# Table of Contents

# Table of Contents

# Table of Contents

# Table of Contents

# Table of Contents

# Table of Contents

# Table of Contents

# Table of Contents

# Table of Contents

# Table of Contents

# Table of Contents

# Table of Contents

# Table of Contents

# Table of Contents

# Table of Contents

# Table of Contents

Welcome to Komodo

# Table of Contents

# Table of Contents

# Table of Contents

# Table of Contents

# Table of Contents

# Table of Contents

# Table of Contents

# Welcome to Komodo

Komodo is ActiveState's cross−platform, multi−language Integrated Development Environment (IDE). Komodo supports development in numerous languages, including Perl, Python, PHP, XSLT, Tcl, JavaScript, and more.

***Get started fast*** with Komodo's Sample Project.

***Upgrading?*** See what's new in this release.

***Feature Showcase***: Fast feature demos showing advanced search functionality, tool usage and more.

***Komodo Tutorials***: Perl, Python, PHP, Tcl, XSLT and Run Commands.

***Komodo Professional*** includes all the features of Komodo Personal, plus:

- Source Code Control
- Visual Package Manager
- GUI Builder

Getting started with Komodo is as easy as opening a file and beginning to edit. However, to ensure that you don't miss any of Komodo's features, take a look at the components of the Komodo workspace described below.

Next, ***Get Started Fast*** with Komodo's Sample Project. Use the Sample Project to familiarize yourself with Komodo's project management, editing features and debugging functionality.

- Open the Sample Project and Sample Files
- Edit a Sample Program
- Debug a Sample Program

# Starting Komodo

## Windows

From within the Windows environment, use one of the following methods to launch Komodo:

- double−click the Komodo desktop icon
- launch Komodo from the Windows program menu (***Start/Programs/ActiveState Komodo 3.0/Komodo***)
- right click a file name in Windows Explorer (and other dialogs that support the standard Windows right−click context menu) and select ***Edit with Komodo***

To start Komodo from a command prompt, enter:

```
komodo [options] [filenames]
```

Multiple filenames may be specified; all specified filenames will be loaded in the Komodo editor pane.

The following command–line options are available:

- **Help**: `-h` or `--help`
- **Show Komodo version**: `-V` or `--version`
- **Open at a specified line number**: `-l` *line* or `--line=`*line*
- **Open with a specified range selected**: `-s` *range* or `--selection=`*range*
  (e.g. `komodo -s 1,5-2,15 example.py` would open `example.py` and select from line 1 and column 5 to line 2 column 15)

## Unix

To start Komodo from a shell prompt, enter:

```
komodo [options] [filenames]
```

The same options described in the Windows command prompt section above apply.

Desktop icons and taskbar applets are not added automatically during installation on Unix. Check your window manager documentation for information on creating these manually. A choice of Komodo icons is available. By default, the icon files (.xpm) are stored in the Komodo installation directory.

# The Komodo Workspace

## The Start Page

Komodo's Start Page is displayed by default when Komodo is first opened, and the **Start Page** tab persists when other files are opened. The Start Page provides quick access to recently opened files and projects. By default, the Start Page also displays links to tutorials and Komodo's Sample Project, as well as the Tip of the Day. Select **Edit**/**Preferences**/**Appearance** to change the contents of the Start Page, or to configure the number of recent files and projects.

To clear the names of selected files or projects from the Start Page, check the boxes next to the filenames, then click **Remove**. To clear the complete list of files or projects, click the "X" button at the top right of each list.
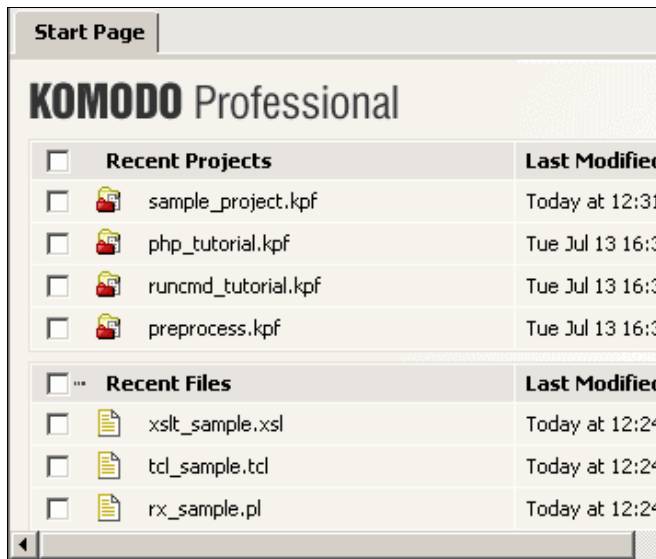
## Title Bar

The title bar displays the path and name of the active file. During debugging, the title bar indicates the state of the debugger. See Debugging Programs for more information.



## Menus

The default drop−down menus are: **File**, **Edit**, **Code**, **View**, **Debug**, **Project**, **Toolbox**, **Tools**, **Window**, and **Help**. The functions accessed from each menu are described in detail in the relevant section of the User Guide. For example, the items accessed from the Debug menu are described in Debugging Programs.



## Context Menus

Komodo displays right−click context menus with options relative to the area of Komodo where the option was invoked, depending the location of the mouse pointer. Use the left mouse button to select items from context menus.

