods are disabled). Click **OK** when you have selected the options you want.
4. Select the printer you want from the Printer Selection dialog box, and click **OK**. The selected elements are output to the selected printer.

To change the default printer setting, select **Print Setup** from the **File**menu. Select the desired default printer and click **Setup** to change its settings. Click **OK** to make the selected printer the default.

RELATED TASKS
Printing source code and other text
Printing the graph view of a class

Printing source code and other text

The source pane of any window and pages in the Scrapbook are printable. To print the contents on one of these panes, select **Print** from the pane's pop-up menu or **Print Text** from the window's **Edit** menu. Select the printer you want from the Printer Selection dialog box, and click **OK**. The contents of the pane are output to the printer.

To change the default printer setting, select **Print Setup** from the **File**menu. Select the desired default printer and click **Setup** to change its settings. Click **OK** to make the selected printer the default.

RELATED TASKS
Printing program elements
Printing the graph view of a class

Printing the graph view of a class

The graph view of a class provides an alternative way of looking at a class and its inheritance path. To print this view for a class:

1. In the Classes page of the Workbench or a class browser, select the class whose graph you want to print.
2. Click the **Graph Layout** button  from the title bar of the Classes pane. The pane changes to show the graph view of the class.
3. Select **Document**, **Print Graph** from the pop-up menu.
4. Select the printer you want from the Printer Selection dialog box, and click **OK**. The graph is output to the printer.

To change the default printer setting, select **Print Setup** from the **File**menu. Select the desired default printer and click **Setup** to change its settings. Click **OK** to make the selected printer the default.

RELATED TASKS
Printing program elements
Printing source code and other text

Customizing the IDE

Setting IDE options

The VisualAge for Java IDE is a flexible work environment that you can adjust to meet your needs and preferences. You can change the way the IDE appears and functions by changing settings in the Options dialog.

To open the Options dialog, select **Options** from the **Window** menu. The Options notebook contains settings for each of the customizable features. The settings are initially set to default values, but you can change the defaults to suit your work style or environment.

Press F1 within the dialog to get help with specific options in the Options dialog.

RELATED TASKS
Bookmarking program elements
Cloning windows
Locking windows open

Customizing key bindings

Key bindings allow you to map editor commands to your choice of shortcut keys. VisualAge for Java provides two standard sets of key bindings. You can choose to use a standard set, modify a standard set, or create your own set of key bindings in the Options notebook.

To open the Options notebook, select **Options** from the **Window** menu, which appears on all IDE windows. The Options notebook contains settings for each of the customizable features. Expand the **General** option and select **Key Bindings**. The settings that appear on the Key Bindings pane are set to default VisualAge for

Java values. The other default profile is an Emacs standard that contains a subset of Emacs functionality. To select this standard, open the Profile list and select **Emacs**.

Whether you use a standard or a modified profile, you may find it helpful to keep a list of key bindings near your keyboard. To print the list of key bindings, select the desired profile in the **Profile** list and click **Print Table**.

Modifying a profile for key bindings
You may want to modify some of the key bindings from a selected profile to suit your work style or environment. To modify key bindings:

1. Open the Options notebook, as described above.
2. In the Key Bindings pane, find the action that you want to modify in the **Action** list and select its corresponding key sequence.
3. Enter a new sequence by pressing the keys on your keypad. If you add more than one key sequence for an action, the sequences are separated with a comma.
4. Click **Apply** to save your changes or **Defaults** to return to the original default bindings for the current profile.
5. Click **OK** to close the window.

A warning dialog box opens if you try to set more than one action to a key sequence.

To remove a key binding from an action:

1. Find the action the you want to modify in the Action list and select its corresponding key sequence.
2. Press **Delete**.
3. Click **Apply** to save your changes or **Defaults** to return to the original default bindings for the current profile.
4. Click **OK** to close the window.

Note that you can not rebind the Delete key alone to any action, but you can bind an action to a combination of keys that include Delete, for example Ctrl+Delete.

Creating a new profile for key bindings
If you do not want to change a standard profile, for example if you are working on a shared machine, you can create your own personal profile.

1. Click **Add**. A dialog opens.
2. Enter a name for your profile.
3. Select a base model for your profile.
4. Click **OK**. Your new profile will appear in the list of profiles.

Note that VisualAge for Java IDE menu accelerator keys, when they are available, take precedence over the user-defined key bindings. For example, in the Workbench, Alt+S is the menu accelerator key for the Selected menu; you could choose to set Alt+S as the key binding for another action (for example undo), but then you would have to press Esc, then S, to use that key binding for undo when you are in the Workbench window. (The Esc key acts as a “sticky” substitute for the Alt key.)

Bookmarking program elements

The Workbench lets you bookmark program elements, making it quick and easy to return to a frequently-used project, package, class, interface, or member within the Projects page. You can bookmark up to five program elements.

Bookmarking a program element
To bookmark a program element:

1. In the Workbench, select the **Projects** page.
2. Select the program element you want to bookmark.
3. Select the bookmark button  , located in the top right-hand corner of the **All Projects** pane. A bookmark number appears next to the bookmark button.

If you hold the mouse pointer over a bookmark number, hover help will show the name of the program element associated with the number.

Moving to a bookmarked program element
To move to a bookmarked program element, click on the corresponding bookmark number. The All Projects pane will expand the tree view to show the program element, and the other panes will be populated with the appropriate information.

Removing a bookmark
To remove a bookmark from a program element, right-click on any bookmark number. Select **Remove Bookmark** from the pop-up menu, then select the bookmark that you would like to remove from the pop-up list.

Locking windows open

Locking a window helps prevent you from accidentally closing a window that you want open. If you try to close a locked window, a dialog will prompt you to confirm your intent, and will give you the option of removing the lock. The Log window, for example, is locked by default, because it displays messages about the IDE, which you would not see if the Log window was closed.

To lock a window open, select **Lock Window** from the **Window** menu; a checkmark will appear beside the menu item.

To unlock a window, select **Lock Window** again; the checkmark will be removed, and you can then close the window without receiving a warning.

Cloning windows

Cloning a window in the IDE creates an identical instance of the current window, which you can then navigate in normally. This is useful if you are part way through working with a particular program element, but need to do something in the same browser and then return to your current location.

For example, if you are browsing a package, MyPackage, editing a class called MyClass1, and you decide you need to quickly work with MyPackage.MyClass2, and then return to MyClass1. Rather than lose your place (and perhaps temporarily introduce errors) by exiting MyClass1, clone the package browser. Then browse to MyClass2, make the needed changes, and close the clone. The original browser will still be opened to MyClass1.

To clone a window, select **Clone** from its **Window** menu. You can clone the Workbench, program element browsers, inspectors, and the Repository Explorer.

RELATED TASKS
Browsing the Workspace

Defining code assist macros

Two code assist features help you write code quickly and without errors: keyword completions and macros.

Defining keyword completions
To view the provided keyword completions:

1. Select **Options** from the **Window** menu.
2. Go to the **Keyword completions** page.
3. Select a keyword completion from the Name list. Its definition is displayed in the Completion field.

The string in the Name list is what appears in the drop-down list when you invoke code assist in a Source pane (Ctrl+Space). The Completion is what is inserted in the source code when you select the name. The text "<|>" indicates where the cursor will be placed.

To edit a keyword completion, select its name and edit the Completion text.

To create your own keyword completion:

1. Click **Add**.
2. Enter a keyword completion name. It must start with a keyword, and you should make distinguishable from other keyword completions, so that when you see it in a list, you can differentiate between it and other completions for the same keyword.
3. In the Completion field, enter the actual text that you want inserted into code whenever you select this keyword completion. If you want to specify where the cursor will be left when you insert the keyword completion, use the text "<|>".
4. Exit the Options dialog.

To insert the keyword completion:

1. Go to a Source pane for a program element that you are editing.
2. Type the keyword and then type Ctrl+Space.
3. Select the keyword completion and press Enter.

Defining macros
Macros provide a way to quickly and easily insert any predefined text into source code. To view the provided macros:

1. Select **Options** from the **Window** menu.
2. Go to the **Macros** page.
3. Select a macro from the Name list. Its definition is displayed in the Expansion field.

Similar to keyword completions, the string in the Name list is what appears in the drop-down list when you invoke code assist in a Source pane (Ctrl+Space). The Expansion is what is inserted in the source code when you select the name. The text "<|>" indicates where the cursor will be placed.

Two special macros are defined:

- **timestamp** inserts the current date and time.

- **[ENTERPRISE] user** inserts the current workspace owner name.

These can be used in other macros by putting `<timestamp>` or `<user>` in the Expansion text of those macros.

To edit a macro, select its name and edit the Expansion text.

To create your own macro:

1. Click the **Add** button.
2. Enter a macro name. **Hint:** Start your macros with a number; this way all macros will be grouped together when you invoke code assist.
3. In the Expansion field, enter the actual text that you want inserted into code whenever you select this macro. If you want to specify where the cursor will be left when you insert the macro, use the text "<|>".
4. Close the Options dialog box.

To insert the macro:

1. Go to a Source pane for a program element that you are editing.
2. Type the start of the macro name and then type Ctrl+Space.
3. Select the macro and press Enter.

RELATED REFERENCES
Code assist

Creating programs and program elements

Creating a solution

A solution is a container that holds projects. Solutions are stored in the repository and are created in the Repository Explorer.

To create a solution:

1. In the Repository Explorer, select the **Solutions** tab.
2. From the **Names** menu, select **New Solution**.
3. Enter a name for the solution. Click **OK**. Your solution is added to the Solutions pane.
4. To add projects to it, select the solution and edition, then select **Add Project** from the **Contents** menu.
5. Select the projects and versions that you want to add to your solution. Click **OK**.
6. The projects you selected appear in the Contents pane.
7. To remove a project from your solution, select the project and, from its pop-up menu, select **Delete**.

RELATED CONCEPTS
Projects and other program elements

RELATED TASKS
Browsing the workspace
Creating a field
Creating a method
Creating an interface

Creating an applet or application
Composing beans visually
Generating method stubs

Creating a project

A project is the top-most program element in the IDE hierarchy of program elements. A project contains all the packages used for a particular work unit, such as an entire application.

To create a new project:

1. Click the **New Project** button  in the Workbench. The Add Project SmartGuide will start up (This SmartGuide is also used for adding projects from the repository to the workspace).
2. Select the radio button next to **Create a new project named**.
3. Provide a name for the project.
4. Click **Finish**.

The new project will appear in the Projects page of the Workbench. To add a descriptive comment for the new project, select it, and fill in a description in the **Comment** pane of the Projects page.

Once you have created a project, you can add other program elements (packages, classes, methods, and so on).

[ENTERPRISE] In the team development environment, any member of the team can create a project.

RELATED CONCEPTS
Projects and other program elements

RELATED TASKS
Browsing the workspace
Creating a package
Creating a class
Creating an interface
Creating an applet or application
Adding projects and packages from the repository to the workspace

Creating a package

A package is Java language construct that groups classes. You can add a package to any project in the workspace.

To create a new package:

1. Click the **New Package** button  in the Workbench. The Add Package SmartGuide will start up (This SmartGuide is also used for adding packages from the repository to the workspace).
2. Follow the SmartGuide instructions to create the new package.

The new package will appear in the Projects and Packages pages of the Workbench. To add a descriptive comment for the new package, select it, and fill in a description in the **Comment** pane.

Once you have created a package you can add classes and interfaces to it.

[ENTERPRISE] In the team development environment, only the *project* owner can add a package to a project.

RELATED CONCEPTS
Projects and other program elements

RELATED TASKS
Creating a project
Browsing the workspace
Creating a class
Creating an interface
Creating a field
Creating a method
Adding projects and packages from the repository to the workspace

Creating a sample applet or application

You can use the Create Applet SmartGuide to create an applet or application to your specifications, without writing any Java code. You can also use the SmartGuide to create some sample code in the applet, which you can run to see quick results.

The sample applet displays the words "Welcome to VisualAge" when run in the applet viewer. If the applet is threaded, the words scroll across the window.

In the following paragraphs, the term "applet" refers to both applets and applications, unless otherwise specified.

To create the sample applet:

1. Launch the Create Applet SmartGuide by clicking the **Create Applet** button  .
2. Enter names for the applet, project, and package.
3. Select the **Applet, JApplet** or **Other** radio button to specify the superclass (the class the new one extends). If you selected **Other**, click **Browse** to select from a list of all classes in the workspace.
4. Ensure both the **Browse applet when finished** check box and the **Compose the class visually** check box are cleared.
5. Click **Next**.
6. To run the applet in an applet viewer, select the **No** radio button.
7. The applet will display text that is either static or scrolling, depending on whether the applet runs on a thread. If you wish to create the scrolling (threaded) version, select the **Yes** radio button.
8. Click **Next**. You do not need to set any event listeners, so click **Next** again.
9. Select the **Write Example** check box.
10. Click **Finish**.

The SmartGuide creates the applet in the project and package that you specified.

When you browse the package, the applet will have a symbol () next to it that shows that it can be run.

To run the applet in the applet viewer, select **Run** > **In applet viewer** from the applet's pop-up menu. If you generated the main method, you can also run it as an application by selecting **Run** > **Run main** from the pop-up menu.

Making Changes to the Generated Applet
If you double-click the applet in the Workbench, the Source pane shows the applet declaration and its fields. You can change the default value of the fields to modify the applet.

For example, change the font from `TimesRoman` to `Helvetica`, and the text string from `"Welcome to VisualAge"` to `"This is my applet."` Save your changes by pressing Ctrl+S. Run the applet again to see the changes.

RELATED CONCEPTS
Creating an applet or application
Generating a customizable visual application
Running an applet from the IDE
Running an application from the IDE
Making run-time changes to an applet

Creating an applet or application

The Create Applet SmartGuide makes creating applets and applications a simple, easy-to-follow process that does not involve writing a word of code. Except where differences between applets and applications are specified, the term "applet" refers to both applets and applications.

1. Select the package you want the applet to belong to. This will set some defaults within the SmartGuide to make your work easier.

2. To launch the SmartGuide, click the **New Applet** button  on the toolbar of the Workbench or a browser. Follow the instructions in the SmartGuide to create and customize the generated applet.

Example: "Generate an example applet"

Use the Create Application SmartGuide if you want to create a customizable visual application. To launch the SmartGuide, click the **Create Application** button  on the toolbar of the Workbench or a browser. For more information, see "Generating a customizable visual application."

[ENTERPRISE] In the team development environment, you must be a member of the package group in order to add an applet to an existing package. If you are creating the package at the same time as the applet, then you must own the project that contains the package.

RELATED CONCEPTS
Projects and other program elements
Visual composition

RELATED TASKS
Generating a customizable visual application
Creating a class
Creating an interface
Adding methods to a type
Adding fields to a type

Running an applet from the IDE
Running an application from the IDE
Publishing code

Generating a customizable visual application

Many visual applications contain the same basic elements, such as a title bar, toolbar, and menus. While you can add all of these elements to your application in the visual composition editor, VisualAge for Java provides a SmartGuide that will create an application that contains these and other basic elements defined by you.

To create and customize a visual application:

1. Click **Create Application**  on the toolbar to open the Create Application SmartGuide.
2. Fill in the **Project** and **Package** text fields to let VisualAge for Java know where to generate your application.
3. Type a name for you application in the **Class name** field.
4. Select **Create Swing based application** or **Create AWT based application.**
5. Click **Next.**

In the Application Details page of the SmartGuide, type a title for your application in the **Title bar text** field, then select from the available options to customize your application. The following options are available:

- **Menu bar** Click **Details** to select which menu items you want to add.
- **Tool bar** Click **Details** to select the toolbar buttons.
- **Status bar**
- **Center and pack frame on screen** Aligns the frame and defines the size of any child windows.

You can also choose to generate the following dialogs:

- **Splash screen** An image that appears while your application is loading.
- **About dialog** Information about your application, for example a contact name or a copyright date.

After you select all the elements that you want for your application, click**Finish**. Your application is generated and opened in the Visual Composition Editor.

Creating servlets using the Create Servlet SmartGuide

Use the Create Servlet SmartGuide to create servlets and the related Web resources files.

Launching the Create Servlet SmartGuide
To launch the Create Servlet SmartGuide:

1. In the IDE, select a project, package, or class.
2. Right-click, and select **Add** >**Servlet**. The Create Servlet SmartGuide opens.

Using the Create Servlet SmartGuide

1. Specify the project and package if the fields are not already filled in. Specify the class and superclass.
2. Select **Inherit from PageListServlet** to generate a class that is extended from the class or subclass of com.ibm.servlet.PageListServlet.
   Attention: For this option to be available, you must add the WebSphere Test Environment feature to your workspace. The WebSphere Test Environment

feature is available in both VisualAge for Java, Professional and Enterprise Editions.
For information on the PageListServlet class, see the WebSphere Studio documentation. For information on the WebSphere Test Environment, see the online documentation.

3. Select **Import Java bean** to work with existing Java beans (for example, an EAB Command or an EJB access bean).
   For details on EAB Commands, see the Enterprise Access Builder online documentation. For details on EJB access beans, see the EJB Development Environment online documentation.
4. Select **Use Single Thread Model** to generate a servlet that runs in a single thread model (i.e. the code generated will implement the javax.servlet.SingleThreadModel interface).
5. Add the appropriate import statements, and indicate the interfaces as needed.
6. Click **Next**. When you select to import a Java bean, the Web Pages Configuration (1 of 2) page of the SmartGuide opens. (If you do *not* select to import a Java bean, then the Attributes page of the SmartGuide opens. See step #11 below.)
7. On the Web Pages Configuration (1 of 2) page, specify the Java bean you want to import, and specify the directories in which you want the HTML, JSP, and servlet configuration file to be generated. To accept the default directories, select **Use Create Servlet SmartGuide default Web resources path**.
   Attention: If the WebSphere Test Environment is added to your workspace, then the resource paths are as follows:
   For the generated JSP and HTML files:
   X:\IBMVJava\ide\project_resources\IBM WebSphere Test Environment\hosts\default_host\default_app\web
   For the generated servlets:
   X:\IBMVJava\ide\project_resources\IBM WebSphere Test Environment\hosts\default_host\default_app\servlets
   Otherwise, the default paths are:
   X:\IBMVJava\ide\project_resources\<*project name*>\<*package name*>\web
   X:\IBMVJava\ide\project_resources\<*project name*>\<*package name*>\servlets
   where X:\IBMVJava is the directory in which VisualAge for Java is installed.
   Attention: The generated file names will resemble the following: *aaa*Input.html, *aaa*Results.jsp and *aaa*.servlet.
8. To determine which scope to select for the generated JSP results page, see the JSP specification available at the Sun site.
9. Click **Next**. The Web Pages Configuration (2 of 2) page of the SmartGuide opens.
10. Click **Add** to select:
    - which Java bean properties you want displayed on the input page (HTML page),
    - which Java bean properties you want displayed on the results or output page (JSP page),
    - and which Java bean action methods to call.
      Attention: For input page properties, you can select from only primitive types.
      For results page properties, you can select from both simple and index properties. However, for complex objects, you must hand-code the toString() method of the Java bean. If you require an error handling page, you must hand-code the logic yourself.
      The SmartGuide does not support the index properties in the input page;

therefore, when using the index properties in the output page, proper initialization of the index properties is required. Otherwise, you will encounter the NullPointerException when running the JSP page.
Some methods may require input parameters. For such methods, a message box will pop up, reminding you to provide the parameters by editing the servlet. The line of code that calls the method is generated as comment line.

