Allows you to specify the run options for executable targets.

Specifies the command line arguments for the executable target. When the target is Run, PowerJ/Power++ will pass the arguments you specify here to the program.

Allows you to stop the program either at startup or before any code is executed.

Causes this target to execute without any special breakpoints. All of the targets usual breakpoints, which are specified in the Breakpoints window, will remain active.

Causes this target to break at the first line of executable code. This allows you to debug the code that is generated by PowerJ/Power++.

Causes this target to break before any of the application's own code is executed. This allows you to debug the startup sequence.

Runs the program on the computer where it is being developed. You may also debug an application on another (remote) computer.

Runs the program on a remote machine. Debugging is done on the local (this) machine.

Specifies the name of the remote computer. The name can be a dotted decimal IP address (for example, 123.45.6.78) or it can be a symbolic machine name (for example, iron.network.com).

For symbolic names, the @ symbol is replaced by a period.

Forces PowerJ/Power++ to connect to the socket given below on the remote machine.

Allows you to specify whether the application is run on this computer or a remote computer connected by a network.

This page lets you debug a Java class on a remote computer. The remote computer must already be running a Sun Virtual Machine, and you must supply the address of the remote computer and the Agent password for the Virtual Machine.

Specifies the name of the remote computer. The name can be a dotted decimal IP address (for example, 123.45.6.78) or it can be a symbolic machine name (for example, iron.network.com).

You must supply a the Agent password for the Sun VM running on the remote computer. The Sun VM prints the Agent password to the console when you start it on the remote computer. You can enter either the password or the path to a file that contains console output from the VM.

Use this option to type in the Agent password from the Sun VM running on the remote computer if you read the password directly from the console.

Use this option if the Sun VM console output on the remote computer is redirected to a file. Specify the path to the file that contains the console output, including the Agent password.

Allows you to specify how your WebApplication target should be run. Since a WebApplication target consists of several different types of files, there are different ways that your target can be run: By running a web server, a web browser, or just by copying the web files to a specified site.

Causes PowerJ/Power++ to copy the web files to a specified site when the target is run. No executable program, such as a web browser, will be run.

Does not copy the files when the Web application is run.

Copies web files to the location you specify, then launches a web browser to allow you to navigate a web site.

You can specify where the files will be copied (published) on the **Publish** page. To specify which web browser will be launched, click **Configure**.

Allows you to choose the web browser that will be launched when the application is run.

Specifies the web browser that will be launched when the WebApplication is run. To change the web browser, click **Configure**.

Allows you to specify a location where web files will be copied when this WebApplication is run. Copying web files to a web site is called publishing because they are used for debugging or production.

Allows you to specify the name or ID of the running process to which PowerJ/Power++ will connect in order to debug this WebApplication. If you leave this field empty, you can select a running process from a list when you click Run.

Allows you to select a process from a list of running processes.

Causes PowerJ/Power++ to restart the NT service representing your web server when the WebApplication is run.

...

Allows you to specify the web browser that will be launched when this WebApplication is run.
When you run a WebApplication, PowerJ/Power++ publishes (copies) the web files to a location you specify, then starts a web browser. You can use the web browser to navigate through the web pages and debug your application.

Launches the Microsoft Internet Explorer when this WebApplication is run.
When you run a WebApplication, PowerJ/Power++ publishes (copies) the web files to a location you specify, then starts a web browser. You can use the web browser to navigate through the web pages and debug your application.

Launches the Netscape Navigator when this WebApplication is run.
When you run a WebApplication, PowerJ/Power++ publishes (copies) the web files to a location you specify, then starts a web browser. You can use the web browser to navigate through the web pages and debug your application.

Launches a web browser that you specify when this WebApplication is run.

Specifies the name and location of the web browser that you want to use to test your WebApplication. Enter the name of the browser, including its full path. To select a web browser from disk, click **Browse**.

Allows you to select the browser that will be run to test your WebApplication.

Specifies the URL of the page that the browser will display when this WebApplication is run.
URL: Universal Resource Location. You can specify a file name such as "c:\webtest\main.html" or a web address such as "www.ngo.com/dogood.html".

Allows you to locate the URL on disk.

Lists the currently running processes and allows you to select the process that uses your DLL. When you "run" this target, PowerJ/Power++ will attach to the process you specify. You will be able to debug your DLL when it is used by the process.

Specifies that the process that uses your DLL is an NT service that needs to be restarted. When you "run" your DLL, PowerJ/Power++ will automatically start the service that you specify.

Specifies the name of the service that needs to be started when your target is run. This name should be the same as the name that would appear in the list of running processes if you left the field blank.

Specifies any parameters needed to set up the process.

Specifies the name of the application that will be launched when you "run" your DLL. You can use this application to test your DLL, since a DLL can not be run on its own.
You can specify the program's path and name, and any command line parameters that are necessary.

Allows you to select an application from disk.

Allows you to specify what will happen when you run this target.
Since this target is a Web Server or a DLL and cannot be executed, you can choose to run an executable file. If you can choose to launch another program or attach to a running process. PowerJ/Power++ will allow you to debug your DLL as it is used by the program or process.

Disallows you from running this target under the debugger's control. If you want to debug this DLL, you should not chose this option.

Specifies the program that will be started when you run this DLL. To change the program that will be run, click **Configure**.

Specifies the process that will be used to debug this DLL. PowerJ/Power++ attaches to the process that you specify, allowing you to debug your DLL as it is used by the process. To change the process that will be used, click **Configure**.

Allows you to specify the location to which the DLL will be copied before being run. When you run the DLL, PowerJ/Power++ will copy the DLL to the location you specify, then start the program or attach to the process that that you specify.
A DLL must be accessible by programs that use it. Usually this means that the DLL must be in the path or the Windows system directory.

Leaves the DLL in the target folder when you run the target.
A DLL must be accessible by programs that use it. Usually this means that the DLL must be in the path or the Windows system directory.

Copies the DLL to the location you specify before running the target. When you run the DLL, PowerJ/Power++ will copy the DLL to the location you specify, then start the program or attach to the process that that you specify.
A DLL must be accessible by programs that use it. Usually this means that the DLL must be in the path or the Windows system directory.

Specifies the location where the DLL will be copied when the target is run. When you run the DLL, PowerJ/Power++ will copy the DLL to the location you specify, then start the program or attach to the process that that you specify.
A DLL must be accessible by programs that use it. Usually this means that the DLL must be in the path or the Windows system directory.

Allows you to choose a location from disk. When you run the DLL, PowerJ/Power++ will copy the DLL to the location you specify, then start the program or attach to the process that that you specify.
A DLL must be accessible by programs that use it. Usually this means that the DLL must be in the path or the Windows system directory.

Does nothing when the server component is run. The other options on this page allow you to run or attach to another program in order to debug your ActiveX server component, because server components can not be executed on their own.

Starts the program that you specify when you run the server component. You should specify a program that uses the server component so that you can debug the server component. To select a program from disk, click **Configure**.
This option allows you to debug your ActiveX server component; server components can not be executed on their own.

Allows you to select a program from disk that will be started with you run the server component. You should specify a program that uses the server component so that you can debug the server component.

Allows you to specify where the server component DLL will be copied when you run the target. The default location is the Components folder in the main PowerJ/Power++ folder.
Copying the DLL to a central location avoids difficulties caused by registering and unregistering ActiveX DLLs. For example, if you delete a project and its registered DLL, the registry entries for the DLL remain.

Allows you to specify where the server component DLL will be copied when you run the target. The default location is the Components folder in the main PowerJ/Power++ folder. To change the location, click **Browse**.
Copying the DLL to a central location avoids difficulties caused by registering and unregistering ActiveX DLLs. For example, if you delete a project and its registered DLL, the registry entries for the DLL remain.

Copies the file to a specified location when you run the target..
Copying the DLL to a central location avoids difficulties caused by registering and unregistering ActiveX DLLs. For example, if you delete a project and its registered DLL, the registry entries for the DLL remain.

Leaves the server component DLL in the target folder when you run the target.
Copying the DLL to a central location avoids difficulties caused by registering and unregistering ActiveX DLLs. For example, if you delete a project and its registered DLL, the registry entries for the DLL remain.

Automatically registers this component when you run the target. It is highly recommended that you leave this option turned on. When you register an ActiveX server component, it becomes available to other applications, including PowerJ/Power++.

Automatically integrates the component with PowerJ/Power++ when its methods are changed. Only applies to components that have already been added to the component palette. The component is only reinstalled when you run the target.
Turning this option on helps to ensure that the PowerJ/Power++ design environment and Reference Card are synchronized.
Note: You must run the server component target to reinstall the component; simply building the component will not suffice.

Closes this dialog without saving changes.

Closes this dialog and saves your changes.

PowerJ/Power++ needs an import library for DLLs. You should use an existing import library if you have one. Otherwise, you can have PowerJ/Power++ generate the import library.
If your DLL was created with PowerJ/Power++ or Watcom C/C++, you do not need an import library.

Generates an import library for the DLL. You should use an existing import library if you have one.

Uses an existing import library. You should use an existing import library if you have one.

Closes this dialog.

Specifies the name and location of the import library for this DLL.

Allows you to locate the import library on disk.

Allows you to choose special breakpoints, or places where execution will pause for debugging. This setting does not affect other breakpoints you may have set.

Disables the debugging facilities of PowerJ when you run the application. You can use this, for example, to try your production code.

Enables the debugging facilities of PowerJ when you run the application. This allows you to set breakpoints and perform other debugging tasks, but may reduce performance slightly. If this item is disabled, it is because debugging is not supported with the Java interpreter you have chosen; you can choose another Java interpreter and try again, or run without the debugger.

Specifies the class that you wish to debug. This is usually the application's main class.
The debugging facilities are not enabled until the class specified here is loaded. In other words, no breakpoints will be triggered until this class is loaded. Specifying a particular class here may improve startup time, but will disable debugging until the specified class is loaded.

Uses the Microsoft Java Interpreter, or virtual machine (VM), to run your applet or application.

Uses the Sun Java Interpreter, or virtual machine (VM), to run your applet or application.

Specifies the parameters that will be used to start the application.

Specifies the parameters that will be used to start the applet, in the form:
ParamName = ParamValue
Put each parameter on a separate line.

Allows you to specify the command line that will be executed when this application is run. You can specify an external program, such as a specific Java virtual machine. This allows you to execute your application in a particular context, and debug it if debugging is supported.

Allows you to use another browser to display your applet when the target is run. Use this, for example, if you want to test with different versions of Netscape Navigator, or with a browser other than Netscape Navigator or Internet Explorer.

Specifies the name of a web browser that will be used display your applet when the target is run.

Allows you to locate an HTML file to be used to view this applet.

Specifies the HTML file to be used to view this applet.

Allows you to select a web browser to be used to display your applet.

Runs the servlet in a console window.

Uses the built-in Java server to run the servlet.

Unloads forms containing ActiveX controls when targets are run. This is required because some ActiveX controls cause the development environment to lock-up while they are being debugged.

Opens the Object Inspector when you double click a component, or access its properties through the context menu. You can always access the property sheets for a component through the Objects window (Shift+F4).
The Object Inspector lists all properties of an object in a simple two-column list. The Property Sheets, on the other hand, expose the properties by means of a tabbed dialog and some special property editing features.

Opens a property sheet when you double click a component, or access its properties through the context menu. You can always access the Object Inspector for a component by pressing F4 when the component is selected, or by clicking **More** on the property sheets.
The Object Inspector lists all properties of an object in a simple two-column list. The Property Sheets, on the other hand, expose the properties by means of a tabbed dialog and some special property editing features.

Specifies how source files will be displayed for editing.

Causes PowerJ/Power++ to open a separate editor windows for each event or function that you edit. This allows you to edit smaller units of code in each window.

Causes PowerJ/Power++ to use the same window for all members of a class. This allows you to see the entire class and member function definitions in one window.

Displays the code that PowerJ/Power++ generates for a class in the class' editor window. You cannot edit generated code.

Prevents you from changing source code while you are debugging a program.

If you change code while debugging the changes will not take effect until you rebuild the target, and the changes can make it difficult to debug the program further. For example, breakpoints can be misplaced and incorrect lines of source code may be displayed.

Reuses the same editor window to display code at the execution location while you are debugging. In other words, when execution jumps from one source code file to another, the PowerJ/Power++ will open the other source file in the same editor window as the previous file.

Uncheck this option if you want to open a separate editor window for each code file being debugged.

Turning this option on minimizes the disruption caused by jumping to another source file while debugging. Turning this option off allows you to view several source files simultaneously while debugging.

Hides design-time forms when you run and debug an application. Turn this on to reduce screen clutter while you are debugging. If you have trouble navigating through your code while the forms are hidden, try using the Objects window (Shift+F4); it provides access to forms, objects, and individual events.