- *Menu Bar Areas and Toolbar Areas*: Options to view or hide individual toolbars and toolbar text.
- *Projects Tab (Project Name)*: Options to open, save, activate, and close the projects, and to add a file to the selected project.
- *Projects Tab (File Name)*: Options to edit, remove or export the selected file ,and access to source code control commands.
- *Toolbox Tab*: Options to work with the specified component.
- *Editor Pane (File Editing Area)*: Options to cut, copy, and paste text, to set a breakpoint, and to edit the file properties and settings.
- *Editor Pane (Tabs)*: Options to close the selected file and to view the file's properties and settings.
- *Bottom Pane*: The context menus available on tabs in the Bottom Pane (e.g. the *Debug* tab and the *Breakpoints* tab) contain subsets of Komodo's top–level menus.
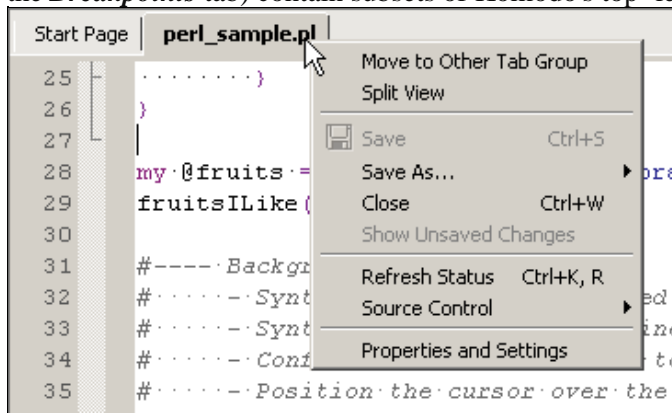


## Toolbars

To hide or show toolbars, or to hide or show button text, do one of the following:

- From the *View* menu, select *Toolbars*.
- Right–click on a menu bar or toolbar, and toggle the check mark beside the pertinent option.
- From the *Edit* menu, select *Preferences*. Click the *Appearance* option and check or uncheck the desired options.

The Standard Toolbar provides quick access to common editing functions. Launch the Komodo User Guide by clicking the Help button.
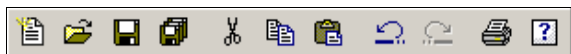


The Tools Toolbar contains the commonly used commands on the *Tools* menu, including Preview in Browser, the Regular Expression Toolkit, the Visual Package Manager (VPM), the interactive shell, and the Object Browser.

The Workspace Toolbar toggles the main components of the Komodo workspace. Use this toolbar to show/hide the Left Pane, Bottom Pane and Right Pane, and to display or shift focus to a specific tab in one of these panes (e.g. the **Toolbox** tab).
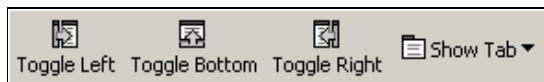


The Debug Toolbar provides quick access to common debugging functions, such as Step In and Step Over. For more information about debugging programs, see Debugging Programs.
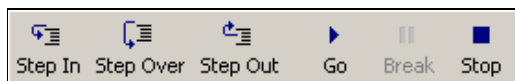


Use the Open/Find Toolbar to open files and search for strings. Find strings in files currently displayed in the editor or in files not currently open in Komodo but located on the filesystem. See Open/Find Toolbar for more information.



The Source Code Control Toolbar (available only in the Professional Edition of Komodo) makes it easy to work with files that are stored in Perforce or CVS. For more about using the Source Code Control Toolbar, see Source Code Control (Komodo Pro).



The Macros Toolbar makes it easier to record, play and save macros. For more information see Macros.



It is also possible to create Custom Toolbars consisting of items that are otherwise stored in the Toolbox or Komodo projects (e.g. run commands, code snippets and directory shortcuts).

http://slashdot.org/ Find in Files Copyright
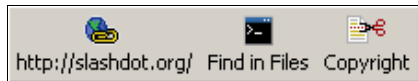
# Left Pane

The Left Pane of the Komodo workspace contains the *Projects* tab and the `Code` tab.

## Projects Tab

The *Projects* tab displays projects that are currently open. Hide or display the components contained in a project by clicking the plus sign to the left of the project name. To display the *Projects* tab, select *View/Tabs/Projects*, or use the associated key binding.
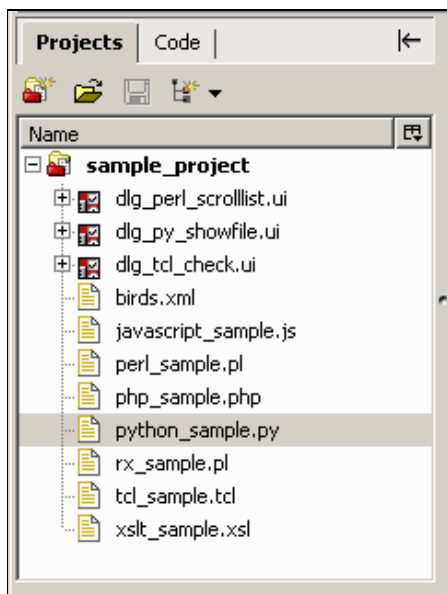


Related Topics:

- Managing Projects and Files
- Managing Tabs and Panes

## Code Tab

The *Code* tab displays a hierarchical view of all code symbols (for example, variables, methods, imports) in an open file. Symbols can be sorted and filtered, and the current scope of a symbol can be located. The lower part of the Code Browser provides additional documentation (when available) on program components. To display the *Code* tab, select *View/Tabs/Projects*, or use the associated key binding.
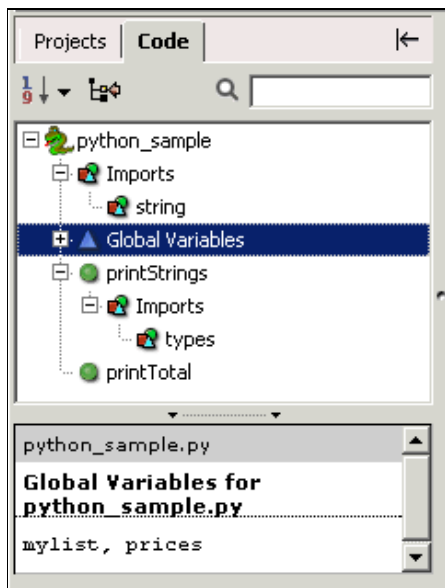
Related Topics:

- Code Browser
- Managing Tabs and Panes

## Right Pane

The Right Pane of the Komodo workspace contains the Toolbox and, optionally, a Shared Toolbox.