11. Click **Next**. The Attributes page of the SmartGuide opens.
12. When you have selected the modifiers, and the method stubs to be created, click **Finish** to generate the servlet code, input page, and output page.

RELATED CONCEPTS
Create Servlet SmartGuide

Creating a class

The Create Class SmartGuide makes creating classes a simple, easy-to-follow process that does not involve writing a word of code.

To create a new class:

1. Before you launch the SmartGuide, select the package you want the class to belong to, and if applicable, the class within that package that the new class will extend. This will set some defaults within the SmartGuide to make your work easier.

2. To launch the SmartGuide, click the **New Class** button  on the toolbar of the Workbench or a browser. Follow the SmartGuide instructions to add the new class to the workspace.

[ENTERPRISE] In the team development environment, you must be a member of the package group in order to add a class to an existing package. If you are creating the package at the same time as the class, then you must own the project that contains the package.

RELATED CONCEPTS
Projects and other program elements
Visual composition

RELATED TASKS
Browsing the workspace
Creating a field
Creating a method
Creating an interface
Creating an applet or application
Composing beans visually
Editing code in the Source View window
Generating method stubs

Generating method stubs

When you create a class with the Create Class SmartGuide, you can choose to generate stubs for the following methods:

- Methods which this class must implement (from interfaces that it is implementing, or abstract methods from a super class)
- Copy constructors that belong to its superclass
- equals, hashCode, finalize, main, and toString

To open the Create Class SmartGuide, click the **New Class** toolbar button  . After defining the class, select the appropriate options under **Which method stubs would you like to create?** After generating the class (and method stubs) you just have to implement the logic for the methods.

Generating method stubs after a class is created
If you have already created a class which needs to implement certain method, or for which you need field accessors, you can generated stubs for the missing methods automatically.

Implementing required methods
Required methods include:

- methods from interfaces which your class implements, and
- abstract methods inherited from a super class

To generate stubs for the required methods:

1. Select the class (which will have an problem symbol  next to it).
2. From the class' pop-up menu, select **Generate > Required Methods**.

RELATED TASKS
Creating a class

Creating an interface

The Add Interface SmartGuide makes creating interfaces a simple, easy to follow process that does not involve writing a word of code.

To create a new interface:

1. Before you launch the SmartGuide, select the package you want the interface to belong to. This will set some defaults within the SmartGuide to make your work easier.

2. To launch the SmartGuide, click the **New Interface** button  on the toolbar of the Workbench or a browser. Follow the SmartGuide instructions to add the new interface to the workspace.

After defining the interface in the SmartGuide, you can edit and add to the generated code in a browser.

[ENTERPRISE] In the team development environment, you must be a member of the package group in order to add an interface to an existing package. If you are creating the package at the same time as the interface, then you must own the project that contains the package.

RELATED CONCEPTS
Projects and other program elements

RELATED TASKS
Browsing the workspace
Creating a field
Creating a method
Creating a class
Creating an applet or application

Creating a method

To create a new method, constructor, or main() in a class or interface:

1. In the Workbench or a browser, select the class or interface to which you want to add the method.
2. Click the **New Method** button  in the toolbar or select **Add Method** from the class's pop-up menu. This will launch the Add Method SmartGuide.
3. Follow the SmartGuide instructions to create a new method.

When you exit the SmartGuide, the new method is created and compiled.

RELATED CONCEPTS

Projects and other program elements

RELATED TASKS

Creating a class
Creating an interface
Adding classes and methods from the repository to the workspace
Creating a field
Editing code in the Source View window

Creating a field

To create a new field in a class or interface:

1. In the Workbench or a browser, select the class or interface to which you want to add the field.
2. Click the **New Field** button  in the toolbar, or select **Add Field** from the class's pop-up menu. This will launch the Add Field SmartGuide.
3. Follow the SmartGuide instructions to create the new field.

When you exit the SmartGuide, the field is created and compiled.

RELATED CONCEPTS

Projects and other program elements

RELATED TASKS

Creating a class
Creating an interface
Creating a method
Editing code in the Source View window
Generating field accessor methods

Generating field accessor methods

When you add a field to a class with the Create Field SmartGuide, you can choose to generate accessor ("get and set") methods for the field.

To open the Create Field SmartGuide, click the **New Field** toolbar button  . After defining the field, select the **Access with getter and setter methods** check box, and set the appropriate options.

You also can add accessors to existing fields by doing the following things:

1. In your Workspace, select a class that contains fields that you want to generate accessors for.
2. From the Selected menu, select **Generate > Accessors**. A Generating accessors dialog box opens.
3. Select the fields that you want to create getter and setter methods for.
4. Complete the rest of the options in the dialog box.
5. Click **OK**.

VisualAge for Java will generate the accessors.

RELATED TASKS
Creating a field

Generating a serial universal identifier (UID)

You can generate a serialVersionUID for serializable classes by following these steps:

1. In the Workbench, select a serialized class that you want to add a serial UID to. A class is serializable if it implements the interface java.io.Serializable.
2. From the Selected menu, select **Generate > Serial UID**.
3. The serial UID is generated. It appears in the **Source** pane as a private field called
serialVersionUID.

RELATED TASKS
Creating a class

Using the Quick Start window

Several of the most frequent tasks you will perform in the IDE are collected in an easy-to-access window called Quick Start. To open the Quick Start window, select**Quick Start** from any IDE window's **File** menu, or press F2.

The list on the left-hand side shows the categories of Quick Start Tasks. Select one to see the available tasks, displayed in the right-hand side. To start a task, select it and click **OK**.

Basic tasks
The basic tasks for creating program elements launch the appropriate SmartGuide to help you create applets, applications, classes, interfaces, projects, and packages.

The Experiment with Code task opens a page in the Scrapbook. The page provides introductory instructions for learning what the Scrapbook can do. It guides you through learning the basic steps to evaluating code and variable values in the Scrapbook.

Team development tasks **[ENTERPRISE]** The team development tasks provide you with easy access to projects in the repository. Also, a team administrator can administer users. The **Management Query** option lets you search for program elements based on status, owner, or developer.

Repository management tasks
The **Compact Repository** task lets you duplicate the current repository, leaving out purged and opened editions.

[ENTERPRISE] You can use the **Change Repository** option to connect your workspace to another repository (shared or local), or to recover if the server fails.

Features tasks
With the **Add Feature** and **Delete Feature** options, you can add to your workspace projects that are provided with VisualAge for Java, but are installed as part of the repository. These projects include IDE samples, Data Access Beans, and several other features. A secondary window will appear to let you select which of these features to add or remove.

RELATED TASKS
Organizing your ideas in the scrapbook
Compacting a repository

Organizing your ideas in the Scrapbook

The Scrapbook window is a flexible text editor and test environment within the IDE. It lets you open and work with text-based files from the file system. It also lets you create your own notes to yourself, and it provides a test area for Java code fragments.

A Scrapbook page can contain any text you want. You can open files, copy and paste from other locations inside and outside the IDE, and write Java code or free-form text. For this reason, it is a good place to keep a to-do list and reminders to yourself, along with ideas for code. Contents of the pages are saved to files, not to the repository.

Opening the Scrapbook
To open the Scrapbook, select **Scrapbook** from the **Window** menu of any IDE window. Each time you start the Scrapbook, it will have only one blank page, regardless of what files were open when you closed the Scrapbook; you must open files you need again. For this reason, lock the Scrapbook open during a development session so that you do not inadvertently close it.

Adding and removing pages
Using multiple pages in the Scrapbook is a good way to organize your ideas and to provide separate testing contexts for different fragments.

To add an empty page, click the New Pagebutton  on the Scrapbook toolbar.

To remove a page, select its tab, and then click the Delete Pagebutton  . You cannot delete a page if it is the only one left.

Opening a file
To open a text-based file in the file system, select **Open** from the **File**menu, or use the shortcut key Ctrl+O. This creates a new Scrapbook page that contains the specified file. The title of the page is the name of the file.

You can edit this file and save your changes back to the file system by selecting **Save** from the **File** menu. When you are finished with the file, delete its page from the Scrapbook. Removing the page that contains a file does not delete the file from the file system; it removes it from the workspace.

Going to a specific line
To go to a specific line in a Scrapbook page, type Ctrl+G. In the Goto Line Number dialog, enter a line number and click **OK**. The cursor will jump to the specified line.

RELATED CONCEPTS
The Scrapbook
Workspace

RELATED TASKS
Experimenting with code fragments
Locking windows open

Experimenting with code fragments

There may be times you have a fragment of Java code that you want to try out before adding it to a class or project in the workspace. Along with being a good place to keep notes and lists of ideas, the Scrapbook also provides a flexible environment for testing and experimenting with any piece of Java code. Contents of Scrapbook pages are saved to files, not to the repository.

To open the Scrapbook, select **Scrapbook** from the **Window** menu of any IDE window.

There is an Experiment with Code page available from the Quick Start menu (**File > Quick Start > Basic**). The page provides introductory instructions for learning what the Scrapbook can do. It guides you through learning the basic steps to evaluating code and variable values in the Scrapbook.

Running code fragments in the Scrapbook
To run a code fragment in the Scrapbook:

1. On a page in the Scrapbook, type in Java expressions and statements, or copy and paste them from another source. Alternatively, open a file that contains Java code.
2. Select all or part of the code on the page by highlighting it with the cursor. Only the highlighted code will be run.
3. Select the **Run** button  from the toolbar.

While the page is running the code fragment, its page symbol changes to indicate that it is busy. No other code can be run from that page while it is busy.

Resetting a busy page
To stop a running code fragment:

1. Click the tab for a Scrapbook page that is running a code fragment.
2. Select **Restart Page** from the **Page** menu.

When a page is reset, all threads started from the page are terminated, and all classes in the code are uninitialized. The compilation context and the text on the page do not change.

Changing the compilation context
When the code fragment runs, the IDE assumes it belongs to a particular class and package that exists within the workspace. This class provides a *compilation context*

for the fragment, which determines what other program elements are inherited by and accessible to the fragment. Each Scrapbook page has one compilation context, which is by default `java.lang.Object`.

To select another compilation context:

1. Click the tab for the Scrapbook page for which you want to change the context.
2. Select **Run In** from the **Page** menu in the Scrapbook.
3. Select a package and a class or interface from the lists.
4. Click **OK**.

The information line at the bottom of the Scrapbook page indicates the compilation context for the page. All code fragments run from the page will use this compilation context.

Going to a specific line
To go to a specific line in a Scrapbook page, type Ctrl+G. In the Goto Line Number dialog, enter a line number and click **OK**. The cursor will jump to the specified line.

Inspecting and debugging code fragments
Just as you can inspect variables for code in the workspace, you can also inspect variable values in code fragments in the Scrapbook. To inspect a code fragment in the scrapbook, select the code that you want to inspect, and then click the **Inspect** button  .

If the selected code returns a result that can be inspected, an Inspector window is launched. If the selected code does not return a result that can be inspected, the following message is displayed on the page:

```
No explicit return value.
```

Similarly, you can debug the code fragment by using the IDE debugger. To launch the debugger, select the code you want to debug, and then click the **Debug** button  .

Example: Running code in the Scrapbook

RELATED CONCEPTS
The Scrapbook
Workspace

RELATED TASKS
Using the Quick Start Window
Organizing your ideas in the Scrapbook
Debugging during the development cycle with the IDE debugger

RELATED REFERENCES
VisualAge for Java IDE symbols

Writing and formatting source code

To create program elements such as classes, methods, and fields, use the VisualAge for Java SmartGuides, which will generate the basic code for you. To fill in the functionality of classes and methods, type in the source code in the Source panes

that most browser pages have. These Source panes are typical text editors, except that they save your code not to files but to the containing program element in the workspace and repository. The Workbench, program element browsers and various other windows such as the Debugger browser, the Scrapbook, and the Search Results window, have Source panes.

Formatting source code
On the Formatter page of the Options dialog (**Options > Coding > Formatter**), you can set the Source panes to automatically give you new lines when you start a compound statement or use and opening brace character.

On the Indentation page of Options dialog (**Options > Coding > Indentation**), you can change the way that source is indented as you type it in.

If you want to apply these formatting settings to methods and class declaration code that was written before you made the settings, or to code that you import from the file system, select **Format Code** from the **Edit** menu. The changes will be applied to the program element currently in the Source pane.

Warning: The **Format Code** action removes whitespace (blank lines) from methods and class declarations.

RELATED CONCEPTS
Development without files
Projects and other program elements
The scrapbook

RELATED TASKS
Browsing the workspace
Creating a class
Creating an interface
Creating a method
Printing source code and other text
Saving changes to code
Searching for declarations of and references to a program element
Searching for text in a source pane
Setting IDE options

RELATED REFERENCES
Code assist

Generating HTML documentation for classes

The Sun J2SDK includes a tool called a Javadoc doclet, which generates HTML files based on special comments in Java class files. VisualAge for Java uses the Javadoc doclet to make it easy for you to generate online documentation for classes that you create within the IDE.

Note: Documentation for the Java 2 platform (J2SDK), and some IBM Java Classes is provided in the VisualAge for Java online information. Click on “Reference” in the help banner frame to find links to this documentation.

Adding comments for Javadoc
When you create a program element in the IDE, a generic comment is generated

automatically for the element. You can customize the comment either in the Smart Guide while you are creating the element, or by editing the comment in the Source pane for the element.

The Javadoc doclet picks up comments that start with /** and end with */, and that are placed immediately before a class, method, or field declaration. You can use Javadoc tags, which start with @, within these delimiters format the comment output. For example, the following code is a Javadoc-style comment and signature for a method, getNameOfEmployee:

```
/** This method returns the name of the employee for the given employee number.
 * @param employeeNumber an int indicating the employee number.
 * @return a String containing the employee's name.
 * @throws an exception if the employee number is out of range.
 */
 public String getNameOfEmployee(int employeeNumber) throws Throwable {
 /* code omitted */
 }
```

Add comments of this nature to your own classes, interfaces, and methods. Most Java language reference books, including the J2SDK documentation, contain detailed information on the tags available for documenting classes with Javadoc. Add regular HTML tags within the text of the comment to format the output text.

By default, HTML tags in Javadoc comments are displayed in pale gray font to increase the readability of the surrounding text. To change the color of the font:

1. Select **Options** from the **Window** menu
2. Expand the **Appearance** node.
3. Select **Source**.
4. In the Colors list, select **HTML Tags**.
5. Click **Change** beside the foreground color and select a different color.

If the description of a class or its member changes, make the change in the source rather than in the generated HTML files. Any changes you make to the HTML files will be lost when you regenerate documentation.

Generating the HTML files

To generate HTML documentation for any class, package, or project:

1. Select the class, package, or project.
2. Select **Document > Generate Javadoc** from the element's pop-up menu. The Javadoc Smart Guide opens.
3. If you have already used the Smart Guide and have saved your Javadoc settings, click **Load Settings from File** and select your saved file to implement the same settings.
4. In the **Directory for Javadoc output** field, specify the directory for the generated HTML files, if you do not want to use the default provided.
5. Select which methods you want to generate documentation for, based on access level. Choose one of the following options:
   - Public methods only
   - Public and protected methods
   - Public, protected, and default access methods
   - All methods (includes private methods)
6. Select which doclet you want to use to generate the documentation.

7. Click **Finish** to generate the documentation using the default settings, or click **Next** to customize your settings.

The generated HTML files link to graphics files in an images directory. If you accepted the default directory for the generated HTML files, then the images are in the right place. If you did not accept the default location, copy the images to the location you specified by doing the following steps:

1. Create a directory called "images" in the directory you specified for the HTML files.
2. If you installed VisualAge for Java in C:\IBMVJava (the default location), go to C:\IBMVJava\ide\javadoc\images
3. Copy the files from this location to the new images directory.

You may find that some reference links between the HTML files are broken. Broken links occur if you do not generate documentation for a complete set of classes. Ensure the class path includes the path to the classes used by those for which you are generating documentation.

Viewing the Generated Documentation
Open the generated files in an HTML browser. The files tree.html, packages.html, and AllNames.html, if you generated them, are useful indexes to the contents of the generated class documentation files. Each class also has its own .html file containing the formatted documentation for its methods and fields. This .html file is named after the class and the package that contains the class, in the format *package.class*.html.

The output for the above example method and its Javadoc comment would appear in an HTML file for the class that contains getNameOfEmployee. A linked index at the top of the file lists all the members, followed by the documentation. The documentation for the method looks similar to the following example:

getNameOfEmployee

```
public String getNameOfEmployee(int employeeNumber) throws Throwable
```

This method returns the name of the employee for the given employee number.

Parameters:
employeeNumber - an int indicating the employee number.

Returns:
a String containing the employee's name.

Throws:
an exception if the employee number is out of range.

Adding projects and packages from the repository to the workspace

The workspace contains all the projects and packages with which you are working. The repository contains all previous or updated editions, and editions that have otherwise been moved from the workspace.

[ENTERPRISE] The repository also contains projects and packages that other users have created.

Adding a project to the workspace
If you are in the Workbench, add a project from the repository to the workspace by completing the following steps:

1. Click the **New Project** button  . The Add Project SmartGuide will start up. This SmartGuide is also used for creating a new project.
2. Select the radio button beside **Add project(s) from the repository**.
3. Select one or more of the available projects from the lists by enabling the desired project and edition check boxes. You can add only one edition of a given project.
4. Click **Finish**.

Alternatively, if you are currently using the Repository Explorer, add a project by completing the following steps:

1. Select a project, and then one of its editions.
2. Choose **Add to Workspace** from the edition's pop-up menu.

The project, including all packages and classes, and resource files contained in that edition of the project, will be added to the Projects page of the Workbench. You can browse it in the Project browser. The resource files are displayed in the Resources page.

[ENTERPRISE] When you add a project to the workspace, the package and class editions that get added are the editions most recently released into that edition of the project. Anyone on the development team can add projects to their workspaces, because this action does not alter the project edition's configuration of package and class editions in the repository.

Adding a package to the workspace
You can add a package from the repository to any project in the workspace, or to a project that does not yet exist, provided that the package has not been added to another project already. If you are in the Workbench, you can add a package from the repository to the workspace by completing the following steps:

1. Click the **New Package** button  in the Workbench. The Add Package SmartGuide will start up. This SmartGuide is also used for creating a new package.
2. Provide the name of a project to hold the package. If a project was selected in the Workbench when you launched the SmartGuide, its name will be in the **Project** name field by default. You can enter the name of an existing project or the name for a new project. If you enter a project name that does not yet exist in the workspace, the SmartGuide will create the project when it adds the package.
3. Select the radio button beside **Add package(s) from the repository**.
4. Select one or more of the available packages from the lists by enabling the desired package and edition check boxes. You can add only one edition of a given package.
5. Click **Finish**.