Lists the available source control systems. Source control systems allow you to share files and projects between several developers without accidentally overwriting another developer's changes.

For more information, click 

Displays source control operations, such as checking in a file, as they are executed.
This option allows you to see what operations are performed. If you uncheck this option PowerJ/Power++ will perform source control operations without displaying individual commands.

Pauses after each source control operation, allowing you to see what operations are being performed. This option is useful if source control operations are performed too quickly for you to recognize them.

Instructs PowerJ/Power++ to build your project while you work.
Building while you work makes the Run menu command respond quicker and decreases system speed slightly.

Specifies the initial target type for new PowerJ/Power++ projects. In other words, if you create a new project, PowerJ/Power++ will automatically create a target of this type in the new project.
You should specify the target type that you most commonly create.

Specifies the whether PowerJ/Power++ will create a new project when it is started.

Causes PowerJ/Power++ to open the most recently used project when it is started.

Causes PowerJ/Power++ to create a new, untitled project when it is started. The untitled project will contain a target of the type that you specify below.
Select this option if you regularly start PowerJ/Power++ with the intent of creating a new project. Below, you should specify the target type that you most commonly create.

Prevents PowerJ/Power++ from creating a new project when it is started.
Select this option if you usually start PowerJ/Power++ in order to work on an existing project.

Allows you to select the target type that will be automatically created when you create a new project. You should select the target type that you create most often. If the target type that you select supports displaying of forms, you can select a default form type in the Form list.

Allows you to select the form type that will be automatically added to the default target for a new project. This option is only available if the default Target type supports the displaying of forms.

Places quotation marks around the filename on the command line. This option is normally required for command lines with spaces in them, and by default it is turned on.

Specifies the file extension for the type of files for which you want to specify a default editor. Type the file extension, or select one of the registered extensions from the list.
The characters that follow the last period in a filename are called the file extension. You can specify how PowerJ/Power++ will open for editing files with a given extension. This allows you to choose what editor to use for a specific type of files.

Describes the file type being configured. This description is displayed by the Windows Explorer.
You can specify how PowerJ/Power++ will open for editing files with a given extension. This allows you to choose what editor to use for a specific type of files.

Specifies how files of the above type are opened by PowerJ/Power++ for editing. When you double click a file of the specified type in the Files view, PowerJ/Power++ will start the editor you specify to allow you to edit the file.
You can have PowerJ/Power++ use its default editor, the editor specified by the system (mimicking the behavior of the Windows Explorer), or an editor program of your choice.

Causes PowerJ/Power++ to open files of the specified type with the default PowerJ/Power++ editor. Unless you specify a different default editor for PowerJ/Power++, the standard text editor window will be used for files of this type.
In other words, when you click on a file of this type in the Files view, PowerJ/Power++ will open it with its default editor, which is normally the PowerJ/Power++ text editor.

Causes PowerJ/Power++ to open files of the specified type with the PowerJ/Power++ text editor. In other words, when you click on a file of this type in the Files view, PowerJ/Power++ will open it with its text editor.

Causes PowerJ/Power++ to open files of the specified type with the default Windows editor for that file type. In other words, when you click on a file of this type in the Files view, PowerJ/Power++ will start the same editor as Windows Explorer.

Causes PowerJ/Power++ to open files of the specified type with the program that you specify. In other words, when you click on a file of this type in the Files view, PowerJ/Power++ will start the editor that you specify here.

Adds or modifies the editor settings for current file type.

Closes this dialog and discards any changes you have made.

Lists the types of files with which you have associated a particular editor program. Associating a file type with an editor will cause PowerJ/Power++ to open files of that type (determined by the file's extension) with the editor you specify. In other words, when you double click a file of the specified type in the Files view, PowerJ/Power++ will start your chosen editor.

Creates a new association between a file type and an editor of your choice. This allows you to use your favorite editor for any type of file.
A file's type is determined by its extension. For example, Bitmap Images have the extension ".bmp".

Removes the selected file type from the list of specially configured file types. Removing a file type from this list will cause files of that type to be opened using the standard PowerJ/Power++ editor.

Allows you to modify the editor associated with the selected file type.

Displays the editor that is associated with the selected file type, and the extension of the selected file type.

Displays the icon associated with the selected file type.

Specifies the file extension for the selected file type.

Specifies the editor that will be used for the selected file type. When you open files of the selected type, PowerJ/Power++ will start the editor specified here to allow you to edit the file.

Specifies the name that represents a specific folder or relative path. When a target is being built, PowerJ/Power++ will replace this **Macro** with the associated **Value** to specify the locations of files such as Java classes, or C++ libraries and header files. This allows you to change the location of these files by changing the value of a macro.

For more information, click 

Specifies the folder name or relative path of a specific folder. When a target is being built, PowerJ/Power++ will replace the **Macro** with this **Value** to specify the locations of files such as Java classes, or C++ libraries and header files. This allows you to change the location of these files by changing the value of a macro.

For more information, click 

Adds or changes the specified macro.

Closes this dialog and discards any changes you have made.

Lists the macros that are used to define the locations of files such as Java classes, or C++ libraries and header files. Use these macros in the Property Sheets of the Targets view. These macros allow you to specify different locations for building projects on different computers.

For more information, click 

Takes the selected macro out of the list.

Creates a new macro. Macros are used to define the locations of files such as Java classes, or C++ libraries and header files. Use these macros in the Property Sheets of the Targets view. These macros allow you to specify different locations for building projects on different computers.

For more information, click 

Allows you to modify the selected macro. Macros are used to define the locations of files such as Java classes, or C++ libraries and header files. Use these macros in the Property Sheets of the Targets view. These macros allow you to specify different locations for building projects on different computers.

For more information, click 

Includes build macros that are defined by PowerJ/Power++ itself in the list above.

Lists the folders where PowerJ/Power++ will search for source files while you are debugging an applet or application. This allows you to debug parts of the application that are not located in the project folder.
Folders are searched from top to bottom.

Allows you to select from the disk a folder containing source files.

Lists folders that contain source files that will not change frequently. If your PowerJ/Power++ projects use source files from other locations that do not frequently change, there is no need for PowerJ/Power++ to check the files to determine whether they need to be compiled.
Specifying that a folder is read-only can dramatically decrease the time it takes to build projects that use files in the read-only folder.

Allows you to select from disk a folder that contains files that are rarely changed.

One or more of the specified options is not valid. Choose Revert to use the previously specified options, or OK to ignore the error.

Discards the options that you specified and resets the options to their previous values.

Ignores the error and applies the current options.
Note: One or more option is invalid.

Uses the default folder for templates.
By default, the templates you create are stored in a folder named Template under the main PowerJ/Power++ folder. In some situations, you may want to specify a different folder. For example, if you have a team of programmers working on a project, you may wish to store templates under a common folder on your local network.

Allows you to specify a location for target, form, and object templates.
By default, the templates you create are stored in a folder named Template under the main PowerJ/Power++ folder. In some situations, you may want to specify a different folder. For example, if you have a team of programmers working on a project, you may wish to store templates under a common folder on your local network.

Specifies the location of target, form, and object templates.

This file is for context help for dynamically generated property sheets.

MTBOX -- edit the items for list boxes and combo boxes.

available on the property sheet for ListBox -- items.

Specifies the initial position in dialog units of the left edge of this control relative to the left edge of the form.

Specifies the initial position in dialog units of the top edge of this control relative to the top edge of the form.

Specifies the control's initial height in dialog units.

Specifies the control's initial width in dialog units.

Specifies how much the height of this control will be changed when the form is resized at run time. Expressed as a percentage of the amount that the height of the form changes.
Resize percentages provide a simple way to ensure that controls on your form appear correctly when the form is resized. For more information, click 

Specifies how much the width of this control will be changed when the form is resized at run time. Expressed as a percentage of the amount that the width of the form changes.
Resize percentages provide a simple way to ensure that controls on your form appear correctly when the form is resized. For more information, click 

Specifies how far this control will be moved to the left when the form is expanded at run time. Expressed as a percentage of the amount the form is resized.
If this control should be moved left when the form is resized, specify 100% for this property.
Resize percentages provide a simple way to ensure that controls on your form appear correctly when the form is resized. For more information, click 

Specifies how far this control will be moved downward when the form is expanded at run time. Expressed as a percentage of the amount the form is resized.
Resize percentages provide a simple way to ensure that controls on your form appear correctly when the form is resized. For more information, click 

Lists the items that will initially appear in the list box or combo box, one item per line. Use this to enter some initial items. You can add more items at runtime using the Add method.

Reads a saved list of items to be placed in the list box or combo box from a text file.

Saves the list of items in a text file.

Specifies the line number where the cursor is and the total number of lines in the list.

MTCOLOR -- change the foreground and background color for a control. available from the Color page of forms.

Displays the foreground and background colors to give you an idea of how the control will appear on the form.

Allows you to select a color for the foreground text from a list of predefined colors. It is often a good idea to select the default color or one of the system colors listed. Doing this will ensure that the color of the control changes with the system colors for users with different settings.

Allows you define a custom color for the control's foreground text.

Allows you to select the background color from a list of predefined colors. It is often a good idea to select the default color or one of the system colors listed. Doing this will ensure that the color of the control changes with the system colors for users with different settings.

Allows you define a custom color for the control's background.

DATACOLS -- the DataColumns dialog -- available for list view and grid in the database page, Browse button.

Lists the columns from the query and allows you to bind to more than one column. Choose a column an press **Add** or double click a column.

List of data columns bound to the control. You can type more columns here; use quotes for unusual characters. Select a column and press delete to remove it. You can re-order columns by cutting and pasting.

Close the dialog without making changes.

Adds the selected column to the list of bound columns.

DDXFORM -- property sheet for form FDX

DDXOBJ -- property sheet for controls with FDX

Specifies the data type of the field in the FDX data structure.

Causes PowerJ/Power++ to generate an enumeration constant for the checked-value for this option button.

Specifies the value of the enumeration constant for this control. The value of the field will be set to this value when this option button is checked.

formmenu.dlg

Gives the form a menu.

Lists the menus available on the form. Select a menu from this list.

formsize.dlg

Allows you to specify the position of the top left corner of the form. The form will be drawn at this position when it is first displayed at runtime.

Centers the form on its parent when it is first displayed at runtime.

Places the form in its design-time position when it is first displayed at runtime.

Specifies the position of the top edge of the form in dialog units.

Specifies the position of the left edge of the form in dialog units.

Specifies the width of the form in dialog units, including borders.

Specifies the height of the form in dialog units, including borders and caption bar.

ActiveXFORM.DLG

Specifies the title of the container, which allows embedded objects to display a title that includes the title of the container when they become active.

Determines whether the server will use the container's window when the object becomes active. Ignored for servers that do not support in-place activation. Uncheck this if you want all objects on this form to use a separate window when they become active.

Determines whether a control in the container can display sizing handles and a hatched border when it is selected. If this property is turned on and **UserMode** is turned off, controls on the form will act like they do at design time.

Places the container in design mode when the application is run. When the container is in design mode, controls will not respond to user input as they would in run mode. You would turn this on, for example, if you were building a container that allows the user to place ActiveX controls and move them around.
If this is turned on (the default), controls will act normally at run time.

Specifies that controls in the container should ignore user input.

Allows ActiveX controls in the container to display sizing handles. You must also turn on the control's DrawSelectHandles property.

Specifies that controls in the container can display a hatched border to indicate that they can be moved. For a control to display a hatched border when it is selected, this property must be turned on and the control's DrawSelectHandles property must be turned on.

Disallows users from dragging ActiveX objects on this form and from dropping ActiveX objects on this form.

Allows ActiveX objects to be placed on the form. This property is only available at design-time because it affects the form's inheritance. If you intend to use any ActiveX objects or controls on this form, you must check this option. If you are not using any ActiveX objects on this form, uncheck this option to save memory and executable size.
If you turn this property on, the form will inherit from the WOleContainer, which gives it the ability to hold ActiveX objects, including ActiveX controls. This property is automatically turned on when you place an ActiveX object on the form, and is never automatically turned off.

query.dlg

Specifies the transaction that this query will use to connect to the database.
For more information, click 

Specifies the raw SQL statement used by this query object. To construct this statement graphically, click **Edit**.

Opens the query editor to allows you to graphically construct the SQL statement for this query object.

Displays a dialog that allows you to see the results of the query and allows you to modify the query.

transact.dlg

Specifies the type of database management system you are using. At present, the only choice is ODBC; this is suitable for any database system with support for 32-bit ODBC 2.0 drivers.

Tests the connection to the database. This may take some time, especially if the database is on another machine.

%%%winmenu.dlg

Assigns a popup menu to this form or control.

Use the right mouse button to click this area to see the popup menu. Selecting an item in the popup menu will take you to the event handler associated with the item.