### Toolbox Tab

Use the **Toolbox** tab to manage and store Komodo components (for example, frequently used files, code snippets, commands, and URLs). Add items to the Toolbox, as well as to folders within the Toolbox. Items can be imported to the Toolbox and exported as Komodo project files and packages. Items added to the Toolbox are displayed with associated icons for easy identification. To display the **Toolbox** tab, select **View|Tabs|Toolbox**, or use the associated key binding.
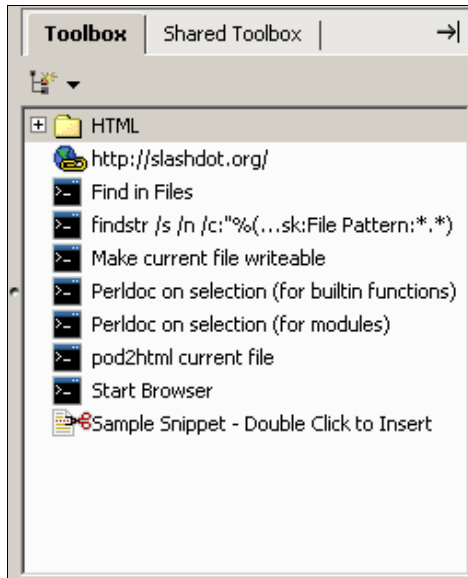
Related Topics:

- [Using the Toolbox](#)
- [Managing Tabs and Panes](#)

## Shared Toolbox Tab

A Shared Toolbox has the same functionality as the [Toolbox](#) except that it can be shared among multiple users. For example, use a Shared Toolbox to store code snippets that are frequently used by a number of programmers. The *Toolbox* tab is only available if the Shared Toolbox preference has been set (select *Edit|Preferences|Shared Support*).
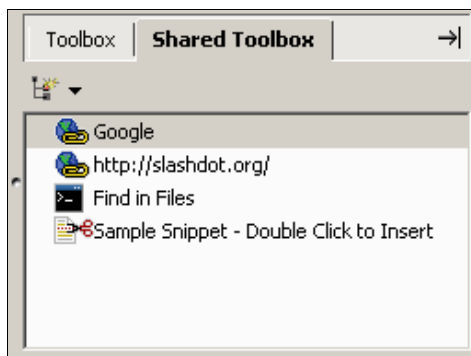


Related Topics:

- [Sharing Toolboxes](#)
- [Managing Tabs and Panes](#)

## Editor Pane

The large pane in the middle of the Komodo workspace is the Editor Pane. The Editor Pane is used for editing and debugging. Each open file has a corresponding tab at the top of the Editor Pane. Change the order of the tabs by clicking and dragging tabs to the desired position. The name of the active file (that is, the file that is currently displayed in the Editor Pane) is displayed in bold text. Use the left and right arrow buttons on the right side of the tabs to scroll though open files. Use the close button "X" on the right side of the tab display to close the active file. An asterisk beside the filename indicates that the file has been changed since it was opened, and needs to be saved. If a file is under source code control, a file status icon to the left of the filename indicates its current status.
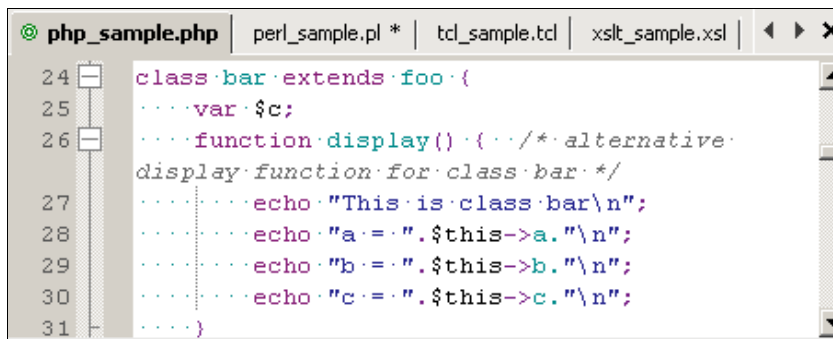


Related Topics:

- Editing Files
- Managing Projects and Files
- Managing Tabs and Panes

## Bottom Pane

The Bottom Pane spans the width of the Komodo workspace and displays at the bottom of the screen. The Bottom Pane contains the following tabs:

- *Breakpoints Tab*: manage breakpoints and spawnpoints in the current debugging session(s)
- *Command Output Tab*: displays the results of commands run in the Run Command dialog box
- *Find Results 1 and Find Results 2 Tabs*: display the results of the Find All function
- *SCC Output Tab*: displays details of source code control commands, such as editing or checking in files
- *Interactive Shell Tab*: displayed when the interactive shell is launched as a stand−alone tool or from within a debugging session
- *Debug Tab*: consolidates views of the debugger output, call stack, program variables (local and global), and watched variables.
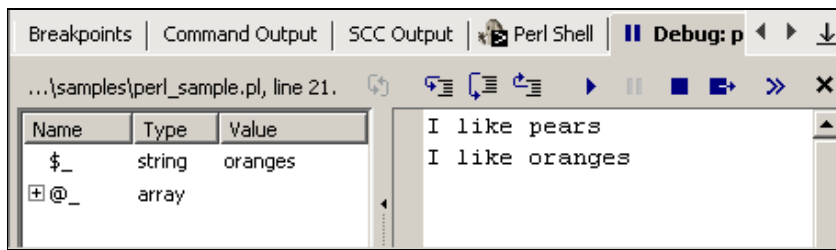
## Managing Tabs and Panes

Use the View menu, Standard Toolbar or Komodo key bindings to choose which tabs are displayed in the Komodo workspace. Use the Standard Toolbar to show and hide the Left Pane, Right Pane and Bottom Pane.