Alternatively, if you are currently using the Repository Explorer, you can add a package by completing the following steps:

1. Select a package, and then one of its editions.
2. Choose **Add to Workspace** from the edition's pop-up menu.
3. Select or provide a name for a project to which the package will be added.

If you specified a new project, it will be created, and then the package will be added to the specified package. You can browse the package in the Projects and Packages pages of the Workbench, or in the project and package browsers.

[ENTERPRISE] When you add a package to the workspace, the class editions that get added are the editions most recently released into that edition of the package. The project to which you are adding the package can not be a versioned edition. If the project is not an open edition, then a scratch edition of the project will be created. Only the project owner can open an edition of the project and add a package to it, since this action alters the project's configuration of contained packages and classes in the repository, which changes the team baseline.

RELATED CONCEPTS
Projects and other program elements
Editions and versioning

RELATED TASKS
Browsing the workspace
Creating a project
Creating a package
Adding classes and methods from the repository to the workspace
Searching for a program element in the repository

Adding classes and methods from the repository to the workspace

If you have deleted a class, interface, or method from the workspace, it still resides within the repository, and you can add it back to the workspace if you find you need it again. Because classes and methods rely heavily on their place in a hierarchy for their definitions, you can add one back only to the program element that previously contained it.

For example, if you deleted ClassA from Package1, then you can add ClassA back to Package1, but not to Package2.

Adding a class back to a package
To add a class (or interface) from the repository to the workspace:

1. Select the package that contained the class before the the class was removed from the workspace.
2. Select **Add**, **Class** from the package's pop-up menu.
3. In the Create Class SmartGuide, select the **Add from Repository** radio button.
4. Select the classes you want to add back by enabling their checkboxes.
5. Click **Finish**.

The classes will be added and compiled.

[ENTERPRISE] To add a class back to a package, the package must be an open edition and you must belong to the package group. If you add a class back to a package, you become the owner of the class within that edition of the package.

Adding a method back to a class
To add a method from the repository to the workspace:

1. Select the class that contained the method before the method was removed from the workspace.

2. Select **Add**, **Method** from the class' pop-up menu.
3. In the Add Method SmartGuide, select **Add from Repository**.
4. Select the methods you want to add back by enabling their checkboxes.
5. Click **OK**.

The methods will be added and compiled.

RELATED CONCEPTS
Projects and other program elements
Editions and versioning

RELATED TASKS
Creating a class
Creating an interface
Adding projects and packages from the repository to the workspace
Creating a method
Searching for a program element in the repository
Deleting program elements from the workspace

Importing files from the file system

To import Java source code files, bytecode files, and resource files from the file system to the Workspace:

1. Start the Import SmartGuide by selecting **Import** from the **File** menu.
2. Select the **Directory** option (for regular files) or the **JARfile** option (for files stored in JAR or zip files).
3. Follow the instructions in the remaining SmartGuide pages to select the files and the target project.

Note: Because the Import SmartGuide can look into JAR and zip files, you do not need to unzip them before importing files that are in them.

Imported Java code is compiled and unresolved problems that are introduced are added to the All Problems page. Open editions of the classes and interfaces are created in the VisualAge for Java repository, and added to your workspace. If you select to automatically version program elements, then they will be versioned after importing.

Resource files (any file imported that is not a .java or .class file) are copied into the local resources directory for the project.

If you import only bytecode (compiled Java code in .class files), and not source code, for a class or interface, then you will not be able to edit the source in the workspace. These classes are indicated in the Workbench and other browser lists

by the bytecode file symbol .

[ENTERPRISE] If the containing package or project in your VisualAge for Java workspace has been versioned, then a scratch edition of the package or project will be created. To prevent this, create an open edition of each project or package before importing.

RELATED TASKS
Exporting code

Finding and fixing problems
Including resource files in a project
Importing from another repository

Including resource files in a project

When developing a program in Java, you sometimes need to use external resources that are not part of the language. For example, it is quite common to use images and audio clips in Java applets.

VisualAge for Java makes certain assumptions about where resource files are located. For a project called ProjectX, it assumes that the resource files will be found in a directory called `IBMVJava\Ide\project_resources\ProjectX` on the client machine (where IBMVJava is the install directory for the product). All projects in the workspace have a subdirectory of the `project_resources` directory, even if no resources have been created for the project.

Viewing resource files inside the IDE
Right-click the project and select **Open to > Resources**. The Resources view provides a view of the disk on which project resources are stored. The title bar above the resources view shows the name of the directory that contains the resources (for example, e:\IBMVJava\ide\project_resources\My Project). Expand the directory tree to view the contents of the project resources directory.

From within the project resources view, you can delete, rename, or open project resource files by using menu options in the Resource menu. Creating, editing, copying, or moving resource files must be done outside the IDE in the file system.

Adding resource files to a project
When you create a resource file, import it with the Import SmartGuide. Select **Directory** as the import source, select **resource** as the file type, and specify the name of the target project. This will copy the resource file into a project's resource directory. From there, it will be accessible to classes in that project.

You can also add resources using the **Resources** page in the Project Browser. From its pop-up menu select **Add > Resources** and select the resource files you want to add to the browser. If you want to create folders to store the resources in select **Add > Folder.**

Importing packages that have resource files
If you import a group of files from the file system, any non-.java, non-.class files can be imported into the project resources directory. In the Import SmartGuide, select the **Resources** option and click **Details** to select which resource files to import.

Exporting resources
When you export a project to a JAR file, resource files are exported to the JAR file along with the classes.

Running applets and applications that use resources
When you run an applet or application from within the IDE, the default CLASSPATH contains the project resources directory so that it can find any resource files that the program uses.

When you run an applet from within the IDE, the code base for the applet is specified as the name of the resource directory. The URL of this directory will be returned by the `java.applet.Applet.getCodeBase()` method.

RELATED CONCEPTS
Resource files and directories
Extensions

RELATED TASKS
Using methods to get resources
Importing files from the file system
Exporting code
Adding an extension to a project

Loading external classes dynamically

The VisualAge for Java IDE lets you run code that refers to classes that are external to the IDE (that is, they have not been imported to the workspace, but rather exist only on the file system). These could be .class files or classes in a JAR or Zip file. They could also be applets on an HTML page on a web server. The classes will be loaded when the program that calls them runs. Use the `forName("Package.ClassName")` method in java.lang.Class to load and link external classes at run time.

To allow the workspace to compile and run code that uses external classes, you must specify the file system path to the external classes or the JAR or Zip file that contains them. This setting is done in one of the following places:

- The workspace class path - Select **Options** from the **Window** menu. Click the **Resources** option. Add to the path by clicking **Edit**.
- The program's class path - Select **Properties** from the applet or application's pop-up menu. Click the **Class Path** tab. Select the following checkboxes, as appropriate:
  - **Include '.' (dot) in class path** - Enabling this option includes the project's resource directory.
  - **Projects path** - Click **Compute Now** to determine classes in other projects that are referenced by this class. Enable this option to include the project resources directories from these other projects in the class path. Click **Edit** to add or remove other projects resource directories. These directories, if saved to the repository, are stored as relative paths to the project codebase (see the Applet or Program page of the class Properties notebook).
  - **Extra directories path** - Enable this option to add more directories to the class path. Click **Edit** to add or remove directories from the list. These directories are stored in the repository as absolute paths, unless the directories you select are subdirectories of the `project_resources` directory, in which case they are saved relative to the project codebase.

  The **Complete class path** field shows all the directories in the class path for the program. All external classes referenced in the code must be found on this path.

[ENTERPRISE] Warning: Class path settings for a program are saved in the repository if you select the **Save in repository (as default)** checkbox in the Properties notebook. However, the setting may include paths that are workstation-specific (local drives that might not exist on team-members workstations), or operating system-specific (file system notations are different on

different systems). Exercise caution when you click **Defaults** in the Properties notebook, because this resets the class path setting to what is saved as default in the repository; the resulting class path may be incompatible with your system. Especially beware of paths in the **Extra directories path** field.

Debugging external classes
The debugger settings include a class source path, in which you can set the path to source code for the external classes used by your application. This allows you to step into methods in external classes when you debug. Set the debug source path on the Debugging page of the Options dialog.

RELATED CONCEPTS
Development without files

RELATED TASKS
Setting the class path
Importing files from the file system
Setting IDE options
Setting breakpoints in external classes
Setting debugger options

Modifying program elements

Editing code in the Source View window

The Source View window enables users to view a complete class, including all of its methods. In this view, you can see and edit the class definition and all of the methods of a class at one time. If the class was imported from the file system, the Source View will retain the sorting and any spacing that was present before it was imported.

The Source View contains a method and field tree that is dynamically updated; as you create new methods and fields, they are listed in the window. Parse errors are dynamically logged.

To edit code in the Source View window, follow these steps:

1. In the Workbench, select **Open Source View** from the class pop-up menu. To view all the classes in a package, select **Open Source View** from the package pop-up menu.
2. The Source View window opens. The fields and methods of the class are listed in the **Members** pane. All the source code for the class is in the **Source** pane. Any problems with the code are listed in the **Problems** pane.
3. To find the code for a particular method or field, double-click it in the **Members** pane. The code will be highlighted in the **Source** pane.
4. To add an empty page, click the **New Page** button  on the Scrapbook toolbar. Enter code in the **Source** pane to create a new class.
5. To remove a page, select its tab, and then click the Delete Pagebutton  .
6. To open a different type, select **Page > Open Type** and select a type the **Open Type** window. Click **OK.**
7. When you have finished editing your code, select **Edit > Save**. If you want to return to the original version of your code without saving your changes, select **Edit > Revert to Saved.**

You can also access the Source View window by opening a class in the Class browser and selecting the **Source** tab. You can only work with one class at a time in this view.

RELATED CONCEPTS

Projects and other program elements
Visual composition

RELATED TASKS

Browsing the workspace
Creating a field
Creating a method
Creating an interface
Creating an applet or application
Composing beans visually
Generating method stubs

Saving changes to code

When you have made changes to a program element's source code by editing it in a Source pane, save the changes by selecting **Save** or **Save Replace** from the Edit menu.

The difference between **Save** and **Save Replace** matters when you have changed the name of a type or method by editing the source code. **Save** saves all the changes made since the last save to a *new* program element, leaving the old one untouched. **Save Replace** saves the changes, including the name change, to the existing program element. This may introduce problems if other program elements make reference to the old name; references to a program element do not change when you change its name.

You can change the shortcut keys for **Save** and **Save and Replace**in the Key Bindings page of the Options dialog (**Window > Options > General > Key Bindings**). On this page you can pick which action has shortcut key Ctrl+S and which has Ctrl+Shift+S.

The IDE will remind you to save your changes if you select a new program element or browser page to work with. For example, if you edit the source code for MethodA, and then select MethodB from a browser pane, a dialog box will prompt you to save your work.

Saving changes to a versioned edition creates an open edition of the program element.

If you make some changes and then need to undo them before you save, select **Revert to Saved** from the **Edit** menu.

When you save your changes, they are automatically compiled. Any problems encountered by the compiler are added to the Problems page of the current browser and the All Problems page of the Workbench.

RELATED CONCEPTS

Incremental compilation
Unresolved problems
Editions and versioning

RELATED TASKS
Deleting program elements from the workspace
Compiling code
Finding and fixing problems
Creating an open edition
Versioning a program element
Saving the workspace
Backing up the repository

Compiling code

The VisualAge for Java IDE automatically compiles Java source code into bytecode for you. The new bytecode is stored only in the workspace, not in the repository.

A new program element is compiled when you exit the SmartGuide that created it. Changes you make to source code are incrementally compiled when you save or when you browse to a different program element. Program elements affected by the changes are also incrementally compiled. **Note:** You can move to different browser pages, and can open other browsers, without causing the code to be compiled.

Program elements from the repository or the file system are compiled when you bring them into the workspace (except for .class files imported from the file system, which are already bytecode, and are stored as bytecode in the repository).

If the compiler finds an error in a program element, it adds the error to the All Problems page of the Workbench and to the Problems page of any browser that contains the program element.

RELATED CONCEPTS
Workspace
Incremental compilation

RELATED TASKS
Saving changes to code
Finding and fixing problems
Adding projects and packages from the repository to the workspace
Adding classes and methods from the repository to the workspace

Finding and fixing problems

VisualAge for Java automatically compiles every change you make to source code. If it encounters an error while compiling, it warns you that it encountered a problem, and gives you the option of fixing it before saving. If the compiler can determine possible alternatives to the problem code, it presents these alternatives to you in a list. You may select one of the alternatives to correct the problem.

If you save the element without fixing the problem, the IDE adds an entry to the All Problems page of the Workbench and to the Problems page of any browser that shows the program element with the problem.

Program elements that contain errors are marked with a red error symbol . If a class contains a method with an error, the class is marked with a gray error symbol .

To fix the problem, click the **Problems** tab to go to the Problems page, where all elements with problems are listed in a tree view. The compiler error message is listed next to the element that contains the problem.

When you select the problem element, the source code causing the problem is highlighted in the Source pane. You can fix the problem on the Problems page, or on any other Source pane.

When you fix the problem and save the code, the code is compiled and the entry associated with it is removed from the Problems page. Use the up and down buttons,  and  , in the Element pane title bar to move to the previous or next problem element. The shortcut keys, Ctrl+N and Ctrl+P will also move the cursor to the next or previous problem in the Problems page.

You can display the problems in a specific group of projects by opening the pop-up menu for the All Problems pane and selecting **Filter by Working Set**. If you have already specified a working set, select it and click **OK**. To create a working set, click **Add** and select the projects and packages for which you want to see problems. Give your working set a name and click **OK**.

If you try to run a class while it or its methods contain errors, the Debugger may open and suspend the program. Once you have fixed all problems, and the class runs, you may still experience run-time errors. Use the debugger to help determine the source of this type of problem.

You cannot export to the file system a class that contains errors (you may export it to another repository, though).

RELATED CONCEPTS
Incremental compilation
Choosing the right debugger for your program
The IDE debugger
Unresolved problems

RELATED TASKS
Saving changes to code
Running an applet from the IDE
Running an application from the IDE

Versioning a program element

Periodically, you will version your open editions of program elements to save copies of them at meaningful stages in their development. You will probably version your classes quite frequently; whereas you may leave packages and projects open for extended periods of time.

[ENTERPRISE] In the team development environment, versioning makes your changes visible to other members of the team, when they explore the repository or when they request a list of editions before replacing the one that is in the workspace. See the table below for considerations related to versioning projects, packages, and classes in the team environment.

To version an open edition:

1. In a browser or the Workbench window, select the program elements that you want to version. (You can make multiple selections by holding down the Ctrl key.)
2. From the program element pop-up menu, select **Manage** > **Version**. The Versioning Selected Items SmartGuide appears.
3. Follow the instructions in the SmartGuide.

To verify that a program element has been versioned, use its pop-up menu to view its properties. You can also see the new version names, instead of the timestamps that mark open editions, in the Workbench and in browser title bars and status lines. If you do not see timestamps or names, click the Show Edition Names 

button.

[ENTERPRISE] If you have just versioned a package, and you or other team members plan to continue changing classes or interfaces within that package, you should create a new open edition*immediately after versioning* the package. Otherwise, each developer who changes a class will end up with a different scratch edition of the package. If you want the team baseline to be updated whenever class owners release changes, release the open edition of the package when you create it.

[ENTERPRISE] The following considerations govern versioning:

Program element	Considerations
Project	• Only the project owner can version an open edition of the project. • Ensure that your workspace contains the latest editions of packages that make up the project, as this is the configuration that will be preserved by versioning. To load the latest editions that have been released by the package owners, select **Replace with** > **Released Contents** from the project's pop-up menu. • All contained program elements must be versioned and released before the project can be versioned: – Open editions of packages and classes that you do not own must be versioned and released by their owners, before you can version the project. – VisualAge for Java will automatically version all contained program elements that you own, when you version the project. – VisualAge for Java will automatically release all versioned packages and classes, regardless of owner, when you version the project.

Package	• Only the package owner can version an open edition of the package. • Ensure that your workspace contains the latest editions of classes that make up the project, as this is the configuration that will be preserved by versioning. To load the latest editions that have been released by the class owners, select **Replace with** > **Released Contents** from the package's pop-up menu. • All contained class editions must be versioned and released before the package can be versioned: – Open editions of classes that you do not own must be versioned by their developers, and released by their owners, before you can version the package. – VisualAge for Java will automatically version all contained classes that you developed, and release versioned classes that you own, when you version the package.
Class or interface	• Only the class *developer* can version the open edition of a class. (To see who the developer is, check the class's properties.)
Method	• Methods are automatically versioned when the containing class is versioned.
Project resource files	• Only the project owner can version an open edition of the project. Project resource files are automatically versioned when the project is versioned. • To load the latest editions that have been released by the project owner, select **Replace with** > **Released Resource** from the file's pop-up menu.

RELATED CONCEPTS
Editions and versioning

RELATED TASKS
Creating an open edition
Comparing editions of a program element
Replacing editions in the workspace (reloading)
Searching for a program element in the repository

Creating an open edition

To make changes to a program element, you must first create an open edition of it. Versioned editions can not be changed.

To create an open edition of a project, package, class, or interface that has been versioned, select **Manage** > **Create Open Edition** from the program element's pop-up menu.

In many cases, you can also create an open edition implicitly by opening a versioned edition in a browser, making changes, and then saving your changes.

An open edition appears in VisualAge for Java windows with a timestamp that shows when it was created. If you can not see the timestamp, click the Show Edition Names  button. Here is an example of an open edition:

```
PackageA (3/28/98 4:21:15 PM)
```

The following table summarizes considerations for creating open editions.

Program element	Considerations
Project or Package	• In VisualAge for Java, Professional Edition, an open edition is created automatically when you change any contained program element. • **[ENTERPRISE]** Only the owner can create an open edition of a project or package that has been versioned. • **[ENTERPRISE]** If you create an open edition of a package and the containing project is not already an open edition, then a scratch edition of the project will be created.
Class or Interface	• An open edition is automatically created if you open the class, interface, or a contained method in a browser, and save changes to it. • **[ENTERPRISE]** If you create an open edition of a class and the containing package is not already an open edition, then a scratch edition of the package will be created.
Method	• A new open edition is created every time you save changes to the method.

RELATED CONCEPTS
Editions and versioning

RELATED TASKS
Versioning a program element
Replacing editions in the workspace (reloading)

Copying or moving a program element

To copy or move a program element:

- Select the program element in any browser.

- Select **Reorganize** from the item's pop-up menu.
- From the submenu, select **Copy** to create an identical element in a different container; or select **Move** to move the element to a different container.
- In the Copy or Move dialog box, provide the name of the target container.

You can copy or move a package from one project to another project

You can copy or move a class from one package to another package.

You can copy or move a method or field from one class to another class.

Copying and moving program elements can introduce problems. The changes are compiled immediately, and problems are added to the Problems page of browsers.

If the program element happens to be a visual composite, delete its generated main() method, regenerate code for the composite, and save it.

RELATED CONCEPTS
Projects and other program elements

RELATED TASKS
Renaming a program element

Renaming a program element

To rename a program element:

1. Select the program element in a browser.
2. From the element's pop-up menu, select **Reorganize** > **Rename**.
3. Provide a new name.