Allows you to select a popup menu for this form or control from a list of popup menus on the form.

Displays the form's controls in the sequence that they are selected by pressing the **Tab** key. When the user presses tab on the form, the controls are selected in the order they appear in this list. Controls are divided into tab groups by horizontal separators.

Gathers the selected controls into a group.

You should group option buttons together in order to have them work together. When the user selects one option button in a group, other option buttons in the group are automatically turned off (the AutoRadioButton property must be turned on for each option button in the group).

Eliminates all grouping from the form. If you do this, all controls will be considered to be one group.

Changes the tab order of the controls according to their position on the form. Controls will be ordered from top to bottom and left to right.

Creates a tab stop for the current control. As the user presses **Tab** repeatedly, controls with tab stops on the form will receive the focus in the same order that they appear in the list.

Removes the tab stop for the current control. As the user presses **Tab** repeatedly, controls with tab stops on the form will receive the focus in the same order that they appear in the list.

Moves the selected control up by one position in the list. Controls are selected in the order they appear in the list.

Moves the selected control down by one position in the list. Controls are selected in the order they appear in the list.

Closes the Tab Editor and saves any changes you have made.

Saves all the changes you have made without closing the Tab Editor.

Allows you to change the order and grouping of the tab stops for controls on the form.

Tab stops cause the control to be selected when the user presses the **Tab** key. Controls are selected in the order they appear in the list.

Displays the menu's structure with indented entries as sub-menus. The menu editor allows you to modify the properties for menu items, program the click events for the menu items, and reorganize the menu’s structure.

To specify the properties of a menu item, select the item by clicking it. The properties of the selected item are displayed on the property sheets on the right. To program the response to an item's menu event, double click the item. To access other actions for a menu item, use the right mouse button to click the item.

Displays and allows you to modify the properties for the currently selected menu item. To change the properties of a different item, select the other item in the tree.

Demotes the currently selected item, making it a sub-menu of the item above it.

Promotes the currently selected item by one level.

Moves this menu item up by one position in the list.

Moves this menu item down one position in the list.

Specifies the text that will be displayed for this menu item.

To specify an access key, precede the letter with an ampersand (&). For example, E&dit will appear as Edit on the menu.

Places a check mark on the menu item when the program is started. You may check and uncheck menu items at runtime within your program using the **Checked** property.

Checked menu items usually indicate options in your program which are active. When a checked menu item is selected, the option should be turned off and the checkmark removed.

Disallows selection of this menu item at startup. Disabled menu items are grayed.

You should disable a menu item if the menu action is not appropriate for some reason, such as **Paste** with an empty clipboard.

Creates a horizontal line between two menu items.

Use separators to group related items in a menu.

Closes the menu editor and saves any changes you have made.

Adds a sibling menu item below the selected menu item. The new item will be created with default properties, which you can modify.

Adds a menu item as a child of the selected menu item. The new item will be created with default properties, which you can modify.

Deletes the selected menu item and all of its children. You cannot reverse this action.

Updates the selected menu item with the current options.

Shows the structure of the menus. Indented items represent child menus.

You may select an item and change it or edit its event handler.

Specifies the suffix for this menu item. The suffix is added to the end of the menu's name to identify this menu item. For example, if you specify **item_2** here and the menu's name is **menu_1**, then the menu item's name will be **menu_1_item_2**.

Specifies the properties for the currently selected menu item. After you have finished changing the properties for a menu item, click **Change**.

Specifies that the menu item is the default. A menu can contain only one default menu item, which is displayed in bold.

Allows you to specify an accelerator key that will automatically trigger the event handler for this menu item. For example, if you specify CTRL + F5, then pressing CTRL+ F5 will cause the same effect as selecting this menu item.
To specify the accelerator, press the key combination while the cursor is in this field.

Specifies the text displayed in the form's status bar when a menu item is selected.

Specifies that the currently selected item is a horizontal line used to group related menu items. Separators just make the menu easier to read, and cannot be selected like other menu items.

Specifies how the selected menu item is displayed - as text (the default), as a bitmap, or owner drawn.
If you select Bitmap, you must also specify a bitmap to represent this menu item. In addition, you may specify bitmaps that will be drawn when the item's **Checked** property is turned on.
If you select Owner Drawn, your application is responsible for drawing the menu item. To do this your application will need to process MeasureItem and DrawItem events.

Specifies the bitmap that will be displayed for this menu item. To have the bitmap displayed, you must specify Bitmap as the item's **Type**.

Specifies the bitmap that will be displayed beside this menu item when the item's **Checked** property is turned on. If you do not specify a bitmap, a standard checkmark will be displayed when the item is checked.
The checked and unchecked images are displayed to the left of menu items that have other bitmaps associated with them.
Note: It is recommended that you use owner-drawn menus for menus with items with checked and unchecked bitmaps. Owner-drawn menu items avoid problems that occur on systems with nonstandard font sizes.

Specifies the bitmap that will be displayed beside this menu item when the item's **Checked** property is turned off.
The checked and unchecked images are displayed to the left of menu items that have other bitmaps associated with them.
Note: It is recommended that you use owner-drawn menus for menus with items with checked and unchecked bitmaps. Owner-drawn menu items avoid problems that occur on systems with nonstandard font sizes.

Creates a new popup menu for the current menu item. Popup menus contain zero or more menu items, and can appear on the menu bar or as a "fly out" menu item.

Disables the menu or menu item when the form is first loaded. You can enable or disable a menu at run time by setting the **Disabled** property.

Enables the menu or menu item when the form is first loaded. You can enable or disable a menu at run time by setting the **Enabled** property.

Specifies that this menu item can be marked with a check mark to indicate that its corresponding option is turned on. PowerJ uses a checkable menu item, for example, to indicate whether the Java console is displayed. The item is under the Tools menu, and is called Show Console.

Displays the toolbar’s buttons and separators in order. To change the properties of a button, click the button. Use the right mouse button to click a button and access the available actions for that button.

Adds a toolbar item below the currently selected item. The new item will have the properties specified in the property sheets on the right.

Adds a small separator below the currently selected item. You should group related buttons on the toolbar by placing a separator on both sides of the buttons.

One way to use grouped toolbar items is as “option buttons”. When the Checked property is set for one toolbar button, the button will appear to be pushed, and any previously pushed button is released.

Erases the currently selected button or separator. This action cannot be reversed.

Shifts the currently selected button or separator upward by one position. On horizontal toolbars, this will have the effect of shifting the item to the left.

Shifts the currently selected button or separator downward by one position. On horizontal toolbars, this will have the effect of shifting the item to the right.

Allows you to edit the Click event for the currently selected button. The Click event contains the code that will be executed when this button is pressed while the application is running.

Creates a new line of toolbar buttons immediately after the current item.

Gives the button the initial appearance of being pushed.

Sets the Checked property for selected button when the toolbar is loaded. A checked button is used to indicate an option that is turned on.

The Checked property is used for a group of toolbar buttons when only one of the buttons can be pressed at a time. When the Checked property is set for one of the buttons, the button will appear to be pushed and any other button in the group will be released. Buttons used in this way are similar to option buttons.

Grays the button initially, and disallows the user from clicking it.

Specifies that a button in the toolbar object is neither pressed nor released. For example, if the a text selection includes bold and normal fonts, a toolbar button for bold should be indeterminate.

If the user clicks a toolbar button in the indeterminate state, its state should change to the checked state.

Hides the button when the toolbar is created.

Specifies the name of the new or currently selected toolbar button. This name is appended to the name of the toolbar to identify this button.

Specifies the bitmap for the currently selected toolbar button. You can select an icon from the predefined items in the list and add icons to the list from the Resources View.

Allows you to specify the style for the currently selected toolbar item.

Specifies the tooltip for this toolbar item. The tooltip will be displayed when the mouse pointer pauses over the selected button.

Displays a grid of dots on the form design window to make it easier to align and size components visually.

Sets the vertical distance in pixels between columns of grid dots.

Sets the horizontal distance in pixels between columns of grid dots.

Causes component edges to be placed along grid lines. This makes it easier to line components up with each other, but you may have to adjust some component sizes manually in the property sheets if you do not want certain components on grid lines. You can change the grid spacing if necessary.

Allows you to specify the spacing between grid dots.

Many of these topics are the same for C++ and Java, so if you update them, update both cs_comp and jcomp

****

Adds the `transient` keyword to the code that PowerJ generates to declare this object. You should enable this property if you are going to make the form serializable but do not want this object to be serialized.

Specifies the string in the database that will cause the checkbox to be checked.

Important: For fixed-length strings of SQL type CHAR(N) the string length is important. If your JDBC driver conforms to the JDBC specification you will need to ensure that the string you specify is N characters. The specification calls for drivers to return all N characters, so it cannot match unless the value you enter is also N characters. The JDBC-ODBC bridge driver does not conform to that part of the JDBC specification - it only returns the non-blank characters.

Specifies the string that will be sent to the database if the checkbox is unchecked.

Specifies the group to which the checkbox belongs. If you specify a group, only one checkbox in the group can be checked. As a result, the checkbox is displayed as an option button, with a round circle rather than a square.

Specifies the column(s) in the database to be listed in the choice control. This column should contain the choice that the user can select.

Specifies the query object that obtains the list of the possible values for the lookup column.

Changes the current row in the database when the selected item in this control changes. This allows the user to change to a specific row in the database by selecting the item in the list that corresponds to that row.

For list boxes, this property can only be used for a single selection list box.

DataBindAsLookup must be false to use DataTrackRow.

Allows you to bind this control to a database, which means that the control will display values from database. For Choice and ListBox components, this option automatically changes the component to use the Powersoft data bound control rather than the AWT control. The Powersoft control extends the AWT control, so that you will have access to all of the base functionality in addition to database access.

Specifies the column(s) to which the control is bound. Columns can be specified by name or by number, and you can specify any number of columns. The list view displays the columns in the order specified.

Specifies the query object that is the source of the data to be displayed by the list box or choice component. Use this to display data from a database in the list view.

For more information, click 

Specifies one or two columns from the lookup source. The first column specified by DataLookupColumns must have the same type as the column specified by DataColumns; this will usually be a foreign key relationship in the database. If DataLookupColumns only specifies one column, the list box displays all the values found in the given column of the query. If DataLookupColumns specifies two columns, the first column gives the values to be matched against DataColumns and the second column gives the actual names to be displayed.

Click **Browse** to select the columns from a list of available columns.

Enables the user to use the component. If you uncheck this option, the control will be disabled and unavailable for use.

Specifies the component's height. This value can be overridden by the layout manager of the component's container.

Specifies the component's width. This value can be overridden by the layout manager of the component's container.

Makes the component visible at run time. You can change this property in your code with the Visible property.

Allows you to choose whether the controls in this container will be accessible to other classes. It is usually recommended that you make controls private, and provide access to them through member functions and properties that you add to the form. This approach "hides" the implementation of your form from other classes, and can give you the flexibility of changing the implementation without changing other classes.

You can make the controls public, which makes them available to any other class, protected, which only makes them available to classes that extend the form's class, or private, which restricts access to the form's class.

Sets the block increment for this scrollbar. This is the value that will be added (subtracted) when the user hits the block down (up) gadgets.

Sets the maximum value for this Scrollbar.

Sets the minimum value for this Scrollbar.

Makes the Scrollbar vertical (up/down). Uncheck this property to make the Scrollbar horizontal (left/right).

Sets the unit increment for this scrollbar. This is the value that will be added (subtracted) when the user hits the unit down (up) gadgets.

Sets the visible amount of this Scrollbar, which is the range of values represented by the width of the scroll bar's bubble.

Specifies the number of columns in the TextArea or TextField.

Specifies the number of rows in the TextArea.

Sets the echo character for this TextField. This is useful for fields where the user input shouldn't be echoed to the screen, as in the case of a TextField that represents a password.

Specifies the type of scrollbar for the text area.

Specifies the name used to access this object in your program. No two objects on the same form can have the same name.

If you rename this object, its name will be changed anywhere it is used in events, user functions, and in the names of its event handlers.

Specifies that the code generated to create each contained component will not make it smaller than its AWT minimumSize property. Otherwise the generated code ignores the minimumSize property of each component.

Specifies the port number to which this socket is listening

Sets the name of the remote host in preparation of a connect operation on a stream socket.

Sets the port of the remote host in preparation of a connect operation on a stream socket.

Automatically connects the socket when the program is run by calling the Connect method. This is a design-time only property.

Specifies that this object is created by another program and passed to your applet by means of a public method on the applet. You should turn this property on if the ActiveX control resides on a web page. When you do, PowerJ will automatically create a method which the web page calls upon loading. In addition, PowerJ generates an OBJECT tag in the default web page to represent the ActiveX control, and generates the Visual Basic Script required to connect the applet to the ActiveX control at run time.

For more information, click 

Specifies the name of the ActiveX control on the web page.