### Showing and Hiding Tabs

To display a tab in the Left Pane, Right Pane or Bottom Pane, select *View|Tabs|<TabName>*, or see Komodo's default key bindings (Help|List Key Bindings) for the appropriate keyboard shortcut. Alternatively, click the *Show Specific Tab* button on the Standard Toolbar.

### Showing and Hiding Panes

To show and hide the Left Pane, Right Pane or Bottom Pane, click the associated button on the Standard Toolbar. Click the close arrow in the top right corner of a pane to close it.

### Resizing Panes

When you are focusing on coding alone, you may want to maximize the Editor Pane. To increase the size of the Editor Pane, hide the Left Pane and the Right Pane. The Left, Right and Bottom Panes can be resized by clicking and dragging.

## Enabling Full Screen Mode

To take advantage of as much of the Komodo workspace as possible, select *View|Full Screen*. When Full Screen mode is enabled, the workspace is maximized, and the toolbars and status bar are hidden. To restore the previous view, select *View|Full Screen* again.

# Getting Started with the Sample Project

Komodo's sample project includes a number of programs in different languages. Use these sample programs to familiarize yourself with Komodo's functionality.

## Opening the Sample Project and Files

On Komodo's Start Page, click *Open Sample Project*. The Sample Project and its associated files will display on the Projects tab.

To open a sample program, double−click the file name on the *Projects* tab. The contents of the file will display in the Editor Pane.

## Editing a Sample Program

Komodo includes sample programs written in Perl, Python, JavaScript, PHP, Tcl, and XSLT. Each program is annotated with comments and exercises that describe Komodo's language−specific features. Open the sample programs for languages that interest you, and read the comments to explore Komodo's editing and debugging functionality.

## Debugging a Sample Program

Komodo provides debugging support for Perl, Python, PHP, Tcl and XSLT. Komodo works with the core language distribution for Perl, Python and PHP to provide interpreter support. XSLT, on the other hand, is entirely self−contained. To debug the sample files for Perl, Python, PHP and Tcl, you must configure the location of the language interpreter. See Configuring the Perl Debugger, Configuring the Python Debugger, Configuring the Tcl Debugger, or Debugging PHP for instructions. Then open the sample file for the desired language, and view the comments in the "Debugging" section. General debugging functionality is discussed below.

1. *Breakpoints*: In the sample program, click on the gray margin to the immediate left of the Editor Pane. A green circle will appear, indicating that a breakpoint has been set. When you run the debugger, program processing will stop at lines where breakpoints have been set.
2. *Start / Step Over / Step In*: To start debugging, click the "Go" button on the Debug Toolbar. When debugging begins, the Bottom Pane will be displayed beneath the Editor Pane in the Komodo workspace. The program will run until a breakpoint is encountered; when program execution pauses at a breakpoint, click "Step In" to move through the program in single line increments, or "Step Over" to execute the entire function (as applicable), or "Step Out" to execute the remainder of a function (as applicable).

3. ***Debug Tab***: The tab labelled ***Debug: \<filename\>*** is displayed when debugging begins. In addition to the debug output, the Debug tab displays the call stack, variables, and variable values.

# Working with Projects

Use Komodo's Project Manager to organize Komodo "components". Components are objects that are relevant to a project. For example, a project might contain all the components related to a specific software product: the product's source files, templates for creating new source files, snippets for adding repetitive sections of code, and run commands for building the output. By using projects to organize components, all the aspects of a project can be quickly accessed from a single location.

The top level of organization is the project file, which can contain files, folders and components like run commands and macros. Projects are XML files stored on disk with a ".kpf" extension. While these project files can be manually edited, it is unnecessary; all project maintenance can be performed within the Project Manager. If you manually edit the project file, make a backup copy first; invalid construction within a project file prevents it from loading.

Components within the Project Manager and the Toolbox are manipulated by an intuitive drag–and–drop interface, by context menus, or via the Project and Toolbox drop–down menus. Options for individual components are accessed via the *Project* drop–down menu, or via right–click context menus.

Use Komodo's sample project to become familiar with project functions.

*Feature Showcases*

- create a custom toolbar
- create a directory shortcut
- export and import a package
- assign a custom keybinding to a component
- store a template in a project

## Displaying the Project Manager

The Project Manager is displayed on a "tab" on the Right Pane of the Komodo workspace. It is displayed or hidden via any of the following methods:

- *View/Tabs/Projects*: Use the *View* drop–down menu to modify Komodo's display characteristics.
- *Key Binding*: The default key binding for opening and closing the Projects tab is 'Ctrl'+'Shift'+'P'. See Configuring Key Bindings to alter or view the current assignment.
- *Workspace Toolbar*: If the Workspace Toolbar is displayed, click the "Show/Hide Left Pane" button. (Select *View/Toolbars/Workspace* to display the toolbar.) Alternatively, click the "Show Specific Tab" button and select *Projects*.

## Creating Projects

Projects are created via any of the following methods:

- *File Menu*: On the *File* menu, select *New/New Project*.
- *Project Menu*: On the *Project* menu, click *New Project*.

- *Project Manager: New Project Button*: On the *Projects* tab, click the ![icon] *New Project* button.

When creating a new project, you are prompted to select a directory where the project file is stored. The project filename becomes the project name displayed in the Project Manager (without the ".kpf" extension).

New projects contain no components. See Adding Components to Projects for information about populating projects.

# Opening Projects

Existing projects are opened by any of the following methods:

- *File|Open Menu*: Select *File|Open|Project*.
- *Project Menu*: Select *Project|Open Project*
- *Project Manager*: On the *Projects* tab, click the *Open Project* button.

Recently opened projects can be accessed from the Komodo Start Page, or from the *File|Recent Projects* menu. (The number of recently opened projects is determined by the Most Recently Used preference.)

The project name and associated files are displayed on the Projects tab. Opening a project only opens the project file, not the components associated with the project.