The element's name will be changed and it will be compiled. Renaming program elements often introduces problems, since subclasses and method signatures that use the class still look for the old name. Click the **All Problems** tab in the Workbench to see the problems caused by renaming the program element.

[ENTERPRISE] You can rename a program element only in a scratch or open edition of the element.

RELATED CONCEPTS
Projects and Other Program Elements

RELATED TASKS
Copying or moving a program element
[ENTERPRISE] Creating an open edition
Finding and fixing problems
Searching for declarations of and references to a program element

Comparing two program elements

You can compare various combinations of program elements (projects, packages, classes, interfaces, and methods) that are in the workspace and in the repository. When you compare program elements, the results are displayed in a comparison results window.

Comparing program elements that are both in the workspace
You can compare any two program elements in the workspace, provided they are the same program element type (that is, you can compare two projects, but not a project and a package). To compare two program elements that are both in the workspace:

1. Select one program element.
2. Hold down the Ctrl key and select the other element.
3. Select **Compare With >** Each other from the elements' pop-up menu.

Comparing program element editions with editions in the repository
You can also compare the following combinations of program elements:

- Two editions of the same project, package, class, interface, or method, one in the workspace, the other in the repository.
- Two editions of the same project, package, class, interface, or method, both in the repository.
- Two packages within the same project edition in the repository.
- Two classes within the same package edition in the repository.
- Two methods within the same class edition in the repository.

See "Comparing editions of a program element" for more information on working with the Comparison Results window when two editions have been compared.

Working with comparison results

> **Note:** The following information applies to comparing two program elements that are both in the workspace. For information on comparing editions of the same element, see "Comparing editions of a program element."

From the Comparison Results window, you can do the following tasks:

- View the differences and step through them.
- Ignore specific differences.
- Show or hide ignored differences.
- Modify program element source.
- Replace parts of one element with parts of the other element.

The Differences pane lists all the difference between the two program elements and their contained elements, in a tree view. Beside each program element is a description of what the difference is: changed source code, changed comment, changed declaration, added element, or deleted element. The element on the right is being compared to the element on the left, so "Added method" for example, means that the class on the right contains a method that the class on the left does not contain.

Stepping through differences
When you select an item in the Differences list, the source for the two program elements that contain the difference is shown in the Source panes, with the difference highlighted. To move to the next difference click the Next Differencebutton  . To return to the previous difference, click the Previous Differencebutton  .

Ignoring differences
There may be some differences you want excluded from the list of differences. To

have a difference ignored, select it and click the **Ignore** button  . To show ignored differences, click the **Show Ignored Items** button so that it is in the down position  . Ignore differences will have parentheses around them in the Differences list. To hide the ignored differences, click the **Show Ignored Items** button so that it is in the up position  .

Modifying program element source
The Source panes in the Comparison Results window are the same as those elsewhere; you can edit the code and save it. There is code assist support (type Ctrl+Spacebar to get help with type and field names). To save changes, select **Save**from the pane's pop-up menu.

Merging the contents of the compared elements
You can change the source code of one of the program elements by merging into it the contents of the other program element, either by replacing a single difference in the code, or by replacing the entire program element. The following menu actions help with merging differences:

Menu option	What it does
Replace with Alternative (in Source pane pop-up menus)	Select an item in the Differences list. If the difference is changed source code or declaration, the corresponding code will be highlighted in each of the Source panes. Then select this option in the Right-Hand Source pane pop-up menu to replace the the highlighted code with the highlighted code in the Left-Hand Source pane. Alternatively, select this option in the Left-Hand Source pane pop-up menu to replace the highlighted code with the highlighted code in the Right-Hand Source pane.
Load Left (in the Differences pane pop-up menu)	Replaces the contents of the program element in the Right-Hand Source pane with the contents in the Left-Hand Source pane. If there is no corresponding program element in the Left-Hand Source pane, then it deletes the program element in the Right-Hand Source pane.
Load Right (in the Differences pane pop-up menu)	Replaces the contents of the program element in the Left-Hand Source pane with the contents in the Right-Hand Source pane. If there is no corresponding program element in the Right-Hand Source pane, then it deletes the program element in the Left-Hand source pane.

These menu options are available only if the merge can be done.

RELATED CONCEPTS
Projects and other program elements

RELATED TASKS
Comparing editions of a program element
Merging editions of a class or interface

RELATED REFERENCES
Code assist

Comparing editions of a program element

All editions of all program elements are saved in the repository. The IDE provides a browser for comparing the edition that is in the workspace with another edition from the repository. You can also use the comparison browser to merge source code from different editions into one edition.

To compare an edition in the workspace with another edition from the repository:

1. From any page of the Workbench window, select a project, package, class, interface, or method.
2. From the program element's pop-up menu, select **Compare** > **Another Edition**. A list will appear, showing all the editions of that program element that reside in the repository.
3. Select the edition with which you want to compare. A comparison results window, similar to the following example, will appear. The title bar displays the names of the two compared editions. The Differences pane lists the program elements that contain differences.
4. Select a program element from the Differences pane. The source code for the two compared editions of that program element will appear, side by side. Differences between the two editions are highlighted in each source pane.
5. To move quickly through the list of program elements in the Differences pane, select **Next Difference** or **Previous Difference** from the pop-up menu, or type **Ctrl+N** and **Ctrl+P**.

You can also compare two editions that are in the repository:

1. Open the Repository Explorer window.
2. Hold down the Ctrl key to select the two editions that you want to compare.
3. Select **Compare** from the pop-up menu.

RELATED CONCEPTS
Repository
Workspace
Editions and versioning

RELATED TASKS
Comparing two program elements
Merging editions of a class or interface
Browsing the workspace
Searching for a program element in the repository
Searching for a program element in the workspace

Merging editions of a class or interface

You can use the VisualAge for Java comparison browser to reconcile source code differences between two editions of a class or interface.

[ENTERPRISE] In the team environment, a class developer or class owner may compare editions to review changes made by other members of the team, and to

merge those changes into a single open edition. The person who does the reconciliation would then version the merged edition, so the class owner can release it.

For information on using the comparison browser, see the related topic on comparing editions.

To update an open edition of a class or interface that is in the workspace, with changes from another version in the repository, do the following steps:

1. From the Workbench window, select a class or interface and compare it to another edition. The comparison results window will open.
2. As you select each method that is listed in the Differences pane, the two source panes will show you where the editions do not match. The source pane marked with an asterisk (*) contains the edition that is in the workspace. This is the edition into which you must merge your changes, if you want to save the results later.
3. To merge *selected* differences from one edition into another, as they are highlighted, open the pop-up menu in the target source pane and select **Replace with Alternative**. You will see the source code change.

   To merge *all* source code differences from one edition into another, select **Load Right** or **Load Left** from the method's pop-up menu in the Differences pane. **Load Right** will merge source from the edition shown on the right into the edition shown on the left. **Load Left** will merge from left to right.

When you have removed all differences between two editions of a class or interface, that program element will be removed from the Differences pane of the comparison window.

RELATED CONCEPTS

Repository
Workspace
Editions and versioning

RELATED TASKS

Comparing editions of a program element
Comparing two program elements
Browsing the workspace
Searching for a program element in the repository

Managing your workspace

Adding a feature to your workspace

Before you can use some of the tools in VisualAge for Java, you must add the feature to your workspace. When you add the feature, the required projects are loaded from the repository.

To add a feature:

1. Select **File > Quick Start**. The Quick Start dialog box opens.
2. In the left pane, select **Features**; then in the right pane, select **Add Feature**.
3. Click **OK**. The Selection Required dialog window opens, listing the features that are currently available in the repository.

4. Select the feature that you want and click **OK**, to add the projects for that feature to the workspace.

If the feature that you want does not appear in the list, you may not have installed the component yet. Refer to the VisualAge for Java README file for instructions on installing additional components.

RELATED CONCEPTS
Projects and other program elements
Editions and versioning

RELATED TASKS
Browsing the workspace
Creating a project
Creating a package
Adding classes and methods from the repository to the workspace
Searching for a program element in the repository

Replacing editions in the workspace (reloading)

The workspace contains only one edition of any program element at a time. The repository contains all editions. At times, you will want to replace the edition in the workspace with an earlier edition from the repository, for example to back out code changes. At other times, you will want to replace it with a newer edition, to catch up with changes that other team members have made. In all cases, the edition that you replace will be removed from the workspace but it will continue to exist in the repository.

Replacing the edition that is in the workspace is also called reloading. Reloading a project or package also reloads the contained packages and classes. Reloading a method removes all breakpoints from the method.

To reload a project, package, class, interface, or method, select **Replace with** from the program element's pop-up menu in the Workbench or a browser. A cascaded menu shows you what replacement options are available. You can replace more than one program element at a time by holding down the Ctrl key when you make your selections.

To verify exactly what you have in the workspace after reloading, click the **Show Edition Names**  button.

[ENTERPRISE] In the team development environment, reload projects or packages to synchronize with a team baseline. To do this, select **Replace with** > **Released Contents** or **Replace with** > **Released** from the project or package's pop-up menu in the Workbench.

Choosing a specific edition to reload
To see the editions that are in the repository and replace the edition in the workspace with one of them:

1. Select **Replace With** > **Another Edition** from the program element's pop-up menu. The Select Replacement for... window will appear, listing all of the editions that are in the repository.
2. Select an edition from the list. The bottom pane of the Select Replacement for... window will show the program source.

3. Click **OK**.

The edition that you selected will be loaded into the workspace.

[ENTERPRISE] Reloading a team baseline
To reload all of the classes for a package in the workspace, select **Replace with > Released Contents** for the package. The class versions that have most recently been released *into that edition of the package* will be loaded into the workspace. Similarly, selecting **Replace with > Released Contents** for an edition of a project will reload all contained package and class editions.

To reload one or more individual classes, select **Replace with > Released Edition** from their pop-up menu.

RELATED CONCEPTS
Editions and versioning
Repository
Workspace

RELATED TASKS
Searching for a program element in the repository
Creating an open edition
Versioning a program element

Releasing a program element or resource file

In VisualAge for Java, Enterprise Edition, you *release* a class, package or resource file to update the team baseline. A baseline is the combination of class editions that make up a specific edition of a package, or the combination of package editions that make up a specific edition of a project.

Releasing is very important in the team development environment, because it determines which editions get added to the workspace when a team member performs any of the following actions:

- Adds a project or package to the workspace.
- Selects **Replace with Released Edition** for a package, class or resource file.
- Selects **Replace with Released Contents** for a project or package.

Resource files are automatically released when the project owner versions the project. Resource files can also be released individually. You can also release entire resource folders and all the resources contained in them. Team members can only release project resource files or folder that they own.

If resources are released individually, a subdirectory is created for the open edition of the project they are released into. Subdirectories for open editions are also created below the `projectname` directory; they also have a `Datestamp` name, which is when the project was created. (For example, if you created an open edition of your project at 5:10 on June 9, 2000), then `Datestamp` would be 2000-06-09 17.10.00)

A new subdirectory is not created every time a resource is released into an open edition; instead, the old version of the resource is replaced by the new one. After you have released a resource file, you cannot re-release it until you have modified it.

Once team members have released their resource files, the project owner can load the latest copies of the them into their workspace, and then version the project, setting a new baselines, making the newly updated resource files available to all the team members.

To release packages and classes:

1. In a browser or the Workbench window, select the editions that you want to release. (You can make multiple selections by holding down the Ctrl key.)
2. From the pop-up menu, select **Manage > Release**.

To release a resource file or resource file folder, follow these steps:

1. Select and double-click the project that contains the project resource files you wish to work with. The project browser opens.
2. Click the **Resources** tab.
3. Select the resource or resource folder that you want to release and from its pop-up menu select **Release**. When you release a folder, the files in it are *not* automatically released. You must select them, as well as the folder, if you want to release them.

Once you have released the editions, the unreleased (>) marker no longer appears beside their names in the Managing page of the Workbench window. The open edition of the containing project or package is updated with the newly released edition of the class, package or project resource file. Team members who had previously added the project or package to the workspace *are not notified* when the baseline is changed. Inform the team when you release a class, package or project resource file, so they can take the appropriate action to replace the edition in their workspaces.

To determine the last released edition, select **Properties** from the program element's pop-up menu and then select the **Info** page of the Properties notebook.

The following considerations govern releasing:

Program element	**Consideration**
Project	• Not applicable.
Package	• Both open and versioned editions of packages can be released. • The project into which you are releasing must be an open edition. • You must be the package owner or the project owner. • When you create a package, VisualAge for Java automatically releases the initial open edition of the package. • When you version a project, VisualAge for Java automatically releases any unreleased packages that it contains, irrespective of who owns the package.

Class	• Only versioned editions of classes can be released. • The package into which you are releasing must be an open edition. • You must be the class owner. • When you version a package, VisualAge for Java automatically versions and releases the contained classes that you own.
Method	• Methods are automatically released when you save them.
Project resource files	• You must be the project owner or the resource file owner • When released manually, resources can only be released into an open edition of a project.

RELATED CONCEPTS

Version control for Java program elements
Baselines, releasing and reloading
Version control for resource files
Ownership and team roles - overview
Workspace

RELATED TASKS

Finding unreleased editions in the workspace
Managing editions of program elements
Versioning a program element
Replacing editions in the workspace (reloading)
Searching the workspace by edition status, owner, or developer
Building a team baseline
Sharing resource files
Replacing a resource file with the released version

Deleting program elements from the workspace

To remove any program element from the workspace, select **Delete** from the element's pop-up menu in the Workbench or any browser. Deleting a program element also deletes all its contained elements; for example, if you delete a package, you also delete all the classes and interfaces in the package.

A message asks you to confirm your intent to delete the object, and reminds you that deleting an element from the workspace does not remove it from the repository. You can add the element back from the repository if you need it later.

Deleting a program element can introduce problems if, for example, classes remaining in the workspace extend a class that you delete. These problems will immediately be added to the Problems page for the appropriate program element.

[ENTERPRISE] In the team development environment, where all program elements are stored in a shared repository, deleting a package or class from the workspace also deletes that program element from the containing project or package. You

must own the package or class in order to delete it from the workspace. Anyone on the team can delete projects from the workspace, as this does not alter the team baseline.

RELATED CONCEPTS
Workspace
Repository

RELATED TASKS
Adding projects and packages from the repository to the workspace
Adding types and methods from the repository to the workspace
Finding and fixing problems

Saving the workspace

Your source code changes are automatically saved in the repository every time you save a method. By contrast, the following workspace information is only saved when you select **Save Workspace** from the **File** pull-down menu, or when you exit the IDE:

- A record of which specific editions of which program elements are in your workspace
- IDE options that you have set
- Sizes, positions, contents, selections, and bookmarks of VisualAge for Java windows
- Breakpoints that you have added to methods
- Contents of the Scrapbook
- Contents of the Log
- Contents of the Console
- **[ENTERPRISE]** The server and repository to which you are connected
- **[ENTERPRISE]** The name of the workspace owner

You should save the workspace when you have made significant changes to the items listed above. You should also save your changed methods frequently. These actions will ensure that your work can be recovered in the event of a system failure.

See the topics listed below for links to information on backing up the workspace and maintaining multiple versions of the workspace.

RELATED CONCEPTS
Workspace

RELATED TASKS
Saving changes to code
Adding classes and methods from the repository to the workspace
Adding projects and packages from the repository to the workspace
Replacing editions in the workspace (reloading)
Setting IDE options
Recovering the workspace
Reinstalling the workspace

RELATED REFERENCES
Important files to back up

Providing a standard workspace

In the team development environment, you may wish to provide a standard workspace at the beginning of a project, to ensure that team members start with the same editions of the same program elements. You can do this by copying the workspace file, ide.icx.

The following procedure is an example of how you can copy the workspace from one client (the source) to another (the target).

1. On the source client, connect to the shared repository.
2. Add the desired projects and packages from the repository to the workspace, and delete projects and packages that are not desired. The result is your standard workspace. See the comments about password validation, below.
3. Exit the IDE. The workspace is saved as the ide.icx file on the source client's workstation. The server and repository names are saved in the client's ide.ini file.
4. Using file system commands, copy the source client's ide.icx and ide.ini files. For example, you might call the copies team1.icx and team1.ini. You may wish to store these on the server, in a directory to which the team has read-only access.
5. On the target client workstation, preserve the existing ide.icx and ide.ini files by renaming them, for example to icx.old and ini.old.
6. Copy team1.icx and team1.ini into the target client's VisualAge for Java program directory, naming them ide.icx and ide.ini.
7. Start the IDE on the target client. The workspace will connect to the shared repository and will contain the desired projects and package editions. By default, the workspace will be owned by the user who created the standard workspace.
8. Change the workspace owner.

In an environment where password validation is enabled for VisualAge for Java, to start the IDE on the target workstation you must provide the password of the user who owned the standard workspace when it was saved. To handle this situation, the administrator could create a dummy user who owns the standard workspace but has no other privileges, and team members could first start the IDE with the dummy user's password and then change workspace owner.

RELATED CONCEPTS

Team client/server configuration
Workspace
Repository

RELATED TASKS

Saving the workspace
Changing workspace owner
Adding users to the repository user list
Enabling password validation - overview
Connecting to a shared repository
Adding classes and methods from the repository to the workspace
Adding projects and packages from the repository to the workspace
Setting IDE options

RELATED REFERENCES

Important files to back up

Creating a scratch edition

Scratch editions are private. They reside in the workspace; no one else can see them in the shared repository. In the workspace, you can have scratch editions of projects or packages, open editions of classes contained in scratch editions of packages, and open editions of packages contained in scratch editions of projects.

If you have configured your VisualAge for Java options to show edition names, your scratch editions will be designated with < > around the program element's version name:

```
PackageA <1.0>
PackageB 1.2
```

In the example above, PackageA is a scratch edition that was created from a versioned edition called 1.0. PackageB is not a scratch edition; it is a versioned edition.

You may use a scratch edition to experiment, for example to learn how someone else's code works, or to test a change that you think the program element's owner should make. You can version program elements that are contained in a scratch edition, but you *can not release* them.

To create a scratch edition of a package, do *one* of these things:

- Modify a class contained in a versioned edition of a package, and then save your changes. A scratch edition of the package will be created automatically.
- Replace the edition of a class in a versioned package with another edition of that class. A scratch edition of the package will be created automatically.

To create a scratch edition of a project, do *one* of these things:

- Create a new open edition of a package that is a scratch edition. A scratch edition of the project will be created automatically.
- Create a new open edition of a versioned package contained in a versioned edition of a project. A scratch edition of the project will be created automatically.

To find all of your scratch editions at once, select the **Management Query** tool from the Workspace menu, and specify **Scratched** as one of your search criteria.

RELATED CONCEPTS

Scratch editions
Editions and versioning
Baselines, releasing, and reloading
Ownership and team roles - overview
Workspace

RELATED TASKS

Versioning a program element
Creating an open edition
Releasing a program element
Replacing editions in the workspace (reloading)
Searching the workspace by edition status, owner, or developer

Recovering the workspace

[ENTERPRISE] A server failure may prevent you from saving the workspace. You may be able to reconnect after the server is started, without losing any work.