Centres the image between the left and right and top and bottom edges of the picture box.

Specifies the distance between the bottom of the control and the bottom of the image.

Specifies the distance between the left of the control and the left of the image.

Specifies the distance between the right of the control and the right of the image.

Specifies the distance between the top of the control and the top of the image.

Specifies the location of the image as a URL. The URL can be absolute, relative to the codebase, or relative to the document base, as specified by the Type property.

Specifies whether the image you specify is an absolute URL, relative to the codebase, or relative to the document base. Untrusted applets that specify absolute URLs may violate security restrictions, and relative URLs must be used.

Specifies the requested period of time, measured in milliseconds, between timer events. If the timer is not run at High Priority, the actual interval may be different than the requested interval; high priority timers are more likely to trigger when requested.

Starts the timer when the applet or application starts.

Specifies the number of timer events to be triggered. If 0, events will be triggered until the stop method is called.

Specifies whether the vertical scroll bar will be displayed always, never, or only when the number of rows in the grid exceeds the number of rows that can be displayed.

Specifies whether the horizontal scroll bar will be displayed always, never, or only when the number of rows in the grid exceeds the number of rows that can be displayed.

Specifies a query object to be used when making updates. If this is null, PowerJ automatically creates a second query object to do the updates.

Allows the query object to make changes to the data in the database. Turning this on results in a slight decrease in database access speed, so you should only turn it on if the query needs to update the database.

Causes the query to go into edit mode automatically if the user makes a change in a row. If you turn this off, your code must explicitly put the query into edit mode if you want to modify an existing row.

Automatically opens the query when the query object is created, executes the associated SQL statement, and moves to the first row retrieved.

Specifies which version of JDBC you want to use when accessing the database. This setting must be the same as the setting for the Transaction object associated with the Query. UseJavaSqlPackage is usually used with Java 1.1, and UseJdbcSqlPackage for typical for Java 1.02.

With AutoDetectPackage, PowerJ checks the transaction object associated with this query object and uses the same JDBCPackage property as the transaction object. Note that this means you cannot use AutoDetectPackage if the query object is not associated with a transaction object.

Causes the query object automatically records important actions in the program's debug log; for example, it records when the query is opened. TraceToLog only has an effect in Debug mode; it does nothing in Release mode.

Note that this property controls information written to the PowerJ debug log, not the database's trace log.

Turning this property on in Debug mode can cause a drastic decrease in database access speed. While it is a valuable debugging tool, it should only be used when needed.

Causes the applet or application to automatically connect to the database when the form is created. If it is unchecked, your code must explicitly issue its own instructions to connect to the database.

Allows you to specify which version of JDBC you want to use when accessing the database. UseJavaSqlPackage is usually used for Java 1.1, UseJdbcSqlPackage is usually used for Java 1.02. With AutoDetectPackage, the application itself attempts to determine which package is appropriate.If you use AutoDetectPackage, WebApplication targets and collection targets (ZIP, JAR and CAB) may not be able to determine whether JDBC classes should be included in the target. For more information, click 

Causes the applet to automatically connect to the database when the form is created. If this is unchecked, your code must explicitly issue its own instructions to connect to the database.

Specifies an integer that is used in connection with **DataKeptRows**. If **DataKeptRows** imposes a limit on the number of rows shown in the grid, the program must add new rows to the grid when you move off the top or bottom of the grid contents. If you move within **DataGuardRows** of the beginning or the end of the current grid contents, the grid adds more rows at this end of the list and discards rows at the other end.

Specifies the maximum number of rows to be displayed in the grid at any one time. By limiting the number of rows, you can reduce the amount of memory used to hold the data and the amount of time needed to fetch the data and store it in the list view. If DataKeptRows is zero, the entire rowset is displayed.

Changes the database cursor position when the user changes row in the grid. MultipleSelection must be turned off for this setting to take effect.

Specifies which interface or interfaces this object's class implements. To implement more than one interface, separate the interface names with commas. If your class implements an interface, you must provide an implementation for each method in the interface.

No help available for this topic.

Closes this dialog box and saves any changes you have made.

Closes this dialog box without accepting any changes you have made.

Adds or removes component(s) to or from the system registry. You must register a component before using it in your PowerJ/Power++ application.

For more information on adding and registering custom components, click 

This dialog helps you add or remove component(s) to or from the system registry. You must register a component before using it in your PowerJ/Power++ application.

For more information on adding and registering custom components, click 

Lists the registered custom components and their locations. You can remove one of these components from the registry, or add new ones to this list. You must register a component before using it in your PowerJ/Power++ application.

For more information on adding a custom component, click 

Removes the selected component from the system registry. Do this when you install a newer version of a component, for example.

Allows you to specify the programmatic ID of the control. The programmatic ID is the name that uniquely identifies the component in the system, and is stored in the system registry. For example, the programmatic ID of an Excel worksheet is: Excel.Sheet

Lists the registered automation servers. If a registered automation server does not appear in this list, type its programmatic ID in the text box above.

Displays only automation servers in the above list. Only use this option if the components you are looking for do not have registered type libraries (i.e. if they are not listed when the **Show type libraries** option is set).

Displays only registered type libraries in the above list.

Specifies the page on the component palette where the new component(s) will be placed. Choose an existing page from the list or type the name of a new page.

Specifies that the source files that are generated by the ActiveX component wizard will not be deleted. These source files are not used once the wizard is finished, but you can save them if you want to modify them and regenerate the component(s).

Lists the component libraries that are on your component palette, and allows you to remove and add component libraries.
You can use this dialog to add and remove components, including ActiveX components for Power++ and PowerJ, and JavaBeans components for PowerJ.

Removes the selected component library from the component palette, but does not delete any associated files. When you remove an ActiveX component, the component remains registered with the system.

Allows you to select a component library to be added to the component palette. Use this for components that you have previously removed. For Power++, you use this to add native Power++ components.

Invokes the ActiveX Component Wizard, which allows you to add ActiveX controls to the component palette.

Invokes the Java Component Wizard, which allows you to add JAR and ZIP files, and classes to the component palette.

Specifies the name of the new file. This file may have any extension, but .cpp and .hpp are commonly used for source and header files.

Specifies which of the targets, if any, the file should be used in.

This dialog helps you create a new source or text file.

Adds the symbol and string that you typed to the list of strings for this project.

Changes the selected string and symbol to the string and symbol that you typed.

Empties the text boxes and allows you to enter a new string.

Specifies the symbol that represents this string in your program. You can use any combination of letters, numbers, and underscores in this symbol (no spaces). This is like a variable name in your program; it is useful to name the symbol with a descriptive name of what it represents in your program.

Specifies the string that represents this symbol in your program. Using a string resource in your program allows you to keep all strings in one place, and makes changing strings easier. You can use these strings, for example, as the text in a dialog or message box.

Takes the selected string resource out of the list of strings for this project.

Lists the string resources available in your program.

Stores information about the target's version as a resource in the target file.

Allows you to manage version information. This field specifies the version of this particular target.
The product version field specifies the version of the entire product. This combination allows you to release updates of a specific target, or a completely new release.

Allows you to manage version information. This field specifies the version of the entire product.
The file version field specifies the version of this particular target. This combination allows you to release updates of a specific target, or a completely new release.

Specifies to target browsers that this version of the target is for debugging purposes. This does not affect your compiler options.

Specifies to target browsers that this version of the target is not a release version. This does not affect your compiler options.

Specifies a language for a language/character set pair. These pairs are stored in the resources, and can be used by the system to determine the character set to be used for displaying text.

Specifies a character set for a language/character set pair. These pairs are stored in the resources, and can be used by the system to determine the character set to be used for displaying text.

Specifies the version information for the current target. This information is used by the system when displaying the properties of the target.
To change the value of an item, click the item, enter the new value, and press **Enter**.

Specifies the language and character set used by this target. The system can use this information to display text correctly on computers that use another language.

Allows you to specify a new language/character set pair to be used by the target. If you define a language/character set pair, your application should provide support for users of that language.

This dialog will declare a new method for the current class. Type the prototype of the method, select the method's scope, and click **OK**.

Creates the new user function and closes this dialog box.

Allows you to enter the full prototype of the new user function; enter anything that you would normally put in the declaration of a function, including #ifdef conditions, and so on. You may not define a function here by entering the function body.

Allows you to specify the visibility of the member function.
Private: Only accessible by the same class. Uses `private` keyword.
Default: Accessible to classes in the same package. This uses no keyword and is often called "package visibility".
Protected: Accessible to classes in same package and subclasses in other packages. Uses `protected` keyword.
Public: Accessible to all Java classes. Uses `public` keyword.

Provides a description of the target type.

This wizard guides you through the creation of a new target (a Win32 executable, and a DLL are examples of targets).

Help for this topic has not been written.

Allows you to specify the name of the target. For an Win32 executable target, this is the name that will be given to the executable. The new target and its supporting files will be stored in a folder with the same name as the target.

Allows you to select the drive (network or local) where the target and its supporting files will be stored.

Allows you to specify the name of the new folder where the target and its supporting files will be stored.

Shows the full path of the folder where the target and its supporting files will be stored.

Allows you to specify the name of the target.

Displays a list of the types of targets that can be created. Select one of the target types by clicking its icon.

Allows you to choose whether an windowed or console executable is created.

Creates a standard windowed application.

Controls whether the PowerJ/Power++ design framework will be used for this target. The PowerJ/Power++ framework allows you to design forms visually, use drag and drop programming, and so on. Turn this option off if you want to manually write your application.

This wizard guides you through the process of creating a form for a target.

Allows you to choose the type of form that will be created. Select a form type and click **Next**.

Allows you to specify the name of the file that contains the form's description. This file is maintained by PowerJ/Power++, and must have the wxf extension.

Allows you to specify the name used to access this form in code. This is not the title of the form.

Allows you to choose which target the new form will be associated with.

Specifies the ODBC data source (database) that the form will be connected to.

Specifies the user name used to connect to the database. If you do not specify a name, you will be prompted it and a password when the connection to the database is made.

Specifies the password used to connect to the database. If you do not specify a password, you will be prompted for it when the connection to the database is made.

Specifies additional connection parameters to be added to the connection string. This string is optional, and depends on what, if any, addition information is required to establish a connection with the data source you are using.

Specifies the SQL statement used for the query. This allows you to specify the columns from the database to be used, and the criteria to be used in selecting rows from the database.
Type an SQL statement or click **Edit** to build an SQL statement graphically.

Lists the predefined layouts for controls on your form. A text box and a label are created for each column in the database. PowerJ/Power++ will arrange the text boxes using the layout you select here.

Determines what kind of database form will be created; a master detail form or a single query.
Master detail views have two database queries; a master query and a detail query. The information in the detail query depends on the current row in the master query.
For more information on master detail views, click 

Allows you to specify how the controls in the detail view will be arranged on the form. Select list of predefined layouts. The image above gives you a rough idea of how the controls will be arranged.

Allows you to specify how the controls in the master view will be arranged on the form. Select list of predefined layouts. The image above gives you a rough idea of how the controls will be arranged.

Specifies the SQL statement used for the query in the detail view.
Type an SQL statement or click **Edit** to build an SQL statement graphically.

Specifies the SQL statement used for the query in the master view. In an upcoming Form Wizard page, you can select the columns that will be linked to the detail view.
Type an SQL statement or click **Edit** to build an SQL statement graphically.

Attempts to connect to the database. If the connection is successful when you test it here, it is likely to be successful at run-time. If it fails, you will need to configure the database connection.

Opens the Query editor to allow you to specify database tables and columns to be used for the new form.

Specifies the caption of the group box that will be placed around the controls associated with the detail view.

Creates a data navigator control that allows the user to move the cursor in the detail view and edit the data if the detail view allows editing.

Lists the columns in the master query and allows you to select the column to be linked to a column in the detail view. Select the column to be linked to a column in the detail view, then select the column in the detail view, then click **Add**.
For more information on master detail views, click 

Lists the columns in the detail query and allows you to select the column to be linked to a column in the master view. Select the column to be linked to a column in the detail view, then select the column in the detail view, then click **Add**.
For more information on master detail views, click 

Displays the linked columns. The detail view will display records where the linked columns match the current row in the master view. For more information on master detail views, click 

Links the selected columns in the master and detail views.
The detail view will display records where the linked columns match the current row in the master view.
For more information on master detail views, click 

Removes the link for the selected column pair.
The detail view will display records where the linked columns match the current row in the master view.
For more information on master detail views, click 

Removes all links between columns in the master and detail views.

Allows you to select the target that will be run. Select a target from the list and click **OK**.

Allows you to locate the missing file. PowerJ/Power++ will automatically update the target with the file's new location.

A file reference by the target cannot be found, probably because it has been moved, renamed, or deleted. To locate the file yourself, click **Find**. To remove the file from the target, click **Remove**. To continue loading the target without the missing file, click **Ignore**.