# Setting the Active Project

Multiple projects can be open at the same time. Components in any open project can be used. However, only one project at a time is "active". The active project is displayed in bold text. When the *Add Component* button in the Project Manager is invoked, the component is added to the active project, regardless of the project that is currently highlighted.

To set the active project, right−click the desired project name and select *Make Active Project*, or select *Project|Make Active Project* from the drop−down menu.

# Adding Components to Projects

Projects are organization containers for "components". Components are items like files, folders, snippets, macros, etc. For information about individual components, refer to the relevant section of the documentation for the specific component.

Filesystem−based components consists of items that exist as entities on the disk: files, dialog projects and templates. Komodo−specific components exist only in Komodo, and are not stored as external disk entities. Komodo−specific components include open shortcuts, folders, snippets, run commands, URL shortcuts, macros and custom menus and toolbars.

Within projects, filesytem−based components are merely references; they cannot be moved or deleted from within Komodo; moving them between projects and the Toolbox, or deleting them from projects or the Toolbox, has no effect on their actual disk−based location or their contents. On the other hand, Komodo−specific components (such as macros and snippets) do not exist outside of Komodo; therefore, they can be moved and copied (for example, from the Toolbox to the Project Manager) and deleted independently of the filesystem.

For example, when a snippet (a Komodo−specific component) is copied from the Project Manager to the Toolbox, a new copy of the snippet is created; the original version still exists in the original location. Subsequent modifications of the snippet in the Toolbox do not change the version of the snippet stored in a project. On the other hand, when the contents of a file (a filesystem−based component) are changed, the changes apply regardless of the location from which the file is invoked.

To add a component to a project:

- *Project/Add*: Use the *Project* drop−down menu. If multiple projects are open, the component is added to the project that is currently active.
- *Project/Add/project_name*: When a project name is selected in the Project Manager, the selected project's name is also displayed in the *Project* drop−down menu. Components can be added directly to that project.
- *Add Button*: Use the *Add* ⬐ ▾ button on the *Project* tab. If multiple projects are open, the component will be added to the project that is currently selected.
- *Context Menu*: Right−click a project name and select *Add/component*.
- *Drag and Drop*: Drag and drop components from:
    - ♦ *Drag−and−Drop−Aware Applications*: Filenames can be dragged from drag−and−drop applications (such as Windows Explorer).
    - ♦ *Project Manager or Toolbox*: Components can be dragged between projects, the Toolbox and container components (such as folders).
    - ♦ *Editor File Name Tabs*: When a file is open in the editor pane, drag and drop from the tab that displays the filename to a project.
- *Import from Filesystem*: Right−click a project and select *Import from File System*.
- *Cut/Copy/Paste*: All components have a right−click menu that includes the options *Cut*, *Copy* and *Paste*. Use these options to move components between projects, the Toolbox and container components.

Some components have special component−specific mechanisms for being added to projects. For example, snippets can be created and added to a project in one step by selecting a block of text in the Editor Pane and dragging it onto a project. URL shortcuts are created by dragging and dropping a URL from a browser address bar or from the Editor Pane onto a project. Refer to the specific component sections for more information about these options.

To remove a component from a project, click the desired component and press 'Delete', or right–click the component and select *Delete* from the context menu.

Komodo–specific components (such as run commands, macros, etc) are permanently deleted. Filesystem–based components (such as files and dialog projects) are not; only the reference to the component within the project is removed.

# Project Display Settings

Use the button on the top right of the Project Manager to select the fields that are displayed in the Project Manager. Click the column headings to organize the items according to the contents of any column.

Column contents depend on the type of component. For example, the *Status* and *Depot Rev* display options only have contents if the component is part of a source code control repository.

The following columns can be displayed:

- *Date*: The date from the filesystem when the component was last saved.
- *Size*: The filesystem size of the component.
- *Status*: The source code control status of the component.
- *Rev*: When a component is part of a source code control repository, this column displays the revision number of the local version of the file.
- *Depot Rev*: When a component is part of a source code control repository, this column displays the revision number of the repository version of the file.
- *Action*: When a component is part of a source code control repository, this column displays the current action (if any) being performed on the file. For example, if a file is currently opened for editing, the action is "edit".
- *Name*: The name of the component. For filesystem–based components (such as files and dialog projects), the name is equivalent to the filename. For Komodo–specific components (such as macros and run commands), the name is the user–defined name assigned to the component.

# Saving Projects

If an asterisk is displayed beside the project name, the project has changed since it was opened or last saved. This happens not only when components are added, edited or removed in the Project Pane, but also when debugging options are changed for files in the project. To save a project:

- *File Menu*: On the *File* menu, click *Save Project*.
- *Project Menu*: On the *Project* menu, click *Save Project*. This saves the active project. If a project name is currently highlighted in the Project Manager, the *Project* menu contains an option for saving the selected project (*Project|project_name|Save Project*).
- *Project Manager: Save Project Button*: On the *Projects* tab, click the *Save Project* button.

- *Project Context Menu*: Right–click the desired project and select *Save Project*.

The mechanisms described above can also be used to save a project to a new project file by selecting *Save Project As...* rather than *Save*. Filesystem–based components (such as files and dialog boxes) are relative references rather than actual entities; in the new project, the reference to the location of the component is preserved. Komodo–specific components (such as macros and run commands) are copied to the new project; there are independent versions of the component in the original project and in the new project.

# Refreshing Project Status

The *Refresh Status* option checks read/write disk status for the project file and for filesystem–based components (such as files and dialog projects) within the project. If the project contains files of a language for which "code intelligence" is supported and enabled (as configured in the Code Intelligence Preferences), *Refresh Status* also updates the code intelligence database with the contents of those files.