If VisualAge for Java terminates abnormally, for example because of a power failure, the workspace is not saved. If something causes the workspace to become corrupted, you can not save it. In either of these situations, *you will not lose the source code changes that you have saved*; they are safely stored in the repository and can be re-added to the workspace if necessary. You *may* lose some workspace information, if you have made any of the following changes since the last time you saved the workspace:

- Added, deleted, or replaced editions of program elements in the workspace
- Changed your IDE options
- Modified the contents of the Scrapbook
- Added breakpoints to methods
- [ENTERPRISE] Changed to another repository or changed workspace owner

To recover from a corrupted workspace error or from an IDE failure:

1. If the IDE is still open, save any important Scrapbook pages and then close all VisualAge for Java windows. You will not be able to save the workspace.
2. [ENTERPRISE] If a server outage caused the client IDE to fail, confirm that the server has been restarted.
3. Restart the IDE. VisualAge for Java will attempt to recover the workspace and resynchronize it with the repository.
4. If the IDE starts successfully, the workspace might not display the open editions that you created or methods that you saved since you last saved the workspace. To retrieve these changes from the repository, re-add or replace the editions in the workspace from the repository.
5. Select **Save Workspace** from the **File** menu in any window.

Failure to recover a corrupted workspace
When you try to restart the IDE, you may see a message indicating that VisualAge for Java could not recover the workspace. If this happens, you must restore the workspace file, ide.icx.

If you have a backup copy of the ide.icx file, restore that. If you do not have a backup copy, you will have to reinstall the workspace that was provided with VisualAge for Java. See the list of related tasks at the end of this topic.

Once you have restored the workspace file and restarted the IDE, you can retrieve the work you had saved in the repository by re-adding or replacing your open editions in the workspace, and you can reset your IDE options. When you have recustomized the workspace, select **File > Save Workspace**.

RELATED CONCEPTS
Workspace
Repository

RELATED TASKS
Reinstalling the workspace
Saving the workspace
Adding projects and packages from the repository to the workspace
Adding classes and methods from the repository to the workspace
Replacing editions in the workspace (reloading)
Setting IDE options

RELATED REFERENCES
Important files to back up
Repository files

Reinstalling the workspace

To reinstall the original VisualAge for Java workspace file:

1. Delete the corrupted workspace file (ide.icx) from the VisualAge for Java program subdirectory.
2. Run the VisualAge for Java installation program. If you accept the default directory shown in the installation wizard and select the same components that you chose when you originally installed the product, then only the ide.icx file will be reinstalled.
3. Start the IDE. VisualAge for Java will connect the fresh workspace to the existing source code repository.
4. **[ENTERPRISE]** Choose a new workspace owner by selecting your name from the list of repository users.
5. As was the case when you first installed VisualAge for Java, the only projects that you see in the Workbench are the standard class libraries. The IDE options revert to their default settings. Re-add the editions of projects, packages, and classes that you need from the repository, and reset the IDE options that you prefer.
6. Select **File > Save Workspace**.

Your workspace should now be fully recovered and ready to use.

RELATED CONCEPTS
Workspace
Repository

RELATED TASKS
Saving the workspace
Recovering the workspace
Adding projects and packages from the repository to the workspace
Adding classes and methods from the repository to the workspace
Replacing editions in the workspace (reloading)
Setting IDE options

RELATED REFERENCES
Important files to back up
Repository files

Managing your repository

Backing up the repository

[ENTERPRISE] The following backup procedure is for a *local* repository, for example on a VisualAge for Java team client or a standalone workstation. Administrators who manage a shared repository on a team server should refer to the related task on backing up a shared repository.

The repository is where all of your source code is stored. The repository file that is provided with VisualAge for Java is called ivj.dat. This file should be backed up regularly.

To back up the repository:

1. Exit the IDE. This action disconnects the workspace from the repository.
2. Using operating system commands or a backup utility, back up \IBMVJava\Ide\repository\ivj.dat.

If your applications use resource files, such as audio clips or image files, it is recommended that you back up those files up at the same time as the repository.

To restore the repository:

1. Exit the IDE.
2. Rename \IBMVJava\Ide\repository\ivj.dat, for example to obsolete.dat.
3. Copy your backup to \IBMVJava\Ide\repository\ivj.dat.

RELATED CONCEPTS

Repository
Resource files and directories

RELATED TASKS

Including resource files in a project

RELATED REFERENCES

Repository files
Important files to back up

Purging program elements from the repository

Purging marks projects or packages for deletion from the repository. Purged program elements do not appear in the Repository Explorer window, but they exist in the repository and can be restored until the repository is compacted. Purging program elements does not free up disk space. It is a preliminary step to compacting, which creates a smaller repository.

When you purge an edition of a *package*, you also purge the classes, interfaces, and methods contained in that package. This provides the greatest benefits when the repository is compacted later.

When you purge an edition of a *project*, you are only deleting information about that project's configuration of packages and project resource files. You do not purge any contained program elements.

[ENTERPRISE] Program elements can only be purged by their owners or by the repository administrator.

Purging program elements

To purge projects and packages:

1. Delete the program elements from your workspace, prior to purging them from the repository.
2. **[ENTERPRISE]** Confirm that the projects and packages have been deleted from all other team members' workspaces.
3. From the **Window** menu, select **Repository Explorer**.
4. In the Repository Explorer window, select the **Projects** or **Packages** page, according to what you want to purge.

5. Select the **Project Name**, **Package Name**, or **Edition** that you want to purge. Selecting a project or package by name will purge *all* editions of that program element. Hold down the Ctrl key to make multiple selections.
6. From the pop-up menu of the selected elements, select **Purge**. A message will appear, asking you to confirm the purge operation.
7. Click **OK**.

The purged items will no longer appear in the Repository Explorer window. If you try to load a project that contained a purged package edition, an error message will indicate that the project could not be loaded because one of its packages "does not exist in the repository". (In fact, the package is still in the repository, and can be restored until the repository is compacted.)

RELATED CONCEPTS
Repository

RELATED TASKS
Searching for a program element in the repository
Compacting a repository
Restoring program elements
Backing up the repository

RELATED REFERENCES
Repository files

Restoring program elements

Purging marks projects or packages for deletion from the repository. After a program element is purged, it no longer appears in the Repository Explorer, but it continues to exist in the repository until the repository is compacted to free up disk space.

If the repository has not been compacted, purged program elements can be restored, allowing you to browse and add them to the workspace once again. To restore purged program elements, follow these steps:

1. From any **Window** menu, select **Repository Explorer**. The Repository Explorer window will open.
2. Select the **Projects** or **Packages** page, depending on what you want to restore.
3. To restore all editions of a project or package, place the cursor in the **Project Names** or **Package Names** pane. To restore selected editions of a project or package, place the cursor in the **Editions** pane. Select **Restore** from the pop-up menu. A list of restorable names or editions will appear.
4. Select the items you wish to restore to the current repository. Click **OK**.

The restored program elements will re-appear in the Repository Explorer.

RELATED CONCEPTS
Repository

RELATED TASKS
Purging program elements from the repository
Searching for a program element in the repository
Compacting a repository

RELATED REFERENCES
Repository files

Compacting a repository

Compacting reduces the size of the repository by eliminating the following program elements:

- All open editions
- All purged editions
- Versioned editions of classes that are contained in open editions of packages

The compacted repository, which contains only versioned editions, can be as much as 50% smaller than the original.

The procedure for compacting a repository depends on which edition of VisualAge for Java you use. If you use "VisualAge for Java, Enterprise Edition (page 78)," skip to the instructions that follow.

Compacting the repository in VisualAge for Java, Professional Edition
In VisualAge for Java, Professional Edition, compacting replaces the existing repository with a smaller repository. To compact the repository:

1. As a precaution, back up the repository (ivj.dat file) before compacting it.
2. Version the open editions of projects, packages, and classes that you wish to keep.
3. Purge any projects and packages that you wish to discard, (To minimize the size of the new repository, purge obsolete *package* editions. Purging an edition of a project removes information about the project but does not purge any of the program elements that it contains.)
4. Open the Repository Explorer window.
5. Select **Compact Repository** from the **Admin** menu. You will be asked to confirm that you wish to replace the repository.
6. Click **Yes** to proceed.

When the repository has been compacted, it will contain only versioned editions of projects and packages, and versioned editions of program elements that they contain.

Compacting a repository in VisualAge for Java, Enterprise Edition
In VisualAge for Java, Enterprise Edition, compacting copies from the existing (source) repository to another (target) repository.

You must be the administrator to compact a repository.

1. Ask team members to purge any versioned editions that they wish to discard, and to version any open editions that they wish to keep. (To minimize the size of the new repository, purge obsolete *package* editions. Purging an edition of a project removes information about the project but does not purge its contained packages, classes, or methods.)
2. Connect to the source repository.
3. Open the Repository Explorer window.
4. Select **Compact Repository** from the **Admin** menu. You will be asked to confirm that you wish to compact the repository.
5. Click **Yes** to continue. A dialog box will appear, asking you to select a new or existing repository to compact into.

6. Select **Use shared repository** and accept the IP address or host name of the server where the target repository will reside. If you are creating a new repository, enter its name in the **Repository name** field; if you are compacting into an existing repository click **Browse** and select it from the list of repositories on that server. (You may wish to compact into an existing target repository, if someone omitted to version necessary program elements before a previous attempt to compact the source repository.)
7. Click **OK** to proceed with compacting the repository. Versioned projects and packages, along with the versioned editions of their contained program elements, will be copied to the target repository.
8. Once you have a new repository that contains all program elements required by the team, instruct all developers to change to the new repository. To check whether anyone is still using the old one, issue the **emadmin list** command.

Keep the old repository on the server for a few days, in case team members need to reconnect to it in order to export additional program elements.

Users who do not own any projects, packages or classes are not copied when the repository is compacted. After the compaction, the Administrator will have to re-create them.

RELATED CONCEPTS
Repository

RELATED TASKS
Versioning a program element
Purging program elements from the repository
Backing up the repository

RELATED REFERENCES
Repository files

Importing from another repository

If you need solutions, projects, or packages that are in another repository, you can import them into your current repository and then add them to your workspace.

You can only import versioned projects and packages. Imported program elements retain their version names and comments.

[ENTERPRISE] Imported program elements retain their ownership settings. Owners, class developers, and package group members who do not yet exist in the target repository's user list are added to it automatically. Program elements imported from VisualAge for Java, Professional Edition, are owned by the repository administrator.

Procedure
To import projects or packages from another repository:

1. In the Workbench window, select **File > Import**. The Import SmartGuide will open.
2. Select **Repository** and click **Next** to go to the next page.
3. Continue following the instructions in the SmartGuide. After selecting information about the repository location, projects, and packages, click **Finish**.

The Log window will record which program elements have been imported. You can now browse the imported program elements from the Repository Explorer window. If you want to make changes to them, you must add them from the repository to your workspace.

RELATED CONCEPTS
Repository
Workspace

RELATED TASKS
Searching for a program element in the repository
Exporting to another repository
Adding projects and packages from the repository to the workspace
Importing files from the file system

RELATED REFERENCES
Repository files

Exporting to another repository

Exporting lets you copy solutions and versioned projects or packages to another repository, for example to exchange program elements with another developer. When you export your projects and packages, your project resource files are also exported. If the repository you are exporting to is called "sample.dat", then your project resources are exported to a folder called "sample.dat.pr".

[ENTERPRISE] You might export to promote your work to a test repository on another server, to divide a shared repository in two, or to create a standalone repository for working at home. For more information, see the related tasks on changing repositories, dividing a repository, and working at a standalone workstation.

To export projects or packages to another repository:

1. In the Workbench window, select at least one project or package, and then select **Export** from the pop-up menu. The Import SmartGuide will open.
2. Select **Repository** and click **Next** to go to the next page of the SmartGuide.
3. Continue following the instructions in the SmartGuide. After selecting information about the server, repository, projects, and packages, click **Finish**.

The Log window will record which program elements have been exported. Another user of VisualAge for Java Version can now import from the repository into which you have exported your program elements.

[ENTERPRISE] Developers who connect to the target repository can browse the exported program elements in the Repository Explorer window and add the exported program elements to their workspaces.

[ENTERPRISE] Exported program elements retain their ownership settings. Owners, class developers, and package group members who do not yet exist in the target repository's user list are added to it automatically.

[ENTERPRISE] If you are creating a new repository for development purposes, remember to export the base libraries on which your classes depend. If you forget to include any program elements, you can export them into the same target repository later. There are four base projects:

- IBM Java Implementation
- Java class libraries
- JFC class libraries
- Sun class libraries

RELATED CONCEPTS

Repository
Workspace

RELATED TASKS

Backing up the repository
Searching for a program element in the repository
Importing from another repository
Exporting code
Exporting bytecode to the file system
Exporting source code to the file system

RELATED REFERENCES

Repository files

Chapter 3. Running and debugging programs

Setting the class path

In VisualAge for Java, each runnable class is responsible for its own class path. The class path is necessary when running a class in order for the class loader to properly find the classes that your class references and the classes that they in turn reference.

You can specify a workspace-level class path that every class uses when it is run (similar to traditional CLASSPATH environment variables). This class path is also needed to find resource files your classes use. The workspace class path is set from the Resources page in the Options notebook.

In addition, each class must have a class path associated with it in order to run. Typically the class path needs to include every project (directory) containing classes the runnable class references or resources it uses. To set the class path for a class:

1. Select the class.
2. Select **Properties** from the class's pop-up menu.
3. Click the **Class Path** tab in the Properties notebook.

Or:

1. Select the class.
2. From the class's pop-up menu, select **Run** > **Check Class Path**.

Both of these methods open the class's properties notebook to the Class Path page.

The class path tab contains the following three-part definition for the class path:

- **Include dot ('.') in class path** This indicates whether or not the project your class is in, is included in the class path. Typically this is enabled, but if you are doing security testing you may want it disabled (classes in the project directory will be found by class look-up but will not be considered system classes).
- **Project Path** This is the list of projects that you ultimately require or classes you use require. It is computed by taking the transitive closure of all classes referenced by your class and adding their projects. You can automatically have this computed for you by clicking **Compute Now**. Alternatively, you can set it manually by clicking **Edit** and providing the project paths. You might define the project path manually if, for example, you want to use some of the project's resources even though your class does not reference any of its classes.
- **Extra Directories** In the event you have special directories containing resources that are not in specific projects, you can add these to the extra directories path.

  [ENTERPRISE] Typically you should avoid using this field. When using it , exercise care because if you specify a path such as c:\personal\mystuff, then users who access your class path settings saved to the repository and do not have that directory structure may experience run-time errors. Ideally you should place any such directories under project_resources. This way the IDE can compute a relative path, which all users who use the class may have.

The class path for your class is formed by concatenating these three settings plus the workspace path.

© Copyright IBM Corp. 1997, 2000

Once you have configured your class path you can save it locally (for example, if you are experimenting with a class) or you can save it into the repository. If you do not save it to the repository, then the information is only in your workspace.

To save the class path settings to the repository, select the **Save in repository (as default)** checkbox before clicking **OK**.

[ENTERPRISE] Once it is saved into the repository anyone else loading your class can then pick up those settings as the default.

Important! When you export a project or package, the class path information stored with the class in the *repository* is used. Any class path values stored only in your workspace are not exported.

For any class, to retrieve the default class path settings that are saved in the repository, click the **Defaults** button.

Debugging external classes
The debugger settings in the Options dialog include a class source path, in which you can set the path to source code for the external classes used by your application. This allows you to step into methods in external classes when you debug. Set the debug source path on the Debugging page of the Options dialog.

Setting the class path for classes that use JFC
If you created a class and wrote code for it that uses JFC, then you must update the class path for it to include the path to the JFC project directory. Otherwise, the class will not run.

To update the class path:

1. Select the class in a browser.
2. From the class's pop-up menu, select **Run**, **Check Class Path**. This will open the class Properties notebook to the Class Path page.
3. Next to the **Project Path** field, click **Compute Now**. The JFC project path will be added.
4. Select the **Save to Repository (as default)** check box.
5. Click **OK**.

RELATED TASKS
Loading external classes dynamically
Setting IDE options

Running an applet from the IDE

You can run applet classes within the IDE. Executable classes, including applets, are indicated with the Runnable symbol  in browsers.

To run an applet in the IDE's Applet Viewer, select **Run** > **In Applet Viewer** from the class's pop-up menu, or select the class and click the **Run** button  in the toolbar. This launches the applet in the Applet Viewer. Control the applet with options in the **Applet** menu, such as **Stop**, **Reload**, **Start**, and **Exit**. If the applet uses standard input and output, the Console window is launched.

Defaults for applet size and behavior are set in the applet's Properties notebook. To open the Properties notebook, select the applet, and select **Properties** from the pop-up menu.

You can also select **Run > In Applet View** with to set the properties as you run the applet.

RELATED TASKS
Running an application from the IDE
Making run-time changes to an applet
Creating an applet or application
Creating a quick, basic sample applet or application
Experimenting with code fragments
Loading external classes dynamically

RELATED REFERENCES
Applet Viewer

Running an application from the IDE

If a runnable class's main method is implemented, you can run it as an application from the IDE. Generating the main method is an option in the New Applet SmartGuide. Runnable classes are indicated with the Runnable symbol  in browsers.

Command line arguments and system properties for an application can be set in the class's Properties notebook. To open the Properties notebook, select **Properties** from the class's pop-up menu. You can also open the Properties notebook by selecting the class and selecting **Run > Run main with** from the pop-up menu. Go to the Program page of the notebook to set arguments and properties.

To run an application from the IDE, select **Run** > **Run main** from the class's pop-up menu, or select the class and click the **Run** button  in the toolbar.

If the application uses standard input and output, the Console window is launched.

RELATED TASKS
Running an applet from the IDE
Creating an applet or application
Creating a quick, basic sample applet or application
Experimenting with code fragments
Loading external classes dynamically

Making run-time changes to an applet

When you are running an applet in the IDE's Applet Viewer, you can edit the applet's source code and then see the results of your changes immediately.

To make a change to a running applet:

1. While the applet is still running, open the Workbench or a browser and find the program element you want to change. Select it to view its source code.
2. Change the source code as needed.

3. Save the changes by pressing Ctrl+S or Alt+S.
4. Go to the Applet Viewer. Select **Reload** from the **Applet** menu.

The applet will start running at the beginning, with the new change incorporated.

RELATED CONCEPTS
Incremental compilation

RELATED TASKS
Running an applet from the IDE
Creating an applet or application
Creating a quick, basic sample applet or application
Saving changes to code
Debugging during the development cycle with the IDE debugger
Experimenting with code fragments

RELATED REFERENCES
Applet Viewer

Debugging during the development cycle with the IDE debugger

The IDE debugger assists in debugging applets and applications running in the IDE. It is a live debugger: it always shows the exact current state of running programs.

The IDE debugger browser displays the following information:

- On the Debug page:
  - All currently running threads, grouped by program
  - When a running thread has been suspended:
    - The methods in the thread
    - The visible variables and their values for the methods
    - The source code for the methods
- On the Breakpoints page:
  - All methods in the workspace that have breakpoints set in them
  - The source code for the methods
- On the Exceptions page:
  - The exceptions that, when caught, will suspend the thread

From the debugger, you can launch Inspectors to look at and modify variable values for suspended threads, Watches to evaluate expressions as you step through a program, and an Evaluation window where you can evaluate an expression during debugging.