Continues loading the target without the missing file. Click this if the file is missing temporarily (for example, if the file is located on a network drive and the computer is not connected to the network).

Removes the reference to the missing file from the target. Click this if the file is no longer used by the target. If the file's location has changed, click **Find**.

Allows you to specify the targets upon which the current target depends.

Allows you to rearrange the Toolbar by adding, moving, or removing buttons. For example, you could add a button for a tasks that you perform often.

Displays larger buttons on the toolbar and component palette. Uncheck this to display smaller buttons.
Using smaller buttons will make more space available on your screen by reducing the size of the main PowerJ/Power++ window.

Lists the toolbars that can be displayed. To hide a particular toolbar, click its name so that it is unchecked.

Displays a popup tool-tip that describes the function of a button or the name of a component when the mouse cursor pauses over the button on the toolbar or component palette.

Displays the main toolbar, which contains shortcuts for common tasks such as running, creating a new form or project, and so on. You can modify what is displayed on the toolbar by clicking **Customize**.

Allows you to specify which of the JDK toolbars that will be displayed. By default, PowerJ displays the JDK 1.1 toolbar, for example, if you are designing a target that uses JDK version 1.1. This option allows you to override the default.

Causes PowerJ to automatically decide which component palettes need to be shown based on the targets that you have open. PowerJ displays the JDK 1.1 component palette, for example, if you are designing a target that uses JDK version 1.1.

Sets all options on this page to the default values: Clicking this will turn on the following options: Show main toolbar, Show required component palettes only, Large buttons, and Tooltips.

Checks whether your program writes to memory that has not been allocated. If an invalid write is detected, a message is written to the Debug Log.

Checks that each allocated piece of memory has been properly freed by the time the program finishes execution. If not, PowerJ/Power++ creates a summary of objects that were not freed properly in the Debug Log.
Setting this option can help to find memory that has been allocated and not deallocated.

Produces information about each memory allocation operation as it occurs. This option will slow down execution time.

Checks the validity of the area used for dynamic memory allocation every time a new piece of memory is allocated. To produce only a summary of the number of memory allocations and deallocations, click this option so that it is unchecked.
This is one way to detect problems of one object overwriting another in this memory area.

Checks the validity of allocated objects whenever your program invokes their methods, or accesses them in some way. If an invalid object is accessed incorrectly, a message will be sent to the Debug Log and a message box will be displayed.
To use these options, build your targets in DEBUG mode.

Checks whether WString objects are valid whenever you invoke a WString method. A WString would be invalid, for example, if your program writes directly to its internal memory (which should never be done), or deletes it and then invokes one of its methods.
To use this option, build your targets in DEBUG mode.

Checks whether objects are valid whenever your program invokes their methods. An object such as a command button would be invalid, for example, if your program deletes it and then invokes one of its methods.
To use this option, build your targets in DEBUG mode.

Checks the validity of handles used to reference objects like bitmaps and fonts. A handle would be invalid, for example, if it has been deleted or freed.
To use this option, build your targets in DEBUG mode.

Lists the files in your project do not use the default build properties. To apply the default build properties to the files in the list, click **Yes**. To maintain the custom build properties for the files in the list, click **No**.

Maintains custom build properties for the files in the list instead of applying the default build properties.
Files in the list have custom build properties. In other words, when you change the default build properties the build properties of the files in the list do not change. Click **No** to maintain the custom build properties for the files in the list.

Applies the default properties to the files in the list.
Files in the list have custom build properties. In other words, when you change the default build properties the build properties of the files in the list do not change. Click **Yes** to make the properties of the files in the list the same as the default properties.

This dialog allows you to save the current form as a template that you can reuse as a base for other forms. When you saving a form as a template, PowerJ/Power++ adds the form to the list of forms in the form wizard.

Displays the icon that will be used to represent this template in the Form Wizard. To modify the icon, click **Edit**. To select a different icon from disk, click **Browse**.

Specifies the name of the form template. Use a short descriptive name. Type a longer description in the box below.

Describes what this template is and why it was created. This description will be displayed in the Form Wizard when this template is selected.

Invokes the image editor to allow you to modify the icon that represents this form in the Form Wizard.

Allows you to select from disk an icon to represent this template in the Form Wizard.

Specifies the target that will use this class. Pick a target from the list.

Opens a list of installed components and allows you install the missing component.

Cancels the loading of the project. The project will be left unchanged, and you will be able to load it when you have installed the missing component(s).

If you have selected "Don't load the project", this button cancels the loading of the project. The project will be left unchanged, and you will be able to load it when you have installed the missing component(s).
If you have opted to find the component, this button will bring up the list of installed components so that you can install the missing component. If you cannot find or install the missing component, PowerJ/Power++ will display this dialog again.

Deletes all the files in the existing folder, and reuses the folder for the new target.

Prompts you for another folder in which to create the new target. This option does not impact the contents of the existing folder.

Places all the new target files in the directory, without deleting any existing files. It is normally a good idea to create each target in its own directory, because PowerJ manages files for you.
You may want to choose this option, for example, if you want to create an application in a directory where there are already some HTML files.

You are attempting to create a target in a folder that already contains files. Normally each PowerJ/Power++ target occupies its own folder. However, in some circumstances, you may want to create the target in a folder with pre-existing files. Choose one of the options below.

Turns on compression for the JAR file, which speeds up the time to download the file, but makes the JAR incompatible with older browsers and makes it impossible to use the JAR file in the classpath.

Allows you to specify whether the JAR will be compressed. Compressing the file speeds up the time to download the file, but makes the JAR incompatible with older browsers and makes it impossible to use the JAR file in the classpath.

Causes PowerJ to create a default manifest file for the JAR target. The default manifest file will contain a list of classes in the archive, but will not specify other information, such as which classes are JavaBeans components. To specify this type of extra information, you must specify a manifest file, to which PowerJ will add other coded internal information about the JAR file format such as the compression scheme used.

Allows you to provide a manifest file for the JAR. Manifest files can contain two types of information: Internal information about how the classes in the JAR are stored, and extra information about the classes in the JAR, such as which of the classes are JavaBeans components.
The manifest file you specify should contain a list of classes that are JavaBeans components, and any other information you want about the classes in the JAR. PowerJ will add the coded internal information to the JAR, such as the compression scheme.

Allows you to locate a manifest file on disk. Manifest files can contain two types of information: Internal information about how the classes in the JAR are stored, and extra information about the classes in the JAR, such as which of the classes are JavaBeans components.
The manifest file you specify should contain a list of classes that are JavaBeans components, and any other information you want about the classes in the JAR. PowerJ will add the coded internal information to the JAR, such as the compression scheme.

Creates no manifest file. This option is not recommended in most cases. Manifest files can contain two types of information: Internal information about how the classes in the JAR are stored, and extra information about the classes in the JAR, such as which of the classes are JavaBeans components.

Publishes all of files in the target with your applet. Use this option if your target includes HTML files, images, or other types of files that need to be deployed with your applet or application.

Only publishes class files. Use this option if you do not want to publish HTML files, images, or other types of files with your applet or application.

Allows you to specify the location of a ZIP file or other folders containing class files used by this component, but not accessible in its tree.
This option is used only while PowerJ is adding the component to the palette, and not at design time or run time. You will rarely need to add anything to this field.

Adds the package you specified to the list of packages that will not be published with the Web Application.

Deletes the selected package from the list of packages.

Publishes all classes used by the web application, including any Powersoft supporting classes used by your applet. You should choose this option if your users do not have the extra classes installed, since it will allow them to automatically download the classes from the Web Server.

Only publishes the classes in your applet, and does not publish the supporting classes such as the Powersoft classes. You should choose this option if your users have the extra classes installed.

Allows you to specify a package that will not be published with your Web Application. Type the name of the package, and press Add. You only need to exclude packages that are used by your application.

Specifies the packages that will not be published with your Web Application. By default, the Web Application publishes all files that are used by the applet(s) or application(s) in the Web Application. However, you may elect to exclude some packages, for example, if your users already have the classes installed, or if the Web Server already has copies of the necessary classes.

Specifies the name of the file to be added to the palette. If you select a class file, an icon will be added to the palette for that class. Note that the class must be marked as public.
JAR files are Java Archive files that hold JavaBeans components and other files. If there is a manifest file (`manifest.mf`) PowerJ will use it to determine which classes to add to the component palette. Otherwise, all classes except property editors, customizers, and Bean info classes are added to the component palette.
ZIP files contain only class files and no file to determine what classes go on the palette. When you add a ZIP file, PowerJ adds all classes files except property editors, customizers, and Bean info classes to the component palette.

Copies the class files associated with the Bean to the UserClasses folder so that they will be accessible at design and run time. If you are adding a JAR file, this option will unzip the JAR and place its contents under UserClasses.
It is best to turn this option off, since turning it off decreases the amount of disk space required for this component. However, you must quit and restart PowerJ before the class will be usable, because the classpath cannot be changed while the Java VM is loaded.

Places an icon on the palette even for classes that do not have a visual representation. This option is only used for ZIP files because they do not have manifest files to specify what classes will go on the palette. Turn this option off if your ZIP file contains many non-visual classes that are not used at design time.

Specifies where the component will be placed on the component palette. This option allows you to group related components together. You can select an existing page from the list, or create a new one by typing a new name.

Allows you to select the version of the JDK for which this component will be used. Normally you should add components to the component palette that corresponds to the version of the JDK that they use.

Uses PowerJ's own scanning facilities, rather than using introspection. Introspection can be very slow, and leaving this option turned on can decrease the time it takes to add a component without affecting the result.

Allows you to select a class, ZIP, or JAR file from disk.

Shows the large version of the icon that will be used if the component does not specify its own icon.

Shows the small version of the icon that will be used if the component does not specify its own icon.

Shows how the small icon will look when it is displayed in the tab stop editor as an example.

Shows how the large icon will look when it is displayed in the Objects window as an example.

Allows you to select an icon file from disk. The large and the small icon should be stored together in a single `ICO` file.

Invokes the image editor and allows you to edit the component's icons. If you invoke the Image Editor to edit the icons, you see the 16x16 version of the icon first. Use Add Icon Image to a 24x24 icon image to the icon file. To switch between the different sized images, use the **Select Icon Image** entry of the **Edit** menu in the Image Editor.

Uses the Visigenic IDL translator to create Java code from IDL (interface description language).

This option is only enabled if Visigenic idl2java is installed, indicated by the registry key called HKEY_LOCAL_MACHINE\SOFTWARE\Visigenic Software\Visibroker for Java.

Uses the Iona IDL translator to create Java code from IDL (interface description language).

This option is only enabled if Iona translator is installed, indicated by the registry key called HKEY_LOCAL_MACHINE\SOFTWARE\IONA Technologies\OrbixWeb.

Uses the IDL translator that you specify to create Java code from IDL (interface description language).

Allows you to specify the full path of another IDL to Java translator.

Allows you to locate the IDL to Java translator on disk.

Uses the default package name for Java classes created by the IDL translator if no module name is specified in the interface description.

Uses the package name that you specify for Java classes created by the IDL translator if no module name is specified in the interface description.

Puts generated files in the same folder as the project file. Project files have the extension WXJ.

Puts generated files in the same folder as the target file. Target files have the extension WXT. This is the default location.

Allows you to specify a location for the generated files.

Specifies the location where generated files will be left.

Does not check the file out. This will leave the file in a read-only state, and any changes you make will not be saved.

Specifies the file that is checked in to your revision control system (RCS). You can choose to check the file out or leave it checked in. If you do not check the file out, you can make changes and see their effect, but cannot save changes.

Checks the file out. Do this if you want to modify the file and save the changes.

Stops PowerJ from asking you whether to check out individual files, and automatically checks out read-only files when you start to modify them.

Cancels the operation and does not remove any additional components from the palette.

Lists each component that will be removed from the palette if you remove the selected library.

Displays the name of the component library that you have chosen to delete.

Leaves this list of components on the palette.

Removes this list of components from the palette.

Removes this list of components and any further component libraries that you selected to remove. To remove only the currently displayed list of components, click **Yes**; PowerJ will ask you before removing any other libraries.

Specifies the name of the component.

Allows you to locate the include folder on disk.

Specifies the location of the JDK version 1.02 on your machine. This is used to set the classpath when your 1.02 applets and applications are run with the Sun VM.

Allows you to specify where PowerJ should find JDK version 1.02 on disk.

Specifies the location of the JDK version 1.1 on your machine. This is used to set the classpath when your 1.1 applets and applications are run with the Sun VM.

Lets you choose where PowerJ should find JDK version 1.1 on disk.

Specifies the location of the Microsoft SDK for Java on your machine. This is used to set the classpath when your 1.1 applets and applications are run with the Microsoft Java VM.