If the project or its components are stored in a source code control system, *Refresh Status* also checks the repository status of the file. Komodo determines whether a file is contained in an SCC repository by the following methods:

- *Perforce*: analysis of the client configuration
- *CVS*: analysis of the CVS control directories

# Importing and Exporting Projects via Packages

Entire projects (including all the components contained in a project) can be exported to a "package" file for distribution to other Komodo users or for the sake of archiving. Packages are compressed archive files that contain the project from which the *Export Package* option was invoked. The *Export Package* option differs from the *Export as Project File* option (provided for project components) in that *Export Package* creates a self–contained archive file that contains copies of all the filesystem–based components (such as files and dialogs). *Export as Project File*, on the other hand, only contains Komodo–specific components (such as snippets and run commands).

To export a project and its contents to an archive, select *Project|project_name|Export Package* from the drop–down menu, or right–click the project and select *Export Package*.

The Package Export Wizard prompts for a *Package Name* and a *Export Location*. The *Package Name* is the file in which the package is stored; it will have the extension ".kpz", and can be opened by any archiving utility that supports `libz` (for example WinZip).

Exported packages can only be imported into "container" objects in Komodo, such as projects, the Toolbox, and folders within projects and the Toolbox.

To import the contents of a package into a project, right–click the project to which you want to import the package, and click *Import Package*. The Package Import Wizard prompts for the name of the package and the location on disk where the files will be extracted. Click *Next* and then click *Finish* to complete the import.

For information about importing packages to the Toolbox or a folder, see Toolbox – Import Package and Folders – Import Package for more information.

# Importing Files from the File System

This option creates files within a project based on the directory structure and file contents of a local or network filesystem. File references within the project are created for imported files; folders are created for directories (depending on the configuration of the import options). To import a filesystem into a project:

- *Project Menu*: On the *Project* menu, click *Import from File System*. This imports the specified filesystem (according to the criteria described below) into the current active project. If a project name is currently highlighted in the Project Manager, the *Project* menu contains an option for importing into the selected project (*Project/project_name/Import from File System*).
- *Project Context Menu*: Right–click the desired project and select *Import from File System*.

Configure the following import options:

- *Directory to import from*: Specify the directory from which you want to import files. Use the *Browse* button to navigate the file system.
- *Files to include*: Specify the filenames to include. Use wildcards ("*" and "?") to specify groups of files. Separate multiple file specifications with semicolons. If the field is left blank, all files in the specified directory are imported.
- *Files and directories to exclude*: Specify the file and directory names to exclude. Use wildcards ("*" and "?") to specify groups of files. Separate multiple file specifications with semicolons. If the field is left blank, no files in the specified directory are excluded.
- *Import Subdirectories Recursively*: Select this check box to import directories (and files contained in those directories) located beneath the directory specified in the *Directory to import from* field. This check box must be checked in order to specify the "Import Directory Structure" option as the *Type of folder structure to create*.
- *Type of folder structure to create*:
  - ♦ *Import directory structure*: If the *Import Subdirectories Recursively* box is selected and this option is selected, Komodo creates folders within the project that represent imported directories. Thus, the directory structure is preserved within the project.
  - ♦ *Make a folder per language*: If this option is selected, imported files are organized into folders according to the language indicated by file pattern in the filename. File associations are configured in the Komodo Preferences. Each folder is named after the associated language, for example, "Perl files", "XML files", etc. Files that don't correspond to a known file pattern are stored in a folder called "Other files".

♦ *Make one flat list*: If this option is selected, all the imported files are placed directly under the project from which the *Import from File System* command was invoked.

# Source Code Control

Source code control refers to projects and/or components stored in a source code control depot (such as a CVS or Perforce repository). There are two aspects to source code control within projects: source code control on the project file itself, and source code control on components contained in projects.

When Komodo integration with a source code control system is configured, icons in the Project Manager display the SCC status of the project file and components contained in the project. See file status icons in the source code control documentation for more information about these icons.

Source code control functions (such as adding files to an SCC repository, or opening files contained in a repository for editing) can be performed both on the project file, and on the components contained in the project. To access SCC commands for a project file:

- *Project Menu*: Click the project name in the Project Manager, then select *Project|project_name|Source Control*).
- *Project Context Menu*: Right−click the desired project and select *Source Control*.

To access SCC commands for the components contained in a project:

- *Project Menu*: Click the project name in the Project Manager, then select *Project|project_name|Source Control on Contents*).
- *Project Context Menu*: Right−click the desired project and select *Source Control on Contents*.

See Source Code Control Commands for a description of each of the commands.

# Reverting Projects

If a project has been altered but has not been saved, use the *Revert Project* option to undo the changes. To access the *Revert Project* option:

- *Project Context Menu*: Right−click the desired project and select *Revert Project*.
- *Project Menu*: Select *Project|Revert Project* or *Project|project_name|Revert Project*.
  *Project|Revert Project* reverts the active project. If a project name is currently highlighted in the Project Manager, the Project menu contains an option for reverting the selected project (*Project|project_name|Revert Project*).

# Closing Projects

Closing a project removes it from the Project Manager. To close a project:

- *File Menu*: On the *File* menu, click *Close Project*.
- *Project Menu*: On the *Project* menu, click *Close Project*. This closes the active project. If a project name is currently highlighted in the Project Manager, the *Project* menu contains an option for closing the selected project (*Project/project_name/Close Project*).
- *Project Context Menu*: Right−click the desired project and select *Close Project*.

If the project has changed since it was last saved, you are prompted to save it. If files contained in the project are open in the editor, you are asked if you wish to close them.

# Deleting Projects

When a project is deleted, the project file is deleted from disk; Komodo−specific components (such as run commands and macros) stored in the project are also deleted. Filesystem−based components (such as files and dialog projects) are not deleted. To delete a project, delete the project file from the filesystem.

# Project Properties

The Project Properties dialog box contains information about the project file, such as its size, its location on disk, and its source code control status.

To access the Project Properties dialog box, right−click the desired project name and select *Properties*, or click the desired project and select *Project/project_name/Properties* from the drop−down menu.