You can open the debugger manually while a program is running to inspect threads and variables. As well, the debugger will automatically open, with the current thread suspended, for any of several reasons:

- A breakpoint in the code is encountered
- A conditional breakpoint that evaluates to true is encountered
- An exception is thrown and not caught
- An exception selected in the Exceptions page is caught
- A breakpoint in an external class is encountered

Once the debugger is open and a thread is suspended you can work with the program in the following ways:

- Inspect visible variable values
- Modify most variable values
- Step through methods
- Modify source code for methods in the workspace
- Replace methods with other editions from the repository
- Modify, clear or disable breakpoints
- Evaluate expressions in the source pane or the Evaluation window
- Define expressions to watch as you step through programs

Using the debugger, you can optionally generate and view the class loading and initialization trace.

RELATED CONCEPTS

Choosing the right debugger for your program
The IDE debugger

RELATED TASKS

Opening the debugger manually
Suspending, resuming, and terminating threads
Setting breakpoints in source code
Configuring and setting conditions on breakpoints
Setting breakpoints in external classes
Selecting exceptions for the debugger to catch
Clearing and disabling breakpoints
Inspecting and modifying variable values
Stepping through methods
Modifying code while debugging
Evaluating expressions in the debugger
Watching an expression's value as you step
Generating the class trace
Setting debugger options

Opening the IDE debugger manually

To open the IDE debugger browser, select **Debug > Debugger** from any **Window** menu or click the Debugtoolbar button  . This will open the Debugger browser to the **Debug** page, which lists running threads, their states, and their methods' visible variables.

If you click the **Breakpoints** tab, or if you selected **Debug > Breakpoints** from the **Window** menu, then the Debugger browser opens to the Breakpoints page, which summarizes all methods in the workspace that have breakpoints set.

If one or more program are running when you open the Debugger, then the running threads are listed in the All Programs/Threads pane of the Debug page. If you suspend the thread, you can look at the methods on the thread, and inspect the methods' visible variables.

If you leave the Debugger open and start a program, the new program's threads will be added to the All Programs/Threads pane.

RELATED CONCEPTS
The IDE debugger

RELATED TASKS
Debugging during the development cycle
Suspending, resuming, and terminating threads
Setting breakpoints in source code
Inspecting and modifying variable values
Stepping through methods
Evaluating expressions in the debugger
Generating the class trace

Suspending, resuming, and terminating threads (IDE debugger)

When one or more programs are running, the Debug page in the Debugger browser shows all running threads, grouped by program. You can suspend, resume, and terminate the threads as needed.

Suspending a running thread
To examine a thread at any point while it is running, you must suspend it manually. Then you can modify or step through its methods and inspect its variables.

To suspend a thread manually:

1. Open the Debugger browser.
2. Select the thread in the All Programs/Threads pane in the Debug page.
3. Click the **Suspend** button  in the toolbar.

Threads halted because of a breakpoint or uncaught exception are suspended automatically.

Once a thread is suspended, expand it in the All Programs/Threads pane to view its method stack. Select a method to inspect its visible variables or work with its code. In the source pane, you can see how the current value of a variable is being calculated by holding the mouse pointer (I-bar) over the variable in the source. After about a second, a pop-up dialog will appear, showing the information.

Resuming running a thread
When a thread has been suspended, either by the manual method above, or automatically when the Debugger opens for a break point or uncaught exception, you can resume running the suspended thread. It will run until suspended manually or automatically, or until the thread terminates.

To resume running a thread:

1. Select the suspended thread.
2. Click the **Resume** button  on the toolbar.

The running is resumed at the point it left off, unless one of the following things has been done while the thread was suspended:

- You replaced a method, in which case it starts again at the start of the method, maintaining all changes to the state of the object that have been made.
- You stepped through code, in which case it starts again after the last step taken.

The program will continue running until it is suspended again or until it terminates.

Note: You can not resume running after an uncaught exception.

Terminating a thread
When you terminate a thread, it is removed from the Debugger browser and cannot be suspended or resumed. Often, terminating a thread will stop the running of the program. To terminate a thread:

1. Select the thread in the All Programs/Threads pane of the Debug page.
2. Click the **Terminate** button  on the toolbar.

The thread will terminate. To restart the thread, you must restart the program from the beginning.

Alternative methods of suspending, resuming, and terminating
Besides using the toolbar buttons described above, you can select **Suspend**, **Resume**, and **Terminate** menu options from the **Selected** menu or the pop-up menu for a thread. Also, pressing F8 resumes a suspended thread.

RELATED CONCEPTS
The IDE debugger

RELATED TASKS
Debugging during the development cycle
Opening the debugger manually
Setting breakpoints in source code
Inspecting and modifying variable values
Stepping through methods
Modifying code while debugging

Setting breakpoints in source code (IDE debugger)

A breakpoint is a signal to the IDE debugger to suspend a program thread that is running the code that contains the breakpoint. When the IDE is running a program and encounters a breakpoint, it suspends the thread (the program temporarily stops running), and the Debugger browser opens so that you can see the run-time stack for the thread.

Breakpoints can be set on any instruction in source code in the workspace. They can be set at any time, including while code is being debugged; that is, you can add a breakpoint to a method in a suspended thread's stack without being dropped to the top of the method.

To set a breakpoint in source code in the IDE:

1. Go to the Workbench or any browser that shows the source code for the place in the program where you want to suspend the thread.
2. Place the cursor in the line.
3. From the **Edit** menu, select **Breakpoint.**

The Configure Breakpoints dialog opens.Fill in the fields as necessary and click OK. For more information on performing this task, refer to the related information at the end of this file.

A breakpoint symbol is placed in the margin of the Source pane next to the line in which you placed the cursor. If you try to set a breakpoint at an invalid location (for example, a comment line), the breakpoint will be set at the closest valid location. If you try to set a breakpoint in a method in which breakpoints cannot be used, a dialog will inform you that there are no valid locations in the method to set a breakpoint.

Setting and clearing breakpoints by double-clicking
Now that you can see where the breakpoint symbol is placed (in the margin of the Source pane next to the code), you can learn the short-cut to setting and clearing breakpoints.

If you set a breakpoint by using the **Breakpoint** menu option in the **Edit**menu, clear it by double-clicking on the symbol. You can likewise set a breakpoint by double-clicking on the margin next to the desired line.

Example: "Running the Hanoi sample with a breakpoint set"

The breakpoints page in the Debugger browser
The Breakpoints page in the Debugger browser shows a summary of all breakpoints that have been set in the workspace. To open this page, select **Debug > Breakpoints** from the **Window** menu. The Methods pane lists all the methods in the workspace that contain breakpoints, and the source pane shows the source code for the method selected in the Methods pane. From this page, you can set breakpoints in methods that already have a breakpoint, and you can remove breakpoints. You can also disable breakpoints (which means that the breakpoints remain set, but the IDE ignores them when running programs).

Other types of breakpoints
Breakpoints can be modified so that they only suspend the thread under certain circumstances; these are called *conditional breakpoints*. You can configure the breakpoint to perform an action or evaluation, which, if results in a true value, will suspend the thread. Also, you can set breakpoints on methods in external classes (classes that reside on the file system and that are called by programs running in the IDE that have the external classes in their class path). See the related tasks below for information on setting these breakpoints.

RELATED CONCEPTS
The IDE debugger

RELATED TASKS
Debugging during the development cycle
Configuring and setting conditions on breakpoints
Setting breakpoints in external classes
Clearing and disabling breakpoints

Configuring and setting conditions on breakpoints (IDE debugger)

Conditional breakpoints are breakpoints that suspend code and open the debugger only when certain conditions are met. For example, you can set a breakpoint to suspend code only if a variable's value falls in a particular range of values.

To set conditions on a breakpoint, right-click the breakpoint symbol , and select **Modify**from the pop-up menu.

In the dialog box that opens, select the **On expression** checkbox. Then you can select a condition from the drop-down list, or you can type in your own condition. (The drop-down list contains up to ten conditions that you have previously set on breakpoints). If the condition is evaluated to a boolean value of true, then the breakpoint suspends the code and opens the Debugger browser.

Example: Using a conditional breakpoint

Configuring the breakpoint to do something
You can also configure a breakpoint to run a Java statement and *then* return true or false. For example, when the IDE encounters the breakpoint, you can output a message and then evaluate the condition. The message will be generated regardless of whether the condition is true or false.

The text entry field in the dialog box has code assist support; you can type the start of a package or class name and press Ctrl+Spacebar to see a pop-up list of available classes or methods. Select the desired one by continuing to type or by using the arrow keys, and press Enter.

Example: Configuring a breakpoint to print a message

Other conditional breakpoint settings
You can also set a breakpoint to halt execution only when encountered from a particular thread, or only on a particular iteration in a program.

Suppose you have two threads running, and both call a method, getAValue(), on which you've set a breakpoint. When the first thread encounters the breakpoint, go to the Breakpoints page and right click the breakpoint. Select **Modify**. Select the **In selected thread** checkbox and select the desired program and thread.

Likewise, if you want to break on a particular iteration in a loop, select the **On iteration** checkbox and input the desired iteration.

The effect of these three conditions (thread, expression, and iteration) are cumulative. For example, if you have selected a thread 'A' and iteration 3, execution will break on the third iteration of the loop on thread 'A', but will ignore occurrences on other running threads.

RELATED CONCEPTS
The integrated debugger

RELATED TASKS
Setting breakpoints in source code
Clearing and disabling breakpoints

Setting breakpoints in external classes (IDE debugger)

The VisualAge for Java IDE can run programs that dynamically load and run external classes. *External classes* are classes that have not been imported into the workspace, but rather reside in a .class file, Zip file, or JAR file on the file system. The path to the file must be part of the class path for the program.

If you want to debug such a program, you have the option of setting breakpoints on methods in the external classes.

To set a breakpoint on a method in an external class:

1. Select **Debug > External .class file breakpoints** from the **Window** menu, or click the **External Breakpoints** toolbar button  in the Debugger browser.
2. The External Method Breakpoints dialog shows a list of methods available for setting breakpoints. Add methods to the list by clicking **Add**.
3. The Add External Methods dialog looks into .class and archive files and lets you select methods within those files to add to the list of methods available for setting breakpoints. To access methods in a .class file:
   a. Click **Directory**.
   b. Browse through the file system to the directory that contains the .class files in which you want to set breakpoints.

   To access methods in a .class file that has been archived:
   a. Click **Archive**.
   b. Select **Zip Files (*.zip)** or **JARFiles (*.jar)** in the **Files of Type** drop-down list.
   c. Browse to the archive file that contains the .class files in which you want to set breakpoints.

   Now, the dialog lists all the .class files in the selected directory or archive. Select one to see the list of methods available for setting breakpoints.
4. If you want to add one of the listed methods to the list of methods available for setting breakpoints, select its check box.
5. When you have selected all the methods you want, click **OK**. The list of methods in the External Method Breakpoints list now shows the methods you selected.
6. To set a breakpoint on one of these methods, select its check box. The breakpoint is not enabled simply by the method being listed in this box; its check box must be selected for the breakpoint to be set.
7. Click OK to exit the dialog.

Once the breakpoint is set, any thread that calls it will be suspended when the method is entered. External breakpoints cannot be conditional and do not display the breakpoint symbol in the source pane margin.

Putting source code on the debug source path

If the source code for the method is available on the file system, and if the path to the source code is included either in the class path or in the debug source path (set in the Debugger Options), you will be able to step into the method code. The debugger looks for the source code first in the class path, and if it cannot find it there, then in the debug source path.

If the source is not available (not on either the class path or the debug source path), you will only be able to step over the method.

Clearing breakpoints on external methods

To clear a breakpoint from an external method, go to the Breakpoints page in the Debugger browser. Select the external method from which you want to remove the breakpoint. Double click on the breakpoint symbol next to the method declaration in the Source pane.

RELATED CONCEPTS
The IDE debugger

RELATED TASKS
Setting breakpoints in source code
Clearing and disabling breakpoints
Stepping through methods
Setting debugger options

Selecting exceptions for the debugger to catch (IDE debugger)

Usually, if an exception is thrown while a program is running, and the program does not catch it, the IDE Debugger opens and the offending thread is suspended. However, if the program does catch it, the debugger will not open, and the program will continue. Even if the program outputs the stack trace when it catches the exception, you might not be able to determine its origin.

To make debugging easier, the IDE debugger lets you effectively set breakpoints on exceptions, so that any time an exception of a certain type is thrown, the debugger suspends the thread that threw it and opens the Debugger browser. You can then see where the exception is happening.

To select a type of exception to be caught by the debugger:

1. In the IDE debugger, go to the Exceptions page.
2. From the list of available exception types, select the types of exceptions you want to set breakpoints on by selecting their check boxes.
3. Click **OK**.

Now when you run a program that throws an exception (of the type you selected), the thread is suspended and the Debugger browser opens, regardless of whether the program catches the exception.

Changing the display order of the exceptions and selecting groups
You can select how the exception types are listed by clicking buttons in the toolbar: by exception type name  , by package name  , or by hierarchy  .

You can also select or deselect groups of exceptions: select all  , deselect all  , the select the current exception and all that inherit from it  , and deselect the current exception and those that inherit from it  .

Example: “Opening the debugger when an exception is thrown”

RELATED CONCEPTS
The IDE debugger

RELATED TASKS
Setting breakpoints in source code

Clearing and disabling breakpoints (IDE debugger)

Once a breakpoint is set, you can remove it at any time, including while you are debugging the code it is in. If you remove a breakpoint from a method while the thread it is in is suspended, the debugger does not drop to the top of the method.

To see a summary of all breakpoints in the workspace, go to the Breakpoints page of the Debugger browser by selecting **Debug > Breakpoints** from any **Window**menu. Select a method in the list to see its source code and the breakpoints.

Clearing breakpoints
To clear a breakpoint in source code or in an external class method, double-click on its symbol in the margin of the Source pane. You can remove breakpoints from any Source pane (not just the one in the Breakpoints page in the Debugger browser).

However, if you are in the Breakpoints page, you can use the following toolbar buttons to clear breakpoints:

	Clears all breakpoints in the currently selected method; removes the method from the Breakpoints page.
	Clears all breakpoints in the workspace.

Disabling breakpoints
Suppose you want to run a program that has breakpoints set throughout its code, but you do not want the debugger to open during the running. You can disable the breakpoints by clicking the **Enable Breakpoints** toolbar button so that is is in the "up" position, as shown:  . The IDE will ignore all the breakpoints it encounters. (The debugger may still launch if an exception is thrown and not caught.) All debugger symbols in the margin of Source panes will change colors from blue to gray.

To re-enable all the breakpoints in the workspace, click the **Enable Breakpoints** button so that it is in the "down" position, as shown:  .

Clearing breakpoints on caught exceptions
To clear a breakpoint on a caught exception:

1. Select the **Caught Exceptions** page in the Debugger browser.
2. Remove a breakpoint on an exception by disabling its checkbox.
3. To remove breakpoints from all external methods or caught exceptions, click the **Deselect All** button  . To deselect an exception type and its subclasses, click the **Deselect Including Subclasses** button  .

RELATED CONCEPTS
The IDE debugger

RELATED TASKS

Setting breakpoints in source code
Setting breakpoints in external classes
Selecting exceptions for the debugger to catch

Inspecting and modifying variable values (IDE debugger)

When a thread has been suspended, the Debugger browser displays all running methods and the variables visible within them.

Select a method in the All Programs/Threads pane. The Visible Variables pane shows the variables in use. You can change which variables are shown by changing the selections in the **Inspector** menu. Variables that themselves contain fields can be expanded to show the fields by clicking the plus symbol ⊞ in the tree.

When you select a variable in the Visible Variables pane, its current value (at the exact point in the program where it was suspended) is shown in the Value pane. If you select multiple variables, the values for each are shown in the Value pane. To select multiple variables:

- Select one; hold down the Shift key; select another: the two variables, plus all variables between them in the list are selected; or
- Select one; hold down the Ctrl key, select another: the two variables are selected; their values are listed in the order you select them. Continue holding down the Ctrl key and selecting.

Opening an inspector
To closely look at one variable that contains fields, select it in the Visible Variables pane and click the Inspect button  in the toolbar. An Inspector window will open, showing the variable's fields, and their values. This information is the same as the information in the Debugger browser Visible Variables and Values panes, and you can select and view the contents in the same ways.

Modifying variable values while the program is running
The values of variables can be modified while the thread is suspended. To modify a variable's value in the Value pane:

1. Select the variable in the Visible Variables pane .
2. Edit the value shown in the Value pane.
3. Select **Save** from the Value pane's pop-up menu.

Alternatively, you can modify the value right in the source pane. For example, if you have an integer variable called depth that has a current value of 4, and you want to change its value to 6, do the following steps:

1. Anywhere in the source pane, type in `depth=6`.
2. Highlight `depth=6`.
3. Select **Run** from the **Edit** menu. This evaluates the expression and changes the value of the variable.
4. To see that the value has changed, find an occurrence of depth in the Source pane and hold the mouse pointer over it for about a second. A pop-up label will appear with the text "depth=(int) 6".
5. Delete the fragment, `depth=6`.

The change in variable value is immediately available to the running program. When you resume running, the *new* value is used.

Example: Modifying a value while running a program

RELATED CONCEPTS
The IDE debugger

RELATED TASKS
Debugging during the development cycle
Suspending, resuming, and terminating threads
Stepping through methods
Modifying code while debugging
Evaluating expressions in the debugger

Stepping through methods (IDE debugger)

When a running thread is suspended at a certain place in the program, the Source pane in the Debug page indicates the point at which the execution stopped by highlighting the corresponding code. You can move forward through the code, step by step, in a variety of ways.

Toolbar icon	Selected menu option	Description
	Step Into	Steps into the current statement, and if the statement calls a method, it adds the method to the stack and stops execution on the first line of the method. If the method is in an external class, but if source is available on the Debugger class path, then this works as though the method were in the workspace; if the source is not available, the external method will be stepped over. Each time you click Step Into, the debugger steps into each method called, adding and removing each to and from the stack as they are stepped through. If you step into a statement that does not call a method, the effect is the same as stepping over the statement.

	Step Over	Runs the current statement, including all methods called within the statement. Stops before the next statement. If you step over a method that takes a significant amount of time to run, the string "`/* Thread is currently stepping*/`" will be inserted into the Source pane. You may wait till it returns or click the **Resume** button  to stop debugging it.
	Run To Return	Runs the current method up to the return statement, and stops before returning to the statement that called the current method.
	Resume	Runs to next breakpoint, until you manually suspend thread, or to the end of the program.

Note that execution may stop earlier than indicated above, if the debugger encounters a breakpoint or an exception.

RELATED CONCEPTS
The IDE debugger

RELATED TASKS
Debugging during the development cycle
Setting breakpoints in source code
Setting breakpoints in external classes
Suspending, resuming, and terminating threads
Modifying code while debugging
Setting debugger options

Modifying code while debugging (IDE debugger)

When a thread has been suspended, most of the methods on the stack can be edited or replaced by another edition of the method (methods required by the system or those in external classes may not be modified).