Lets you choose where PowerJ should find Microsoft SDK for Java.

Lets you specify a standard set of folders that are added to your CLASSPATH every time PowerJ runs a target. The folders and files specified here will not appear in the CLASSPATH properties for targets.

You need to restart PowerJ after making changes to have the changes take effect at design time.

Use this to add a JAR or ZIP file to the global CLASSPATH.

Use this to add a folder to the global CLASSPATH.

Allows you to specify how the servlet will be run.

Lists the applets in the Web Application and allows you to select one to debug.

Allows you to debug the Dynamo script used by some part of this Web Application.

Disables the debugging facilities while you are running the Web Application.

Allows you to debug a web server extension used by the Web Application. You can specify what extension to debug by clicking **Configure**.

Allows you to debug one of the applets in the Web Application.

Specifies the Dynamo linked folder where the Dynamo script is located. You may need to create a linked folder in Dynamo before starting to debug.

Uses the Web Post Wizard from Microsoft to publish the Web Application’s files. The wizard is available from Microsoft as part of the Internet Explorer starter kit. See the documentation provided with the Web Post Wizard for more information.

Allows you to specify a server that will be debugged when you run the Web Application.

Specifies the process name to which the debugger should attach in order to debug the server extension.

Uses Microsoft Internet explorer to run your applet.

Removes the missing component from the target.

Specifies the location of the JAR, ZIP, or class files. For JAR and ZIP files, this should be the full path and file name. For class files, you should only specify the full path, not the class name.

Specifies the target where the new class or component will reside.

For a components you should usually select the component library, not its design-time companion (with the DT prefix). For example, if your component library is called MyComponent.lib, chose this as your target rather than DTMYComponent.lib.

Gives a brief description of the selected class or component type. Select an item from the icons above and click **Next**.

Lists the possible types of classes and components. For more information on component types, click **Help**.

Specifies the name that will be used in the code to identify this class or component. You should choose a name that describes the component. The name must not be the same as the name of another class. To ensure that the name is unique, you could give the class a prefix that identifies your organization or department.

Specifies the name of the file where information about the class or component will be stored. The default name is the name of the class plus the .wxc extension. You can change the name, but do not change the extension.

Ensures that this class will be available to users of the DLL. Not used for ActiveX server components.

Specifies the name of the new property. The name must not be the same as the name of any of the component's other properties. The property's name can only contain characters that are valid as function names in Java and C++ (spaces are not permitted).

The name should describe the property, or the effect of turning the property on. For example, the when the Disabled property is turned on, the object is disabled.

Specifies the type for the property. Setting this type allows PowerJ/Power++ to create methods for setting and retrieving the value of the property. If the property reflects the value of some internal value or state, the property's data type should be the same as the data type of the internal value or state.

For example, the data type for a property with two states (on/off, true/false) is boolean.

Allows you to specify the prototype for the access functions associated with the component’s new property. The default access functions will set or retrieve a value with the same data type as the property.

Choose this option if setting or retrieving the property requires more complex processing than simply changing one internal value. Otherwise, choose **Member variable with inline Get- Set-**.

Specifies the prototypes for the access functions for the new property. A property usually has two types of access functions - one to set and the other to retrieve the value of the property.

The access method that modifies the value of the property usually has the Set prefix. The access method that retrieves the value of the property usually has the Get prefix. For example, the access methods for a property called Instrument would be called SetInstrument and GetInstrument.

Access functions can be overloaded. For example, to set the value of a string, you might want to pass a pointer (char*), or another string.

Specifies that the only purpose of this property is to modify the value of an internal variable. If you choose this option, PowerJ/Power++ will create simple member functions that set and retrieve the value of the internal variable in the class declaration.

This option is more efficient for properties that only modify the value of an internal variable. If a property requires more processing, you should turn on the **Member functions** option.

Specifies the name of the internal member variable that stores the value of this property.

Specifies the name of the new method.

Allows you to specify one or more prototype for the method. A method may have several overloaded prototypes. Prototypes must end with a semicolon.

Use an existing event for the component. For example, if your component is an extension of the basic list box you may just want to use an event that has already been defined for list boxes. Choose the event from the list.

Use a new event for the component. Type the name of the event.

Specifies a descriptive name for the event. This name appears in the Object Inspector and the Events pop-up menu. Only available for new events.

For example, the Click event has an EventID of WClickEvent but a descriptive name of Click (which is seen in menus, the Object Inspector, and so on). This descriptive name is also used in generating names for event handlers. If you chose an existing EventID on the previous page, PowerJ/Power++ automatically uses the descriptive name already associated with that event.

Java topics follow

Specifies that the class contains one ore more method declarations with no implementation. All interfaces are considered to be abstract. Use this for base classes that define methods but do not implement them; the methods are implemented by classes derived from the abstract class.

Specifies the base, or parent class for the new class. You should specify the fully qualified class name, such as java.awt.Component

Prevents this class from being extended or subclassed. Declaring a class to be final allows the compiler to perform some optimizations when invoking methods of the class.

Specifies which interface or interfaces this class implements. To implement more than one interface, separate the interface names by a comma. If your class implements an interface, you must provide an implementation for each method in the interface.

Specifies that this class is an interface, or a class with method definitions but no implementations. Interfaces are like abstract classes, but all their methods must be abstract. In addition, all member variables of an interface must be final and static. In other words, all their member variables must be constants.

Specifies the package to which this class belongs. A class has access to all classes, public and non-public, in its package.

All classes in a package are stored in the same subdirectory, and can access each other without specifying a package name.

Specifies that this class can be accessed by classes outside its package.

This wizard helps you make your class trigger the events of an event listener.

Allows you to trigger the events of an existing listener. Select an event listener from the list and click Finish.

Allows you to trigger the events of a listener that is not known by the system. You must provide the fully qualified name of the listener. You may also need to define the listener by creating a new interface.

You use this option if you need to trigger an event that is not provided by an existing event listener. Your event listener can extend an existing listener, meaning that it has all the events of the existing listener but may have more data or more events.

Specifies that the CAB target will contain an applet and one or more class file.

Specifies that the CAB will contain a library of class files, but no applet.

Specifies the unique name for this CAB file.

Generates a new unique identifier for this CAB file.

Specifies the unique identifier for this CAB file. To change identifiers, it is recommended that you click **Generate** to have PowerJ generate an ID for you.

Specifies that this CAB should be downloaded as an untrusted application, meaning that it does not require access to sensitive resources on the client machine.

Specifies that this CAB should be downloaded as an trusted application, meaning that it will have more access to resources on the client machine, but the user will generally be warned before the application is allowed to run.

Specifies the version of the CAB file. The four levels of version numbers allow you to increment major and minor version numbers.

Specifies that the CAB does not have a signature. A signature helps the client machine to identify the creator of an applet for security reasons.

Specifies that the CAB has a signature. A signature helps the client machine to identify the creator of an applet for security reasons.

Specifies the number of bytes that PowerJ will reserve for the CAB’s signatures. A signature helps the client machine to identify the creator of an applet for security reasons.

Uses PowerJ's built-in Java compiler to compile Java code.

Uses Microsoft’s Java compiler to compile Java code.

Uses Sun’s Java compiler to compile Java code.

Uses Sun’s Java library when compiling your Java code. When you set this option, your class files should be able to work with either the Sun or Microsoft Java VM.

Uses Microsoft's Java library when compiling your Java code. When you set this option, your class files may not work with the Sun Java VM.

Causes PowerJ to use the settings from the Debug version of this target for the Release version as well. Checking this causes the Debug settings to override the settings in this dialog.

Use the PowerJ build strategy options to specify whether you prefer faster builds (that may cause run-time errors) or having PowerJ rebuild every file when one or more have changed.

Rebuilds all the class files associated with the target when any source file has been changed since the last build.

Rebuilds only the class files whose source files have changed since the class file was last built.

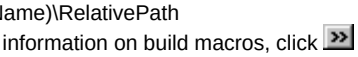
This can reduce the amount of time required to build the target. However, there is a chance that you will introduce error conditions that are not detected by the build operation. For example, suppose that you edit the definition of class A to remove the definition of a particular function. Also suppose that class B uses that function. When you rebuild the target, PowerJ rebuilds the class file for class A (since it has recently changed), but doesn't look at class B. Therefore, PowerJ doesn't notice that class B tries to use a function that no longer exists. This can lead to run-time errors.

If you use this option you should occasionally do a clean and build of the target to detect errors.

Stipulates the options used when building a target for debug.

Performs no special optimization.

Lists the locations of header files. You can specify a path relative to the folder where the target is built using the $^: macro, or use build macros to specify a given folder:
$^:..\..\MyFolder
$(MacroName)\RelativePath
PowerJ/Power++ searches folders in order from the top to the bottom of the list.
For more information on build macros, click 

Allows you to specify a folder to add to the list. You can specify a path relative to the folder where the target is built using the $^: macro, or use build macros to specify a given folder:
$^:..\..\MyFolder
$(MacroName)\RelativePath
For more information on build macros, click 

Inserts the folder that you typed into the list. The folder is inserted just before the highlighted folder. You can specify a path relative to the folder where the target is build, or use build macros to specify a given folder:
..\..\MyFolder
$(MacroName)\RelativePath

Deletes the selected folder from the list.

Moves the selected folder down by one position in the list.
PowerJ/Power++ searches folders in order from the top to the bottom of the list.

Moves the selected folder up by one position in the list.
PowerJ/Power++ searches folders in order from the top to the bottom of the list.

Inserts the folder that you typed at the top of the list.

Deletes the selected folder from the list.

Allows you to control the linker settings.

Creates a separate “map file” for the module.
A map file tells you where the variables and functions in your program are located in memory, and can be useful when debugging.

Does not create a “map file” for the module.
A map file tells you where the variables and functions in your program are located in memory, and can be useful when debugging.

Allows you to specify the location on disk of the map file.

Forces your program symbols to have the exact pattern of upper and lower case letters.

Ignores the difference of upper and lower case letters for symbols.

Specifies the size of memory allocated for the program stack (bytes).

Performs floating point operations in software.
Your program will be slower, but it will run on computers without mathematical co-processors, such as 486-SX and some 386 computers.

Performs floating point operations with the math-coprocessor.
You program will be faster, but will only run on computers with math-coprocessors installed.

Performs floating point operations with the main processor.
Your program will be slower, but it will run on computers without math-coprocessors, such as 486-SX and some 386 computers.

Click this if you want your program to run on a 286 or better computer.

Click this if you want your program to run on a 386 or higher computer.
Your program will not run on a 286 computer.

Click this if you want your program to run on a Pentium computer.
Your program will run on a 386 computer, but is optimized for the Pentium.

Lists your program's #define macros.
When you insert a macro in this list, the compiler defines that macro at the beginning of your source file.

Inserts the library that you typed into the list.

Deletes the selected libraries from the list.

Allows you to select the name of a library with the mouse.

Adds the reference to the list. References are symbols that are used by your program but not excplicitly referenced. By adding a reference to a symbol, you instruct PowerJ/Power++ to link that symbol even if the symbol is not used by the target.

Deletes the reference from the list. References are symbols that are used by your program but not excplicitly referenced.

Lists the symbols that are used by your program but not explicitly referenced. By adding a reference to a symbol, you instruct PowerJ/Power++ to link that symbol even if the symbol is not used by the target.

Specifies a symbol that is used by your program but not explicitly referenced. By adding a reference to a symbol, you instruct PowerJ/Power++ to link that symbol even if the symbol is not used by the target.

Specifies the name and relative path of the file.

Specifies the location on disk of the current file.

Specifies the size of the current source file in bytes.

Specifies the date when the source file was last saved.

Causes no debugging information to be generated for the target. This makes debugging the target very difficult but it reduces the amount of disk space needed to build the target.

Includes line numbers in the compiled code for use with debugging. The value of some variables may not be available for debugging because of optimizations.

Provides all possible information for debugging. Choosing this option reduces the number of optimizations that can be performed.

Displays all warning messages. Warning messages can point to code bugs.

Suppresses display of compiler warning messages. Warning messages can point to code bugs.

Halts the build process when there are warnings. Warning messages can point to bugs in your code.

Continues the build process even when there are warnings. Warning messages can point to bugs in your code.

Check this to balance the importance of speed and size of your target (application, DLL, etc.).

Adds the macro to the list.

Changes the macro selected in the list to the values entered above.

Removes the selected macro from the list.

The name in the macro:
#define name value

The value in the macro:
#define name value

Empties the above text boxes and allows you to enter a new macro.

Lists the macros defined for this file. These macros will be defined when this file is compiled (or files of this type for default options).

Specifies the name used by other applications to access a function or variable in this DLL.
If you are accessing this library from another target created with PowerJ/Power++ you do not need to specify an alias; just use the function and global variable names as they are defined in the source code.