# Using the Toolbox

The Komodo Toolbox stores components that are frequently used. The Toolbox is similar to a persistent project; however, rather than storing components that are specific to a particular project, it stores components that are used for a variety of tasks.

The Projects Manager and the Toolbox share a common set of components. Components are of two types: Komodo–specific components (such as macros, run commands and custom menus) and filesystem–based components (such as files and dialog projects).

Components within the Project Manager and the Toolbox are manipulated using an intuitive drag–and–drop interface, by context menus, or via the *Project* and *Toolbox* drop–down menus. Component options are accessed via the drop–down menus, or via right–click context menus.

*Feature Showcases*

- create a custom toolbar
- create a directory shortcut
- import a filesystem
- assign a custom keybinding to a component
- store a template in a project

## Displaying the Toolbox

The Toolbox is displayed on a "tab" in the Right Pane of the Komodo workspace. It is displayed or hidden via any of the following methods:

- *View|Tabs|Toolbox*: Use the *View* drop–down menu to modify Komodo's display characteristics.
- *Key Binding*: The default key binding (in the default key binding scheme) for opening and closing the Toolbox tab is 'Ctrl'+'Shift'+'T'. See Configuring Key Bindings to alter or view the current key assignment.
- *Workspace Toolbar*: If the Workspace Toolbar is displayed, click the "Show/Hide Right Pane" button. (Select *View|Toolbars|Workspace* to display the toolbar.) Alternatively, click the "Show Specific Tab" button and select *Toolbox*.

## Adding Components to the Toolbox

The Toolbox is an organizational container for frequently used "components". Components are items like files, folders, snippets, macros, etc. For information about individual components, refer to the relevant section of the documentation.

Filesystem–based components consists of items that exist as entities on the disk: files, dialog projects and templates. Komodo–specific components exist only in Komodo, and are not stored as external disk entities. Komodo–specific components include open shortcuts, folders, snippets, run commands, URL shortcuts, macros and custom menus and toolbars.

Within the toolbox, filesytem–based components are merely references; they cannot be moved or deleted from within Komodo; moving them between projects and the Toolbox, or deleting them from projects or

the Toolbox, has no effect on their actual disk−based location or their contents. On the other hand, Komodo−specific components do not exist outside of Komodo; therefore, they can be moved and copied (for example, from the Toolbox to the Project Manager) and deleted independently of the filesystem.

For example, when a snippet (a Komodo−specific component) is copied from the Project Manager to the Toolbox, a new copy of the snippet is created; the original version still exists in the original location. Subsequent modifications of the snippet in the Toolbox do not change the version of the snippet stored in the Project Manager. On the other hand, when the contents of a file (a filesystem−based component) are changed, the changes apply regardless of the location from which the file is invoked.

To add a component to the Toolbox:

- ***Toolbox|Add***: Use the ***Toolbox*** drop−down menu.
- ***Toolbox|Add|folder_name***: When a folder name is selected in the Toolbox, the selected folder's name is also displayed on the ***Toolbox*** drop−down menu. Components can be added directly to that folder.
- ***Add Button***: Use the ***Add***  button on the ***Toolbox*** tab.
- ***Drag and Drop***: Drag and drop components from:
    - ♦ ***Drag−and−Drop−Aware Applications***: Filenames can be dragged from drag−and−drop applications (such as Windows Explorer).
    - ♦ ***Project Manager or Toolbox***: Components can be dragged between projects, the Toolbox and container components (such as folders).
    - ♦ ***Editor Filename Tabs***: When a file is open in the Editor Pane, drag and drop from the tab that displays the filename to the Toolbox.
- ***Import from Filesystem***: Right−click a folder and select ***Import from File System***.
- ***Cut/Copy/Paste***: All components have a right−click menu that includes the options ***Cut***, ***Copy*** and ***Paste***. Use these options to move components between projects, the Toolbox and container components (such as folders).

Some components have special component−specific mechanisms for being added to projects. For example, snippets can be created and added to the Toolbox in one step by selecting a block of text in the Editor Pane and dragging it into the Toolbox. URL shortcuts are created by dragging and dropping a URL from a browser address bar or from the Editor Pane into the Toolbox. Refer to the specific component sections for more information about these options.

To remove a component from the Toolbox, right−click the desired item and select ***Delete***. Alternatively, click the desired component and press 'Delete', or right−click the component and select ***Delete***.

Komodo−specific components (run commands, macros, etc) are permanently deleted. Filesystem−based components (files, dialog boxes) are not; only the reference to the component within the project is removed.

# Exporting and Importing Toolbox Contents

The Toolbox has commands for exporting and importing the contents of the Toolbox as a package. Packages, which are stored with a ".kpz" extension, are compressed archive files, and can be opened by any archiving utility that supports `libz` (for example WinZip). The ***Export Package*** option differs from the ***Export as Project File*** option in that copies of filesystem–based components (such as files and dialog projects) are included in the archive. Conversely, ***Export as Project File*** creates a project with a reference to the component's original location and does not create copies of the components.

Both Komodo–specific components (such as run commands and snippets) and filesystem–based components (such as files and dialog projects) can be included. The exported file can subsequently be loaded as a project or imported into the Toolbox. This provides a mechanism for sharing a Toolbox among multiple Komodo users; another mechanism is described in the "Shared Toolbox" section below.

To export the contents of the Toolbox to a project file:

1. From the ***Toolbox*** menu, select ***Export***.
2. In the ***Export Package*** dialog box, choose the destination directory and enter a filename.
3. Click ***Save***.

To import a previously exported Toolbox:

1. From the ***Toolbox*** menu, select ***Import***.
2. In the ***Import Package*** dialog box, select the project file containing the item(s) that you want to import.
3. Click ***Open***.

# Sharing Toolboxes

A Shared Toolbox is a single Toolbox shared by multiple Komodo users. When a Shared Toolbox is enabled, a second tab called ***Shared Toolbox*** is displayed beside the ***Toolbox*** tab in the Right Pane. This tab displays the contents of the `toolbox.kpf` file, located in the Common Data Directory specified in Komodo's Shared Toolbox preference.