To edit a method on the stack:

1. Select it in the All Programs/Threads pane.
2. Edit its source code in the Source pane as required. The Source pane has code-assist; type Ctrl+Spacebar to get help with method and field names.
3. Select **Save** from the Source pane's pop-up menu.

To replace a method with another edition:

1. Select it in the All Programs/Thread pane.

2. Select **Replace With > Previous Edition** or **Replace With > Another Edition** from the method's pop-up menu.
3. If selecting with another edition, select the desired edition.

In either case, when you resume running the program, execution will drop to the beginning of the method; any side effects of running the method before are not undone.

RELATED CONCEPTS
The IDE debugger
Editions and versioning

RELATED TASKS
Replacing editions in the workspace (reloading)
Saving changes to code
Debugging during the development cycle
Inspecting and modifying variable values
Suspending, resuming, and terminating threads
Stepping through methods

Evaluating expressions in the IDE debugger

When a thread is suspended in the debugger, you can view the source code for all methods in the thread, and you can evaluate any expression in the source code to see what its value is, given the current values of visible variables in the program.

Using an Inspector window
To evaluate an expression in the debugger and display the results in an Inspector window:

1. In the Debug page of the Debugger browser, suspend a thread in a running program.
2. In the All Programs/Threads pane, select a method in the suspended thread. Its source code will be shown in the Source pane of the browser.
3. Select an expression that will evaluate to some value.
4. Click the **Inspect** button  , or select **Inspect** from the selected code's pop-up menu.

An Inspector window will open to show the value of the expression, given the current values of the program's variables.

Displaying the value in-line
To evaluate an expression and display the result in the text of the Source pane:

1. In the Debug page of the Debugger browser, suspend a thread in a running program.
2. In the All Programs/Threads pane, select a method in the suspended thread. Its source code will be shown in the Source pane of the browser.
3. Select an expression that will evaluate to some value.
4. Select **Display** from the selected text's pop-up menu.

The value of the expression, along with the value's type, will be output as selected text in the Source pane. Press the Delete key to remove the highlighted text.

Note: This technique can be used to evaluate expressions in the Scrapbook as well. In the Scrapbook, the toolbar button  displays the resulting value in-line.

Evaluating expressions is useful, for example, for debugging if-statement and loop conditions that are producing unexpected results.

Example: Evaluating a condition in the debugger

Evaluating an expression in the Evaluation window
Similarly to evaluating and expression in-line, you can copy an expression into the debugger's Evaluation window and evaluate it there. This way, messages (such as an expression's value) are inserted in the Evaluation window instead of the source code pane.

To open the Evaluation window, click the Evaluation Area toolbar button  .

Copy in the expression from the debugger source pane to the Evaluation window. Select it and right-click. From the pop-up menu, select one of the following options:

- Run (to run the selected code)
- Display (to display the results of running the selected code in the Evaluation window)
- Inspect (to open an Inspector on the results of running the selected code)

RELATED CONCEPTS
The IDE debugger

RELATED TASKS
Debugging during the development cycle
Inspecting and modifying variable values
Suspending, resuming, and terminating threads
Modifying code while debugging

Generating the class trace (IDE debugger)

The debugger will generate a trace of class loading and initialization, if you select the Class Trace option. The class trace is useful for determining which classes your program uses, and can help in debugging.

To turn the trace on:

1. Select **Options** from the **Window** menu.
2. Select the Debugging page.
3. Select the **Trace class initialization for running programs** checkbox.
4. Click **OK**.

Note: When this option is enabled, some processing time is required to compute and store the trace. As a result, the program may run significantly more slowly when this option is enabled.

To see the trace:

1. Open the Debugger browser by selecting **Debug > Debugger** from the **Window** menu.
2. In the Workbench or a browser, start a program running.
3. While the program is running, select the program (not a thread) in the All Programs/Threads pane.
4. The trace will be shown in the Source pane. Copy the contents to the clipboard to save them.

As soon as the program terminates, the trace will disappear from the Source pane. If your program runs quickly, you may want to add a breakpoint near the end of the program's code so that the program will stay in the debugger's list of active programs long enough for you to view the trace.

Example: "Generate the class trace for Hanoi"

RELATED CONCEPTS
The IDE debugger

RELATED TASKS
Debugging during the development cycle
Setting breakpoints in source code
Setting debugger options

Setting debugger options (IDE debugger)

Debugger options are set in the IDE Options dialog. To open the dialog, select **Options** in the **Window** menu. Expand the **Coding** branch of the tree view by clicking on the plus symbol  . Select the **Debugging** item.

For details about each setting on the Debugging page of the Options dialog, press F1 when you are in the dialog.

RELATED TASKS
Debugging during the development cycle
Suspending, resuming, and terminating threads
Configuring and setting conditions on breakpoints
Setting breakpoints in external classes
Generating the class trace

Chapter 4. Exporting code

Exporting code

Once you have finished developing, testing, and debugging your project within the IDE, export it to the file system by using the Export SmartGuide. The following options are available to you when you export:

- Export code to a directory in the file system.
- Export code to a JAR file.
- [ENTERPRISE] Export versioned packages and projects to another repository.

You cannot export code if it is not in the repository. As well, you must ensure that the path you are exporting to allows exporting.

Exporting code to a directory
To export code to a directory in the file system:

1. Select the project, package, or type from which you want to export.
2. Select **Export** from the **File** menu.
3. In the Export SmartGuide, select the **Directory** radio button. Click **Next**.
4. Provide a target directory for the export. Exported projects will create subdirectories of the target directory.
5. Select the types of code you want to export (bytecode, source code, or resource files), and click **Details** to select the specific classes, interfaces, or resource files.
6. If you are exporting an applet and you want to generate an HTML file to launch it, select the **.html** option and click Details to select the applets for which you want the HTML launch file.
7. Select other options, if needed, and click **Finish**.

Exporting code to a JAR file
To export code to a JAR file:

1. Select the project, package, or type from which you want to export.
2. Select **Export** from the **File** menu.
3. In the Export SmartGuide, select the **JAR file** radio button. Click **Next**.
4. Provide a target JAR file name for the export. If you specify one that already exists, it will be overwritten.
5. Select the types of code you want to export (bytecode, source code, resource files, or beans), and click **Details** to select the specific classes, interfaces, resource files, or beans.
6. If you are exporting an applet and you want to generate an HTML file to launch it, select the **.html** option and click Details to select the applets for which you want the HTML launch file.
7. Select other options, if needed, and click **Finish**.

RELATED TASKS
Exporting bytecode
Exporting source code
Exporting resource files
Exporting for debugging

© Copyright IBM Corp. 1997, 2000

Exporting to another repository
Importing files from the file system
Including resource files in a project

Exporting bytecode

You can export bytecode for a class *only* if it has no unresolved problems remaining.

To export bytecode for your classes to .class files in the file system:

1. In the Export SmartGuide, after you have selected the target directory or JAR file for the export, select the **.class** check box.
2. Click the **Details** button.
3. Select classes and interfaces from the workspace by enabling their check boxes. Click **OK**.
4. Select other export options, if needed.
5. Click **Finish**.

The bytecode will be exported to .class files in a subdirectory of the target directory. The subdirectory is named after the package containing the class. Periods in the package name become new subdirectories. For example, if you export a class named Class1 in a package named My.Package to a target directory c:\VAJava\ide\export, then the Class1.class file will be in c:\VAJava\ide\export\My\Package.

If you intend to debug the exported code with an external debugger, you must select the **Include debug attributes in .class files** option.

RELATED TASKS

Exporting code
Exporting source code
Exporting resource files
Exporting for debugging
Exporting to another repository
Importing files from the file system
Finding and fixing problems

Exporting source code

You can export source code to a directory in the file system or to a JAR file. To export source code for classes and interfaces:

1. In the Export SmartGuide, after you have selected the target directory or JAR file for the export, select the **.java** check box.
2. Click the **Details** button.
3. Select classes and interfaces from the workspace by enabling their check boxes. Click **OK**.
4. Select other export options, if needed.
5. Click **Finish**.

The source code will be exported to .java files in a subdirectory of the target directory. The subdirectory is named after the package containing the class. Periods in the package name become new subdirectories. For example, if you export a class named Class1 in a package named My.Package to a target directory c:\VAJava\ide\export, then the Class1.java file will be exported to c:\VAJava\ide\export\My\Package.

RELATED TASKS
Exporting code
Exporting bytecode
Exporting resource files
Exporting for debugging
Exporting to another repository
Importing files from the file system

Exporting resource files

When you export code to a directory or to a JAR file, you can optionally also export resource files that the code uses.

To export resource files:

1. In the Export SmartGuide, after you have selected the target directory or JAR file for the export, select the **resources** check box.
2. Click the **Details** button.
3. Select resource files from the project directories by enabling their check boxes. Click **OK**.
4. Select other export options, if needed.
5. Click **Finish**.

The resource files will be copied to the target directory. The bytecode will be exported to .class files in a subdirectory of the target directory. The subdirectory is named after the package containing the class. Periods in the package name become new subdirectories. For example, if you export a class named Class1 in a package named My.Package to a target directory c:\IBMVJava\ide\export, then the Class1.class file will be in c:\IBMVJava\ide\export\My\Package.

RELATED TASKS
Exporting code
Including resource files in a project

Exporting for debugging

After you have developed and debugged a program in the IDE, export it to the target system platform and test it there. If you want to debug the code on the target system, you can use an external debugger to analyze the program.

[ENTERPRISE] VisualAge for Java, Enterprise Edition, comes with a Distributed Debugger to help you debug your Java programs on specific platforms.

In order for an external debugger to work with the code, you must export source code *and* bytecode, and you must select the **Include debug attributes in .class files** option in the Export SmartGuide.

RELATED CONCEPTS
Choosing the right debugger for your program

RELATED TASKS
Exporting code
Exporting bytecode
Exporting source code

Exporting to another repository

Exporting lets you copy solutions and versioned projects or packages to another repository, for example to exchange program elements with another developer. When you export your projects and packages, your project resource files are also exported. If the repository you are exporting to is called “sample.dat”, then your project resources are exported to a folder called “sample.dat.pr”.

[ENTERPRISE] You might export to promote your work to a test repository on another server, to divide a shared repository in two, or to create a standalone repository for working at home. For more information, see the related tasks on changing repositories, dividing a repository, and working at a standalone workstation.

To export projects or packages to another repository:

1. In the Workbench window, select at least one project or package, and then select **Export** from the pop-up menu. The Import SmartGuide will open.
2. Select **Repository** and click **Next** to go to the next page of the SmartGuide.
3. Continue following the instructions in the SmartGuide. After selecting information about the server, repository, projects, and packages, click **Finish**.

The Log window will record which program elements have been exported. Another user of VisualAge for Java Version can now import from the repository into which you have exported your program elements.

[ENTERPRISE] Developers who connect to the target repository can browse the exported program elements in the Repository Explorer window and add the exported program elements to their workspaces.

[ENTERPRISE] Exported program elements retain their ownership settings. Owners, class developers, and package group members who do not yet exist in the target repository’s user list are added to it automatically.

[ENTERPRISE] If you are creating a new repository for development purposes, remember to export the base libraries on which your classes depend. If you forget to include any program elements, you can export them into the same target repository later. There are four base projects:

- IBM Java Implementation
- Java class libraries
- JFC class libraries
- Sun class libraries

RELATED CONCEPTS

Repository
Workspace

RELATED TASKS

Backing up the repository
Searching for a program element in the repository
Importing from another repository
Exporting code
Exporting bytecode to the file system
Exporting source code to the file system

RELATED REFERENCES
Repository files

Deploying code

When you export and deploy an applet or application built with VisualAge for Java, you also need to deploy the runtime for the features with which you created the code, if any, and put the deployed runtime JAR or Zip on your class path.

In general, the JAR files are compressed and are for use when running applets off of a server. The Zips are uncompressed and should be placed on the CLASSPATH of the deployment machine for running applications locally.

Depending on which features you have installed, the following runtime libraries are provided in the eab/runtime35 or runtime30 directory of your install image, unless otherwise indicated

Feature (E)=Enterprise Edition only	Workspace Project Name	Runtimename (one of each .jar and .zip unless otherwise specified)	Supported Runtime Platforms

Java-to-C++ (E)	IBM Enterprise CPP Access Builder Library	`eab/runtime20/ivj2cpp.zip`
		`ivjcpp30.zip`
		WIN lib\ivjtjs20.dll
			WIN

Access Builder for SAP R/3 (E)	IBM Access Builder for SAP R/3 Libraries	`ivjsap35.jar infobus.jar` AIX eab/lib/libivjsij35.so eab/lib/libivjsid35.so WIN eab\bin\ivjsij35.dll, eab\bin\ivjsid35.dll, eab\bin\\librfc32.dll LINUX eab/linux/libivjsij35.so eab/linux/libivjsid35.so SOLARIS eab/solaris/libivjsij35.so, eab/solaris/libivjsid35.so 390 eab/os390/libivjsij35.so, os390/libivjsij35.x, os390/libivjsid35.so, os390/libivjsid35.x, os390/librfc, os390/librfc.x 400 eab/os400/ivjsij35.savf	AIX WIN LINUX SOLARIS 390 400
Persistence Builder (E)	VisualAge Persistence, VisualAge Persistence Common Runtime	`ivjpb35.jar` `ivjpb35wsa35.jar`	AIX WIN

ET/400 (E)	IBM Enterprise Toolkit for AS/400	`as400ut.jar, jt400.jar, jt400mri.zip`
		
		These Toolbox .jar runtime files can also be found in the eab\runtime30 directory
		
		`DATA400.JAR`
		`JT400PROXY.JAR`
		`JT400SERVLET.JAR`
		`JUI400.JAR`
		`UITOOLS.JAR`
		`UTIL400.JAR`
		`X4J400.JAR`
		`JT400ACCESS.ZIP`
		`UTILITIES.ZIP`
			Non visual also:
			 

Domino™ Agent Runner	n/a	`IVJAgentRunner.jar` and `AgentRunner.nsf` (in runtime35\domino\ar\)	
EJB Development Environment (E)	IBM EJB Tools	If you plan to deploy your enterprise beans to a non-WebSphere production server, and your enterprise beans employ either associations or access beans and you must also deploy the following file: `ivjejb35.jar`	

e-Connectors (E)	Common Connector Framework	`ccf` (in IBM Connectors/classes) This is a prerequisite for all e-Connector runtimes.	AIX WIN SOLARIS
	CICS® Connectors	`ctgclient` (in IBM Connectors/CICS/classes) `ctgserver` (in IBM Connectors/CICS/classes)	AIX WIN SOLARIS 390 400
	Encina Connector	`delconn` (in IBM Connectors/classes) `delight11` (in IBM Connectors/classes)	AIX WIN SOLARIS 390 400
	Connector SAP	`ivjsap35.jar infobus.jar` AIX eab/lib/libivjsij35.so eab/lib/libivjsid35.so WIN eab\bin\ivjsij35.dll, eab\bin\ivjsid35.dll, eab\bin\\librfc32.dll LINUX eab/linux/libivjsij35.so eab/linux/libivjsid35.so SOLARIS eab/solaris/libivjsij35.so, eab/solaris/libivjsid35.so 390 eab/os390/libivjsij35.so, os390/libivjsij35.x, os390/libivjsid35.so, os390/libivjsid35.x, os390/librfc, os390/librfc.x 400 eab/os400/ivjsij35.savf	AIX WIN LINUX SOLARIS 390 400
	Connector IMS™ TOC	`imstoc` (in IBM Connectors/classes)	AIX WIN
	Connector MQSeries®	`mqccf` (in IBM Connectors/classes)	AIX WIN SOLARIS
	Connector HOD	`hod40connector, hod40converters` (in IBM Connectors/classes)	AIX WIN SOLARIS 390 400
Enterprise Access Builder for Transactions (E)	IBM Enterprise Access Builder Library	`eablib`	AIX WIN SOLARIS
Java Record Framework (E)	IBM Java Record Library	`recjava`	AIX WIN SOLARIS
SQLJ	SQLJ Runtime Library	`sqlj-runtime`	AIX WIN
Data Access Beans	IBM Data Access Beans	`ivjdab`	AIX WIN

XML Parser for Java	IBM XML Parser for Java	xml4j2015.jar	AIX WIN LINUX SOLARIS 390 400

The runtime directories also contain the unzipped form of most of the runtime libraries so that your CLASSPATH need only contain the IBMVJava/eab/runtime35 and runtime30 directories, as opposed to each runtime Zip. This simplifies setting up for testing applications on your development machine.

Warning: It is important that you test the deployment of your applications as your end-users will see them, specifically without the unzipped runtimes on their CLASSPATH. You can easily do this by removing the IBMVJava/eab/runtime35 directory (and other runtime directories you may have added) from your CLASSPATH variable.

Runtimes for previous releases are provided in the IBMVJava/eab/runtime30 directory.

Deploying an application to debug it
If you want to debug an application that has been deployed, you must install the appropriate debugger back-end on the target machine. See the online help for the VisualAge for Java debugger you are using for more information.

Deploying an applet on the Network Station

Once you have tested your applet in the IDE, you are ready to deploy your program as a applet on the Network Station. Deploying an applet for use by the Network Station is no different than deploying an applet for any other browser. See the related task Exporting and Publishing Code for more information.

RELATED CONCEPTS
VisualAge for Java for the Network Station

RELATED TASKS
Exporting and publishing code
Deploying an application on the Network Station

Deploying an application on the Network Station

Once you have tested your application in the IDE, you are ready to deploy your program as a stand-alone application on the Network Station.

Development considerations
One addition you may need for your developed code is to add System.exit(0) to the windowClosed() method of the application class. Not having an explicit System.exit(0) statement may mean that a Java application may not terminate completely. Since the Network Station only allows a single Java application to run, this may prevent you from running another application without logging off and back on.

Deploy your code
When deploying your client program as a stand-alone application, deploy your

code to a directory on the Network Station Manager machine. This directory must be in the list in the TFTP configuration utility to ensure that the Network Station can get the code as it is required.

Configure the Network Station Manager
The application can be configured as a default for all users or for specific users. It can also be configured to run as an Autostart application or as an application on the menubar. If you configure it as an Autostart application, it will be run whenever the user it is configured for logs on to the Network Station.

In either configuration case, the application or class name is specified in the Network Station Manager administration utility with the full package and class name. Arguments can be supplied and the classpath needs to be specified. The classpath may not accept Zip or JAR files and may also require that the class paths be separated by colons (:). If Zip and JAR files are not supported, unzip the JAR and Zip files that you need from the eab\runtime directory in your VisualAge for Java installation. Ensure that they are in a directory that is in the TFTP list for the Network Manager.

Running the application
If the application is configured as Autostart then it will run as soon as you log on to the Network Station. If it is configured as a menu item then launch it from the menubar. To see any problems with loading classes or output from the program, open the Console window by typing Shift+Alt+Home.

You may run only one application at a time.

RELATED CONCEPTS
VisualAge for Java for the Network Station

RELATED TASKS
Deploying an applet on the Network Station

Chapter 5. IDE hints and tips

Applying IBM Service fixes with Fix Manager

On occasion, IBM Service releases temporary fixes for VisualAge for Java. They are downloadable from the VisualAge for Java website at http://www.software.ibm.com/ad/vajava/ (follow the *Support* link).