Specifies the internal (mangled) name of a function or variable in this DLL. This name usually contains characters not used in function names since it is an internal name.
If you are accessing this library from another target created with PowerJ/Power++ you do not need to specify an alias; just use the function and global variable names as they are defined in the source code.

Lists the internal (mangled) names of functions and variables in this DLL and the names used by other applications (aliases) to access them. Aliases are sometimes needed when a DLL is used by an application that expects entry-point names to be decorated differently than PowerJ/Power++ does by default.
If you are accessing this library from another target created with PowerJ/Power++ you do not need to specify an alias; just use the function and global variable names as they are defined in the source code.

Changes the selected alias/name pair to the new values specified.

Deletes the selected alias/name pair from the list.

Empties the edit boxes, allowing you to type in new values.

Adds to the list the alias/name pair you type.

Uses optimizations for 386 computers. Regardless of this setting, the program will run on any 386 or greater.

Uses optimizations for 486 computers. Regardless of this setting, the program will run on any 386 or greater.

Uses optimizations for 586 (Pentium) computers. Regardless of this setting, the program will run on any 386 or greater.

Uses optimizations for 686 (Pentium Pro) computers. Regardless of this setting, the program will run on any 386 or greater.

Allows you to specify the level of optimization used by the compiler. Regardless of this setting, the program will run on any 386 or greater.

Lists the places in the program where execution will stop (breakpoints). If the breakpoint specifies advanced settings, the breakpoint will only trigger when all the conditions are satisfied.

Enables every breakpoint in the project. If a breakpoint is enabled, it will be triggered. If it is disabled, it will be ignored (and execution will continue as usual).

Disables every breakpoint in the project. If a breakpoint is disabled, it will be ignored (and execution will continue as usual). If it is enabled, it will be triggered.

Removes all the breakpoints. If you don't want to remove breakpoints, but you want to ignore them, you can disable them with the **Disable All** button, or by disabling only specific breakpoints.

Allows you to specify a line of code to be executed when the breakpoint is triggered, and optionally, to continue program execution without pausing at the breakpoint.

Allows you to specify conditions under which the breakpoint will be triggered.

Closes the breakpoint editor and saves any changes you have made.

Creates a new breakpoint.

Triggers the breakpoint only if the statement entered here evaluates to TRUE, or a non-zero number.

Skips the breakpoint ‘n’ times before triggering, then triggers the breakpoint every time afterwards. You specify ‘n’ in the edit box. Uncheck to trigger the breakpoint each time it is executed.

If you have specified a condition for the breakpoint, this count (n) is only incremented when the condition is TRUE.

Executes the statement you specify when the breakpoint triggers.

If you have specified a condition and a count (‘n’) for the breakpoint, the condition must be satisfied 'n' times before the breakpoint is triggered.

Continues program execution when the breakpoint triggers. The statement you entered will be executed just before the breakpoint.

Shows or hides more sophisticated debugging options.

Lists the available symbols. Select the symbol whose value or address you want to use.

Specifies the name of the symbol you want to use. Type the full or partial symbol name, or select from the list.

Allows you to select a symbol to be placed in the text box.

Lists the modules used by the current target and allows you to select one.

Select the module that contains the symbol you want to use.

Lists all symbols including symbols from imported libraries. Uncheck this to display only symbols defined by in the current target.

Closes the dialog and returns the symbol you selected.

Scrolls the assembly window to the address you specify. Type the new address or click **Symbol Lookup** to select the symbol that corresponds to the address you want to see.

You can enter an address or an expression. If you enter an expression, the expression will be evaluated as an address.

Scrolls the memory window to the address you specify. Type the new address or click **Symbol Lookup** to select the symbol that corresponds to the address you want to see.

You can enter an address or an expression. If you enter an expression, the expression will be evaluated as an address.

Changes the address of the breakpoint. Type the new address or click **Symbol Lookup** to select the symbol that corresponds to the address.

You can enter an address or an expression. If you enter an expression, the expression will be evaluated as an address.

Changes the value stored in the specified memory location. Type the new value or expression.

Changes the value of the specified variable. Type the new value or expression.

Allows you to choose a symbol representing the address of the assembly code that you want to see.

Allows you to choose a symbol representing the new address of the breakpoint.

Allows you to choose a symbol representing the address in memory that you want to see.

Closes the dialog and displays the address you specified at the top of the assembly window.

Closes the dialog and displays the address you specified at the top of the memory window.

Closes the dialog and changes the breakpoint to the address you specified.

Closes the dialog and changes the memory value to the value you specified.

Closes the dialog and changes the register's value to the value you specified.

Closes the dialog and changes the variable to the value you specified.

Closes the dialog. The assembly window will not be changed.

Closes the dialog. The memory window will not be changed.

Closes the dialog. The breakpoint will not be changed.

Closes the dialog. Memory will not be changed.

Closes the dialog. The variable will not be changed.

Select the correct symbol from the list of possible matches and click **OK**.

Closes the dialog and returns the unambiguous symbol you selected.

Closes the dialog. Does not return a symbol.

Changes the indices of the first and last array elements that are shown.

Specifies the index of the first array element that you want to see.

Specifies the index of the last array element that you want to see.

Closes the dialog and changes the display bounds of the array.

Specifies the watch or inspect expression.

Closes the dialog and changes the value of the variable to the value you specified.

Just wait. PowerJ/Power++ is copying the target to the remote machine.

To stop this process, click **Cancel**.

Just wait. PowerJ/Power++ is building a sorted list of symbols in your program.

Specifies which target program to run and with what parameters.

Specifies the options for memory tracing.

Cuts off communications between this computer and the remote computer.

Monitors communications between this computer and the remote computer. When the lights move from left to right, the computers are communicating.

Specifies the text that you want to Find.

Specifies the text that will replace the text in the **Find What** box.

Allows you to specify the direction of the search (whether text is searched for before or after the search start position).

Searches for occurrences that are words, and not a part of a larger word.

Searches for text before the search start location.

Searches for text after the search start location.

Searches for text that has the same pattern of upper and lower case as the text you stipulate.

Uses a powerful search mechanism, where you specify a template to search for. Using pattern matching, you could search for any character followed by any number, for example.
For more information, click 

Starts searching again at the other end of the file when the end of the text is reached.
Use this if you want to search the entire file and your cursor is in the middle of the file.

Searches only in the highlighted area.

Locates but does not replace the next occurrence of the text that you specified. You can replace the text when it is found.

Replaces this occurrence of the search text with the text you specified.

Replaces every occurrence of the search text with the text you specified.
Only click this if you are sure that the find/replace text is correct and you want to replace every occurrence.

Repositions the editor at the line number you specify.

Repositions the editor at the line number you specify and closes this dialog box.

Locates files containing the text you stipulate.

Locates files in your project containing the text you stipulate.

Allows you to specify the text to be searched for, and the rules for searching.

Shows a list of the places where the text you specified was found.

Allows you to specify where the system will search for text.

Specifies the text you want to find.

Uses a powerful search mechanism, where you specify a template to search for. Using pattern matching, you could search for any character followed by any number, for example.
For more information, click 

Searches for text that has the same pattern of upper and lower case as the text you stipulate.

Specifies the folder to search for files containing text. Click **Browse** to select the folder.

Searches the folder you specified and all sub-folders. Sub-folders are folders within folders.

Searches component headers.

Searches in all event handlers and user-defined functions.

Searches the C-library header files included in your project.

Searches the Java Development Kit classes used by your project.

Searches for occurrences that are words, and not a part of a larger word.

Searches for all occurrences. Uncheck this to find the first occurrence only.

Lists the files where the text was found. To edit one of the files, select it and click **Edit**.

Lists the places where the text was found. To edit the text, select the occurrence and click **Edit**.

Halts the search.

Opens the selected item for editing in a new edit window.

Starts the search.

Opens the file for editing and closes this dialog box.

Allows you to specify the folder to be searched.

Allows you to choose the target whose source files will be searched.

Specifies the number of spaces that are displayed in place of a tab character.

Specifies the number of spaces that the editor window inserts to automatically indent a block of code.

Replaces tab characters with a number of spaces. Above, type the number of spaces.

Maintains tab characters while editing instead of replacing them with a number of spaces.

Determines whether tab characters are retained in a file or replaced by the number of spaces you specified above.

Resets all tab settings to the defaults.

Colors text based on what it represents (e.g. comment, string).

Shows the different kinds (syntax elements) of text and the colors used to display them. Click on the text representing the syntax element you want to change. Then select the foreground and background colors for that element.

Specifies the name of the color scheme.

Specifies what the text represents (e.g. comment, string).

Changes the colors in the editor window without closing this dialog.

Stores this color scheme. Once stored, you can recall it at a later time. By selecting it from the **Scheme** list.

Erases this color scheme from the list of stored schemes.

Specifies the background color for the current **Syntax element**. Select a color or **Default** from the list.
To specify the colors for a different syntax element, click the sample code window near a character representing the syntax element you want to change or select an element from the list of elements.

Specifies the foreground color for the current **Syntax element**. Select a color or **Default** from the list.
To specify the colors for a different syntax element, click the sample code window near a character representing the syntax element you want to change or select an element from the list of elements.

Specifies the default background color for the editor window. This allows you to change the background color of all syntax elements that use the default background color. Select **Default** from this list to use the standard background color for the system.

Specifies the name that will identify the current color scheme. This allows you to store custom color schemes and change back and forth to different color schemes.

Allows you to insert an object’s method. You enter the arguments for the method you choose. If a method is overloaded (multiple parameter lists), you can select the one that you want to use.

This button is only available when you’re editing a program (event handler).

Displays the PowerJ/Power++ Component Library Reference for the selected method or component.

Lists the methods available for each component.

The reference card allows you to paste a method directly into your program. The Parameter Wizard then prompts you for the method’s arguments.

Locate the method in the tree and select it. Then click **Parameters** to enter the Parameter Wizard. If the method is overloaded (multiple parameter lists), you can choose the one you want from the list in the Parameter Wizard.

Specifies the class that contains the method you are looking for.

Specifies the method you are searching for.

Enter the full or partial name of the method and press the **Find** button.

Allows you to search for a method by name from a list of all available methods for a class.

Lists the methods that match your search criteria. Use this to put a method into your program or to access the online Reference Manual. Click **Options** to change the way PowerJ/Power++ searches for a method**.**

Select the method that you want to use. If the method is overloaded, select the correct version from the box below. Then click **Parameters**.

Allows you to change the way the list is searched.

For example, you can search for all methods containing the letters ooky, or only those methods that start with the letters ooky.

Causes PowerJ/Power++ to start searching for the method.

Empties the search box and begins a new search.

Describes the way PowerJ/Power++ will search the list.

Click **Options** to change search method.

Counts the methods that match your search criteria.

Click **Options** to change search method.

Specifies how PowerJ/Power++ searches for a method.

Choose a method from the drop-down list.

Waits for you to type the search phrase and click **Find Now**.

This method doesn’t search while you type.

Updates the search results after each key that you press.

Use this method for incremental or partial searches.

Waits for a pause in your typing before searching.

Allows you to specify what libraries to search.

Enter the variables used for the method’s parameters and return value.

Specifies the object whose method you want to call.

Type variables used for the method’s parameters here.

Required parameters are arguments of the method that you must specify.

Type the values for optional parameters here.

If you don’t specify values for optional parameters, PowerJ/Power++ uses their default values.

Check this if you want to store the return value in a variable.

Type the name of the variable where you want to store the return value.

No help topic for this button.

This method has more than one version. Select the version of the method that you want to use in your program.

Methods with more than one version are said to be overloaded. You can tell the difference between overloaded methods by their parameter lists.

Lists the versions of the method. Select the version that you want and click **Parameters**.

Methods with more than one version are said to be overloaded. You can tell the difference between overloaded methods by their parameter lists.

Sets the alignment for this label to the specified alignment.
For more information, click 

Sets the specified boolean to indicate whether or not this TextComponent should be editable.
For more information, click 

Sets the font of the component.

For more information, click 

Sets the button with the specified label.

For more information, click 

Sets the button with the specified label.

For more information, click 

Sets whether this list should allow multiple selections or not.

For more information, click 

Sets the resizable flag.

For more information, click 

Sets the resizable flag.

For more information, click 

Sets the Checkbox to the specifed boolean state.

For more information, click 

Sets the text for this label to the specified text.
For more information, click 

Sets the text of this TextComponent to the specified text.
For more information, click 

Sets the title of the Dialog.
For more information, click 

Sets the title for this Frame to the specified title.
For more information, click 

Sets the value of this Scrollbar to the specified value.
For more information, click 

Sets the alignment for this label to the specified alignment. Possible values are `Label.LEFT`, `Label.RIGHT`, and `Label.CENTER`.

For more information, click 

Sets the block increment for this scroll bar.

For more information, click 

Sets the number of columns for this text area.

For more information, click 

Sets the number of columns in this text field.

For more information, click 

Sets the echo character for this text field.
For more information, click 

Sets the flag that determines whether or not this text component is editable.
For more information, click 

Enables or disables this component, depending on the value of the parameter `b`. An enabled component can respond to user input and generate events. Components are enabled initially by default.

For more information, click 

Sets whether or not this menu item can be chosen.

For more information, click 

Sets the font of this component.
For more information, click 

Sets the button's label to be the specified string.
For more information, click 

Sets this check box's label to be the string argument.

For more information, click 

Sets the label for this menu item to the specified label.
For more information, click 

Sets the maximum value of this scroll bar.

For more information, click 

Sets the minimum value of this scroll bar.

For more information, click 

Sets the flag that determines whether this list allows multiple selections.

For more information, click 

Sets the orientation for this scroll bar.

For more information, click 

Sets the resizable flag.

For more information, click 

Sets the resizable flag, which determines whether this frame is resizable. By default, all frames are initially resizable.

For more information, click 

Sets the number of rows for this text area.

For more information, click 

Enable/disable SO_TIMEOUT with the specified timeout, in milliseconds. With this option set to a non-zero timeout, a call to accept() for this ServerSocket will block for only this amount of time. If the timeout expires, a **java.io.InterruptedIOException** is raised, though the ServerSocket is still valid. The option **must** be enabled prior to entering the blocking operation to have effect. The timeout must be > 0. A timeout of zero is interpreted as an infinite timeout.

For more information, click 

Enable/disable SO_TIMEOUT with the specified timeout, in milliseconds. With this option set to a non-zero timeout, a read() call on the InputStream associated with this Socket will block for only this amount of time. If the timeout expires, a **java.io.InterruptedIOException** is raised, though the Socket is still valid. The option **must** be enabled prior to entering the blocking operation to have effect. The timeout must be > 0. A timeout of zero is interpreted as an infinite timeout.

For more information, click 

Sets the state of this check box to the specified state. The boolean value `true` indicates the "on" state, and `false` indicates the "off" state.

For more information, click 

Enable/disable TCP_NODELAY (disable/enable Nagle's algorithm).
For more information, click 

Sets the text for this label to the specified text.
For more information, click 

Sets the text that is presented by this text component to be the specified text.
For more information, click 

Sets the title of the Dialog.
For more information, click 

Sets the title for this frame to the specified title.
For more information, click 

Sets the unit increment for this scroll bar.

For more information, click 

Sets the value of this scroll bar to the specified value.
For more information, click 

Shows or hides this component depending on the value of parameter `b`.
For more information, click 

Sets the visible amount of this scroll bar.

For more information, click 

Regular expressions

If **Pattern matching** is checked in any PowerJ or Power++ Find dialog box, the program will interpret the text you type in the **Find** and **Replace** fields as *regular expressions*. A regular expression specifies a pattern of characters for a search or replace operation.

Search strings

The following table lists the strings with special meaning, or *metacharacters*, that you can use in PowerJ and Power++ search regular expressions.

String	Matches
`^`	The beginning of a line of text.
`$`	The end of a line of text.
`.`	A period matches any single character. Therefore, `".at"` matches `bat`, `cat`, `eat`, and so on.
`*`	Matches zero or more occurrences of the preceding character. For example, `Whe*` matches `Wh`, `Whe`, `Whee`, `Wheee`, and so on.
`+`	Matches one or more occurrences of the preceding character. For example, `Whe+` matches `Whe`, `Whee`, `Wheee`, and so on, but not `Wh`.
`?`	Matches zero or one occurrence of the preceding character. For example, `Whe+` matches either `Wh` or `Whe`.
`@`	Everything after the `@` is searched with case sensitivity. For example, `@This` only matches `This`, not `this` or `THIS`.
`~`	Everything after the `~` is searched with case insensitivity. For example, `~This` matches `this`, `THIS`, `This`, and so on. `@` and `~` can be combined within a regular expression to turn on case sensitivity in some areas and turn it off in others.
`\`	Turns off the special meaning of the next character. For example, `\$` represents a dollar sign character, not the

	end of the line.
`(`reg exp`)`	Use parentheses to treat the enclosed regular expression as a single unit. For example, `(abc)*` stands for zero or more occurrences of the complete string `abc`.
`!`	If `!` appears as the first character in a regular expression, all subsequent characters have any special meanings they may have. If `!` appears anywhere else in a regular expression, it has no special meaning.
`a|b`	Matches either `a` or `b`. For example, `(;| )$` matches either a semicolon or a blank at the end of a line.
`[abc]`	Matches any one of the characters inside the square brackets. For example, `[be]at` matches `bat` or `eat`, but not `cat`.
`[a-z]`	Matches any one of the characters in the range that appears in the square brackets. For example, `[0-9]` matches any single digit.
`[^abc]`	Matches any character *except* the ones listed in the square brackets. For example, `[^be]at` matches any string ending in `at` except `bat` and `eat`.
`[^a-z]`	Matches any character except the ones in the range that appears in the square brackets. For example, [^0-9] matches any character except a digit.

Special replacement strings

In a replace operations with pattern matching, the following replacement strings have special meaning.

String	**Replaces**
`&`	Stands for the string matched by the find string. For example, if you ask to replace `The` with `&m`, it changes `The` into `Them`.

`\n`	Stands for a linefeed (newline) character.
`\t`	Stands for a horizontal tab character.
`\u`	Changes the next item in the replacement string to upper case. For example, `\u&` stands for the original find string converted into upper case.
`\l`	Changes the next item in the replacement string to lower case. For example, `\l&` stands for the original find string converted into lower case.
`\U`	Changes all following items into upper case until the appearance of `\e` or `\E`. For example, `\U&&\E` stands for two copies of the original find string, both converted into upper case.
`\L`	Changes all following items into lower case until the appearance of `\e` or `\E`. For example, `\L&&\E` stands for two copies of the original find string, both converted into lower case.
`\e` or `\E`	Marks the end of a `\U` or `\L` construct.
`\`*number*	Refers to the string that matches the *number*'th parenthesized part of the find string. For example, suppose the find string is `(a)b(c)` and the replacement string is `\2\1`. The result is `ca`, since `\2` corresponds to the second parenthesized part `(c)` and `\1` corresponds to the first `(a)`.
`\\`	Stands for a backslash character.

Notes

Regular expression metacharacters can be combined in many ways. For example:

```
^([0-9])+
```

This stands for any line beginning with one or more digit characters.

One of the most commonly used search regular expressions is `.*` (a period followed by an asterisk) which stands for zero or more characters of any kind. For example:

```
W.*EventData
```

This matches any string of characters beginning with `W` and ending with `EventData`. This would match such strings as

```
WEventData
WDragEventData
WTreeViewEventData
When you see a piece of EventData
```

Regular expressions never match more than one line of text. For example, `W.*EventData` would not match a text string where the `W` appeared on one line and `EventData` appeared on some later line.

Regular expressions always match the longest appropriate string. For example:

```
N.*o
```

This regular expression matches the entire line:

```
Now I have to go to the zoo
```

This occurs instead of matching the shorter pieces:

```
No
Now I have to
Now I have to go
```

PowerJ and Power++ keyboard shortcuts

PowerJ and Power++ provide keyboard shortcuts for many operations.

PowerJ and Power++ keyboard shortcuts

Help shortcuts

F1	Get help about current context

PowerJ and Power++ keyboard shortcuts

Project shortcuts

CTRL+S	Save current project
F5	Run project
CTRL+F5	Build target

PowerJ and Power++ keyboard shortcuts

View shortcuts

CTRL+>	Go to next window	
CTRL+<	Go to previous window	
F7	Switch to the code editor window associated with the current form. If the code editor window is already active, switch to the form design window for the active class.	
F4	Open Object Inspector	
SHIFT+F2	Open Reference Card	
SHIFT+F3	Open Classes window	
SHIFT+F4	Open Objects window	
SHIFT+F5	Open Files window	
SHIFT+F6	Open Resources window	Only available for C++.
SHIFT+F7	Open Targets window	
SHIFT+F11	Open Debug Log	
SHIFT+F12	Open Error Log	
CTRL+F6	Go to next window	Only available for C++.
CTRL+SHIFT+F6	Go to previous window	

PowerJ and Power++ keyboard shortcuts

Searching shortcuts

CTRL+SHIFT+F3	Find in project
CTRL+SHIFT+F	Find in Files

Debugging shortcuts

SHIFT+F9	Breakpoints view
CTRL+1	Locals view
CTRL+2	Call Stack view
CTRL+3	Watches view
CTRL+4	Assembly view
CTRL+5	Registers view
CTRL+6	FPU Registers view
CTRL+7	Threads view
CTRL+8	Memory view
CTRL+9	Stack view
F10	Step over
F8	Step into

The following shortcuts are available for C++ debugging, and Java debugging using the Microsoft VM.

CTRL+UPARROW	Go up the call stack
CTRL+DOWNARROW	Go down the call stack

Form design window shortcuts

The following shortcuts apply when the form design window is active.

CTRL+C	Copy the selected object(s) to the clipboard
CTRL+X, SHIFT+DEL	Cut the selected object(s) to the clipboard
CTRL+V, SHIFT+INS	Paste any objects held by the clipboard
DEL	Delete the selected object(s)
SHIFT+LEFT/RIGHT	Decrease/Increase the width of the selected component
SHIFT+UP/DOWN	Decrease/Increase the height of the selected component
CTRL+arrow key	Move selected object(s) in the specified direction
CTRL+SHIFT+ARROW	Move faster in the indicated direction
SHIFT+F10	Open context menu for selected object
Application Key	Open context menu for selected object
TAB	Select next object in tabbing order
SHIFT+TAB	Select previous object in tabbing order
Arrow keys	Select next closest object in the specified direction
ENTER	Open Object Inspector
ALT+ENTER	Open selected object’s property sheet

Object inspector shortcuts

The following shortcuts apply when the Object Inspector is active.

TAB	Cycle from right column to left to combo box at top giving name of object
SHIFT+TAB	Like TAB, but reverse order
ENTER	Accept current settings and move back to form design window
ALT+DOWNARROW	Drop down a combo box showing possible values for right column
CTRL+ENTER	Click the … button in the right column
Arrow keys	Move one step through list
PAGEUP, PAGEDOWN	Scroll through list
Double-click	Move to next item in combo box or … button; or if on seperator, resize cloumns to fit values
Other characters	In left column, search for entry beginning with those characters; in right column, **Properties** page, change value of property

Editor shortcuts

The following shortcuts apply when the active window is a code editor window.

General shortcuts

F7	Switch to the form design window associated with the current editor window. If the form design window is already active, switch to the code editor window for the active form.

Run-related shortcuts

F5	Run target
CTRL+F5	Build (compile) target
F9	Toggle breakpoint on current line

Text manipulation

CTRL+A	Select all text in the editor window
CTRL+U	Create a new user function
CTRL+P	Print contents of current window
CTRL+Z	Undo previous editing action
CTRL+Y	Redo previously undone action
DEL	Delete selected text
CTRL+C	Copy selected text to the clipboard
CTRL+X	Cut selected text to the clipboard
CTRL+V	Paste text from the clipboard
CTRL+L	Cut the current line to the clipboard
CTRL+SHIFT+L	Delete the current line
CTRL+BACKSPACE	Delete previous word
CTRL+DEL	Delete next word
CTRL+T	Indent current line
CTRL+D	Outdent current line (remove one level of indentation)
F6	Open prototype Only available for C++.
SHIFT+F10	Open context menu

Moving through text

CTRL+F	Find/replace in code
F3	Find next occurrence of find string
F2	Open the source corresponding to the selected function name
CTRL+G	Go to line
CTRL+J	Jump to previous cursor location. (Undo Find, F6, F2)
CTRL+]	Go to matching brace
F12	Go to next error found in code
SHIFT+F12	Go to previous error found in code

F11	Remove errors
UPARROW	Go to previous line
DOWNARROW	Go to next line
PAGEUP	Go up one page
PAGEDOWN	Go down one page
HOME	Go to beginning of line
END	Go to end of line
CTRL+HOME	Go to beginning of code
CTRL+END	Go to end of code
CTRL+TAB or ALT+B	Open the drop-down list of bookmarks
ALT+UPARROW	Go to preceding bookmark
ALT+DOWNARROW	Go to next bookmark
CTRL+M	Add a bookmark on current line
CTRL+SHIFT+M	Go back to most recently created user bookmark

Miscellaneous shortcuts

F2	Rename (in places where such an operation makes sense; for example, Rename on Classes window renames the selected object)
CTRL+O	Open file in Files window
TAB, SHIFT+TAB	Switch the focus from one part of the window to the other in the Files window and Classes window