Once you have downloaded a fix, store it on your machine according to the instructions in the fix's readme file. After downloading and storing the fix, you must apply it to VisualAge for Java.

To apply a fix you have downloaded:

1. In the Workbench, from the **Workspace** menu, select **Tools > Fix Manager**.

   The **Fixes** list in the Fix Manager window shows all fixes that you have downloaded onto your hard drive. Those fixes that have already been applied have an asterisk (*) beside them. Fixes that have been downloaded, but have not yet been applied do not have the asterisk.
2. Select a fix that you want to apply, and click the >> button. It will be added to the **Fixes to Load** list on the right-hand side. You can add more than one fix to the **Fixes to Load** list.
3. Click OK. The fixes listed in the **Fixes to Load** list will be applied.

When you apply a fix, it remains on your hard drive so that if you have to replace your workspace in the future, you do not lose the downloaded fixes; you simply have to reapply them.

[ENTERPRISE] If you are working in a team environment and want to apply a fix to only one developer's copy of VisualAge for Java, you can do so. It is not necessary to apply the fix for all of the team members. If you wish, you can create a backup copy of the team repository for the team member who is applying the fix to work with.

Troubleshooting in the IDE

If a Class that Uses JFC Will Not Run
If you created a class and wrote code for it that uses JFC, then you must update the class path for it to include the path to the JFC project directory. Otherwise, the class will not run.

To update the class path:

1. Select the class in a browser.
2. From the class's pop-up menu, select **Run**, **Check Class Path**. This will open the class Properties notebook to the Class Path page.
3. Next to the **Project Path** field, click **Compute Now**. The JFC project path will be added.
4. Select the **Save to Repository (as default)** check box.
5. Click **OK**.

© Copyright IBM Corp. 1997, 2000

If a Browser Shows Incorrect Information
If an IDE browser is displaying incorrect or inconsistent information, follow these suggestions:

- **Refresh the browser.** Select **Refresh** from the **Window** menu to refresh the contents of the browser. Under normal circumstances, this action has no effect. However, if there are internal inconsistencies, this action may correct the problem.
- **Close the browser and reopen it.** This action may correct problems that are not corrected by refreshing the browser.
- **Reload the program element.** To reload a program element that is behaving suspiciously.
  1. Select the program element.
  2. Select **Replace With > Another Edition** from its pop-up menu.
  3. From the list, select the edition that is already in the workspace (marked with an asterisk,*).

  Under normal circumstances, these steps have no effect; however, if there are internal inconsistencies, they may correct the problem because they may trigger the IDE to recompile the program element.
- **Delete the project and add it back.** If re-adding the program element does not correct the situation, follow these steps to delete and add back the project that is behaving suspiciously. Do not delete any of the standard class libraries that come with VisualAge for Java.

  1. In the Workbench, select the **Show Edition Names** toolbar button  to see edition names.
  2. Select the project that you want to temporarily delete from the workspace.
  3. Make a note of the edition name of the project.
  4. Select **Delete** from the project's pop-up menu. Deleting the project might add unresolved problems, but you can ignore them.
  5. Click the **Add Projects** toolbar button  .
  6. Choose to add projects from the repository.
  7. Select the project that you deleted.
  8. Select the edition that you noted in Step 3.
  9. Select **Finish** to add the deleted project edition from the repository back to the workspace.

  This step forces the IDE to recompile the project's packages, types, and methods. It also clears any unresolved problems that VisualAge for Java reported when you deleted the project.

If a Java Program Does Not Respond
To stop a program that does not respond:

1. Select the debug button () on any tool bar.
2. In the All Programs/Threads pane of the debugger, select all the threads that were created for the program.
3. Select **Terminate** from the pop-up menu to terminate the threads.

If a Scrapbook Page Remains Busy
If a Scrapbook page remains busy, select **Reset Page** from the page's pop-up men.

RELATED TASKS

Browsing the Workspace
Replacing Editions in the Workspace (Reloading)
Suspending, Resuming, and Terminating Threads
Experimenting with Code Fragments

VisualAge for Java IDE symbols

When you move and hold the pointer over most symbols in the IDE, hover help and the status line present information about them. The following symbols do not display this help.

Program elements

 solution

 project

 package

 class

 interface

 applet

 project resource file

 project resource folder

Inner program elements

 inner default field

 inner protected field

 inner private field

 inner public field

 inner default method

 inner protected method

 inner private method

 inner public method

 inner class

 inner integer

Access modifiers for methods and fields

 default method

 private method

 protected method

 public method

 default field

 private field

 protected field

 public field

Other modifiers for classes, methods, and fields

 abstract

 final

 native

 static

 synchronized

 transient

 volatile

Other symbols

 executable class

 only bytecode, not source code, exists in workspace (imported .class file).

 class or method with unresolved problems

 class with methods that have unresolved problems

 class or method with compiler warnings

 class with methods that have compiler warnings

 code that the Visual Composition Editor generated

 class that the Visual Composition Editor edited

 thread

Scrapbook page symbols

The symbol on each page of the Scrapbook window changes according to the status of the code in the page. The following symbols indicate the different states:

 Page is not associated with a file

 Same page, busy running code

 File for the page has been modified and not saved

 Same page, busy running code

 File for the page has been saved

 Same page, busy running code

Shortcut keys

You can rebind some of these shortcut keys in the Key Bindings page of the Options dialog. To open the dialog, select **Window > Options > General > Key Bindings.**

Code assist

Menu option	Shortcut key
not applicable	Ctrl+Spacebar (or Ctrl+L)

General (most windows and browsers)

Menu option	Shortcut key
Quick Start	F2
Undo	Ctrl+Z
Redo	Ctrl+Y
Cut	Ctrl+X
Copy	Ctrl+C
Paste	Ctrl+V
Select All	Ctrl+A
Format Code	Ctrl+W
Find/Replace	Ctrl+F
Search	F4
Maximize Pane	Ctrl+M
Save *	Ctrl+S
Save Replace *	Ctrl+Shift+S
Breakpoint	Ctrl+B
Method Template	Ctrl+T

* The shortcut keys for **Save** and **Save Replace** can be switched by changing the save option on the Coding page of the Options dialog.

Scrapbook

Menu option	Shortcut key
New Page	Ctrl+N
Open	Ctrl+O
Save	Ctrl+S

Problems pages

Menu option	Shortcut key
Next Problem	Ctrl+N
Previous Problem	Ctrl+P

Search results

Menu option	Shortcut key
Next Match	Ctrl+N
Previous Match	Ctrl+P

Comparison results

Menu option	Shortcut key
Next Difference	Ctrl+N
Previous Difference	Ctrl+P

Bean Info page

Menu option	Shortcut key
Revert	Ctrl+R
Save	Ctrl+S

Visual Composition Editor

Menu option	Shortcut key
Save Bean	Ctrl+S

Debugger

Menu option	Shortcut key
Run	Ctrl+E
Revert to Saved	Ctrl+R
Display	Ctrl+D
Inspect	Ctrl+Q
Step Into	F5
Step Over	F6
Run to Return	F7
Resume	F8
Run to Cursor	F9

Inspectors

Menu option	Shortcut key

Run	Ctrl+E
Revert to Saved	Ctrl+R
Display	Ctrl+D
Inspect	Ctrl+Q

Console and log windows

Menu option	Shortcut key
Clear	Ctrl+R

Repository files

Repository file names
In the file system, the name of the repository provided with VisualAge for Java is ivj.dat. It is recommended that you use .dat as the file extension for any new repositories that you create. To ensure that new repositories can be used on any server operating system, adhere to the 8.3 file-naming convention.

NetWare EMSRV for NetWare supports long filenames only on NetWare where the volume in question has the LONG or OS/2 namespace added. Long filename support is required for the stored resource management feature used by VisualAge for Java 3.5.

WIN When using EMSRV for Windows NT® or 2000, long filenames may be created and viewed on FAT, FAT32, and NTFS volumes.

OS/2 When using EMSRV for OS/2, long filenames may be created and viewed on HPFS volumes. Long filename support is required for the stored resource management feature used by VisualAge for Java 3.5.

[ENTERPRISE] Shared repositories must be local to the server where the repository server is running. You can not use EMSRV to manage remote repositories.

[ENTERPRISE] The EMSRV working directory
For ease of use, it is recommended that you store shared repositories in the EMSRV working directory. This allows team members to connect to the shared repositories without providing path information.

WIN OS/2 NetWare By default, the EMSRV working directory is the same directory where the emsrv executable program is installed. You can change the working directory by using the **-W** parameter of the **emsrv** command when you start the repository server.

Repository size
It is not unusual to have repositories that are between 200 and 300 megabytes, or larger.

OS/2 The maximum repository size is 2 gigabytes.

NetWare The maximum repository size is 4 gigabytes.

WIN The maximum repository size is 2 gigabytes for FAT drives, 4 gigabytes for **FAT32 drives,** and 16 gigabytes for NTFS drives.

AIX HP-UX SOLARIS The maximum repository size is 16 gigabytes.

LINUX The maximum repository size is 2 gigabytes.

If your repository is approaching the maximum size, you should create a smaller repository by compacting it.

RELATED CONCEPTS
Repository

RELATED TASKS
Backing up the repository
Purging program elements from the repository
Compacting a repository
Exporting to another repository
Importing from another repository

Applet Viewer

The Applet Viewer displays applets running in the IDE. (Note: This is the applet viewer shipped with the JDK). You can run multiple applets concurrently (in multiple Applet Viewers), and can make run-time changes to them by altering the code and reloading the applet. The Console window will display standard output from the applet; the Console is also where you supply standard input for the applet.

The Applet Viewer's **Applet** menu has the following options:

Option	Description
Restart	Restarts the loaded applet from the beginning.
Reload	Reloads all code and resources associated with the applet.
Stop	Stops the running applet.
Save	Saves the applet as a serialized object to a file.
Start	Starts the applet, starting at the point in the code at which it was stopped.
Clone	Opens another Applet Viewer on the same applet.
Tag	Displays the HTML tags for the applet.
Info	Displays information on the applet and its parameters.
Character Encoding	Shows the type of character encoding that is used for character conversion.
Print	Print the applet

Properties	Displays the Applet Viewer Properties, where you can change the following settings: • Http proxy server • Http proxy port • Network Access • Class Access • Allow Unsigned Applets
Close	Closes the applet viewer and reclaims resources.
Quit	Closes the applet viewer.

For more information about the Applet Viewer, please see the JDK documentation on the world wide web.

RELATED TASKS
Running an applet from the IDE
Making run-time changes to an applet

Code assist

Source panes, SmartGuides, and some other dialogs and browsers (for example, the Configure Breakpoints dialog) contain *code assist*, a tool to help you find the classes, methods, and fields you are looking for without having to refer to class library reference information. Code assist is accessed by typing Ctrl+Spacebar.

Code assist is able to derive potential completion at the insertion point, providing that up to this point the source is compilable. It will provide contextual inference, including visible variables in regard to their scoping rules. Possible completions are visible methods, fields, local variables, and types. Note that the latter will automatically be qualified if required (for example, if they are not part of the same package or are imported). In addition, code assist will automatically restrain type completion to exception types in throws clauses or catch blocks.

When you type Ctrl+Spacebar, classes, methods, parameters, and types that could be inserted in the code at the cursor are shown in a pop-up list, from which you can select one. Code assist performs a visibility check and classes, methods, and fields that are not visible are not displayed. If the code assist mechanism cannot find a member that fits the current location of the cursor, the information line at the bottom of the pane will indicate that no code assist is available for the current context.

Code assist for types
To insert the name of a class or interface in your code, enter the first one or more letters of the type name, and then type Ctrl+Spacebar. A pop-up list appears, containing types that start with what you have entered. Enter more letters to narrow down the list. Select an item to insert it into your code at the cursor. If the type needs to be qualified, the qualification is also automatically inserted.

Example:
Create a test project and package. In the test package, create a class called AssistTest. In the AssistTest class, create a method called assistMethod.

Suppose you want to declare a local Integer variable, `i`. In the body of the assistMethod source, type the following letters:

```
In
```

Type Ctrl+Spacebar. The list of available types is long. To find "Integer," enter the letters "te". Now, "Integer" will be near the top of the list. Select it using the arrow keys and Enter.

Now, finish the declaration, so that the method looks like this:

```
public void assistMethod() {
 Integer i;
}
```

Save the method by typing Ctrl+S. You will use this test method in the next example.

Code assist for methods and fields

Code assist will also list the methods and fields available for an object or class. Enter *objectName.*, and optionally one or more letters from the start of the method or field name, and then type Ctrl+Spacebar. The list of methods and fields for the object will pop-up. Select one to insert it in the code.

Example:

In the assistMethod method you created in the previous example, below the line that declares `i`, enter the following code (the period is important):

```
i = Integer.
```

Type Ctrl+Spacebar. A list of methods and fields in Integer will pop up. Enter the letters "val", until you find "valueOf(String) Integer". The parameter types (in this case "String") and return type ("Integer") are shown.

Select "valueOf(String) Integer", and it will be inserted into your code. Enter a string such as `"35"` between the parentheses and end the line with a semi-colon. The method source will now look like this:

```
public void assistMethod() {
 Integer i;
 i = Integer.valueOf("35");
}
```

If you request code assist for a method or field from a class that requires qualification, the class must be qualified before you type Ctrl+Spacebar. Otherwise, no code assist will be available. Generally, the code that appears before the cursor must be compilable before you request code assist.

Example:

Suppose java.util.* is not in your class' import statement. This means that the class ResourceBundle must be qualified when you use it in your class. If you type the following code, and then type Ctrl+Spacebar to get the list of methods available, no list will be available:

```
public String newMethod ( ) {
ResourceBundle a = ResourceBundle.
      // place cursor after period
      // and type Ctrl+Spacebar
```

However, if you type the following code, where the class qualification is provided, code assist is available:

```
public String newMethod ( ) {
ResourceBundle a = java.util.ResourceBundle.
      // place cursor after period
      // and type Ctrl+Spacebar
```

An easier way to produce a qualified name in this case (assuming you do not want to add this class or package to the import list) is to place the cursor *before* the period and type Ctrl+Spacebar. Select the class name from the list and it will be fully qualified for you automatically. Then type the period and Ctrl+Spacebar. The list of methods in ResourceBundle will now pop-up.

Code assist for method parameters
Code assist includes pop-up help for method parameters. For example when you select "valueOf(String) Integer" from the pop-up list in an example, above, the following text is inserted at the cursor:

```
valueOf()
```

The cursor is automatically placed between the parentheses, and the name of the parameter (if source is available), and the pop-up label "String" appear to let you know what type of parameter to add.

Keyword completions and macros
You can create your own code assist for Java keywords and any macro you want to insert in Java source code. For information, see Defining code assist macros.

Code assist options
You can set the options for code assist in the Options dialog to automatically insert a completion where there is only one completion listed.You can also choose which key(s) (return, tab, or space) you would like to use to select an item in the completion list . The Options dialog for code assist is available at **Window > Options > Coding > Code Assist**.

Important to note:

- Code assist is case insensitive, except for the first letter of a Class name.
- Code assist will not provide assistance on empty statements or code declaring inner classes.
- Limited code assist is available in a class definition.
- If typing Ctrl+Spacebar does not launch code assist on your system, try using Ctrl+L.
- You can change the shortcut key for code assist on the Key Bindings page of the Options dialog.

Important files to back up

As a VisualAge for Java developer or administrator, you should back up two files on a regular basis:

- The source code repository (page 122)
- Resource files used by your applications (page 122)

You may wish to back up two additional files to avoid having to manually reconstruct them after a system failure:

- The workspace (page 122)

- The VisualAge for Java initialization file (page 122)

Source code repository
The repository file, ivj.dat, is the most important file to back up. It contains all of the source code that you have developed, except for editions that are purged prior to compacting the repository. It also contains Visual Composition Editor information and the bytecode for .class files imported from the file system.

[ENTERPRISE] There may be multiple repositories (.dat files) with different names on the server. Clients may have their own local repositories as well. Make sure all of these are being backed up regularly.

For links to more information about backing up repositories, see the list of related topics at the end of this document.

Resource files
If your application uses resource files, such as audio clips or image files, they should be backed up at the same time as the repository. See the list of related topics at the end of this document for links to more information about resource files and directories.

Workspace
The workspace file, ide.icx, contains the bytecode for the specific editions that you have added from the repository. The workspace file is normally saved when you exit the IDE. You can also save it from the **File** pull-down menu in any VisualAge for Java window.

If the ide.icx file is corrupted or lost, you do not lose source code that you have saved, as it is always stored in the repository. However, you would lose the following information:

- A record of which specific editions of which program elements are in your workspace
- IDE options that you have set
- Breakpoints that you have added to methods
- Contents of the Scrapbook and Console windows
- Bytecode

[ENTERPRISE] Each client has its own ide.icx file, which should be backed up at the same time as the client's ide.ini file.

Initialization file
[ENTERPRISE] The ide.ini file contains information about the server and repository to which you were connected the last time that you exited the IDE or saved the workspace. You should back up ide.ini at the same time as ide.icx.

RELATED CONCEPTS
Workspace
Repository

RELATED TASKS
Saving the workspace
Backing up the repository
Including resource files in a project

Purging program elements from the repository
Recovering the workspace
Reinstalling the workspace

RELATED REFERENCES

Repository files

Notices

Note to U.S. Government Users Restricted Rights — Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:
IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR CONDITIONS OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

© Copyright IBM Corp. 1997, 2000

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Lab Director
IBM Canada Ltd.
1150 Eglinton Avenue East
Toronto, Ontario M3C 1H7
Canada

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples may include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

(c) (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. (c) Copyright IBM Corp. 1997, 2000. All rights reserved.

Programming interface information

Programming interface information is intended to help you create application software using this program.

General-use programming interfaces allow the customer to write application software that obtain the services of this program's tools.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

Warning: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

© Copyright IBM Corp. 1997, 2000

Trademarks and service marks

The following terms are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

- AIX
- AS/400
- DB2
- CICS
- CICS/ESA
- IBM
- IMS
- Language Environment
- MQSeries
- Network Station
- OS/2
- OS/390
- OS/400
- RS/6000
- S/390
- VisualAge
- VTAM
- WebSphere

Lotus, Lotus Notes and Domino are trademarks or registered trademarks of Lotus Development Corporation in the United States, or other countries, or both.

Tivoli Enterprise Console and Tivoli Module Designer are trademarks of Tivoli Systems Inc. in the United States, or other countries, or both.

Encina and DCE Encina Lightweight Client are trademarks of Transarc Corporation in the United States, or other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

ActiveX, Microsoft, SourceSafe, Visual C++, Visual SourceSafe, Windows, Windows NT, Win32, Win32s and the Windows logo are trademarks or registered trademarks of Microsoft Corporation in the United States, or other countries, or both.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Intel and Pentium are trademarks of Intel Corporation in the United States, or other countries, or both.

Other company, product, and service names, which may be denoted by a double asterisk(**), may be trademarks or service marks of others.

© Copyright IBM Corp. 1997, 2000