The functionality of the Shared Toolbox is determined by the user's access rights to the `toolbox.kpf` file. All users with "read" rights to the file are able to use items from the Toolbox. To prevent users from altering the shared toolbox, ensure that they only have "read" (and not "write") access to the file.

The Shared Toolbox is populated by the same methods described in Adding Components to the Toolbox.

# Working with Files

Komodo provides a variety of methods for accessing and editing files. While files can be opened and edited individually, they can also be stored in projects or the Toolbox as components.

If Komodo is configured to integrate with a source code control system (SCC), status icons beside the filenames indicate the file's current SCC status, and SCC options are available from the *File* menu and the right−click context menus. This integration is described in detail in the Source Code Control section of the Komodo documentation.

Files are manipulated in various ways: via the *File* menu, via context menus in the editor, via context menus on the file tab (above the editor), and as components within projects and the Toolbox.

This document describes file functionality, such as opening, printing, and saving files. See the Editing page for information about editing files.

## Creating Files

To create a new file, click the "New File" button on the standard toolbar. (To display the standard toolbar, click *View*/*Toolbars*/*Standard*.) A new file with the default file extension for the file type is created and opened in the Komodo editor. Use the New Files page in Komodo's Preferences to specify the default file extension for new files.

### Creating Files from Templates

New files can be created based on pre−defined templates that contain default content for specific file types. See the Templates documentation for information about configuring custom templates.

To create a new file from a template, select *File*/*New*/*File*. The New File dialog box contains numerous pre−defined templates organized into categories. Select the desired category, then select the desired template within that category.

Click *Open* to create a file with the contents of the template file. The file is loaded in the editor.

The *File*/*New*/*File* menu displays a list of the most recently accessed templates. To alter the number of template files displayed, refer to the New Files page in Komodo's Preferences.

To add a template to the Toolbox for quick access, select the desired template and click *Add to Toolbox*.

# Storing Files within a Project or the Toolbox

Files can be stored in a project, in the Toolbox, or within folders in a project or the Toolbox. Storing a file within a project or the Toolbox creates a reference to the actual file located on the disk; it does not affect the file itself. Therefore, the same file can be stored in multiple projects and folders, both in projects and within the Toolbox; changes to the original source file apply regardless of the source of the file reference; deleting a file from a project or the Toolbox does not delete the file from the filesystem.

While most file options (such as source code control commands, viewing unsaved changes, and refreshing the file status) are accessible regardless or whether or not the file is contained in a project or the Toolbox, some commands are specific to the project/Toolbox environment. These commands are described below.

## Creating an Open Shortcut to the Current File Location

Open... shortcuts, stored within a project or the Toolbox, are references to filesystem directories. When an open shortcut is invoked, the standard Open File dialog box is displayed with the contents of the directory.

Open shortcuts can be created using a specific file as the context. Right−click a file in a project or the Toolbox and select *Make "Open..." Shortcut*. This creates an open shortcut to the directory where the file is stored.

## Exporting Files as Projects

The *Export as Project File* option is used to create a new project file containing the file from which the option was invoked. To access this option, the file must be stored in a project or the Toolbox. Right−click the desired file name and select *Export as Project File*. You are prompted to provide the name of the new project file and the directory where it will be stored.

To open the new project file, select *File|Open|Project*.

## Exporting Files to a Package

Files can be archived and distributed among multiple Komodo users via "packages". Packages are compressed archive files that contain the file from which the *Export Package* option was invoked. Packages are stored in files with a ".kpz" extension, and can be opened by any archiving utility that supports `libz` (for example WinZip). The *Export Package* option differs from the *Export as Project File* option in that copies of filesystem−based components (such as files and dialog projects) are included

in the archive. Conversely, *Export as Project File* creates a project with a reference to the component's original location and does not create copies of the components. When *Export Package* is invoked, you are prompted for a name and file location for the package.

Exported packages can only be imported into "container" objects in Komodo, such as projects, the Toolbox, and folders within projects and the Toolbox. See Toolbox – Exporting and Importing Toolbox Contents, Projects – Importing and Exporting Projects via Packages, or Folders – Import Contents from Package for more information.

# Opening Files

There are numerous methods for opening files in Komodo. These include:

- *Command–Line Argument*: When Komodo is invoked from the command line, files can be specified as open arguments. See Starting on Windows or Starting on Unix for more information.
- *File/Open*: Use the *File/Open* menu option.
- *Project Manager or Toolbox*: Double–click, drag and drop, or use the file's right–click context menu to open a file contained in a project or the Toolbox.
- *Open/Find Toolbar*: Use the Open/Find Toolbar. To display the Open/Find Toolbar, select *View/Toolbars/Open/Find*.
- *Most Recently Used List*: The most recently opened files are displayed on Komodo's Start Page. This list is also accessible from the *File/Recent Files* menu. The number of files in the most recently used list is determined by the Appearance preference.
- *Drag and Drop*: Drag and drop one or more files from another drag–and–drop away application (such as Windows Explorer) onto the Komodo editor.

## Opening Files with the Open/Find Toolbar

The Open/Find Toolbar provides quick access for opening files and finding strings. The toolbar is displayed in Komodo by default. To hide or show the toolbar, select *View/Toolbars/Open/Find*.

Use the *Open* field on the *Open/Find Toolbar* to open one or more files in the filesystem. The *Open* field generates a drop–down list of files and directories as you navigate the hierarchy. To open a file, enter the file path and name, and then press 'Enter'.

For example, to open a file named "debug.txt" in the directory *C:\temp\log* on a machine running Windows, enter *C:\tmp\log\debug.txt* in the *Open* field and then press 'Enter'. As you enter each backslash, a drop–down list displays the files and directories beneath the current directory.

Enter "./" or ".\" in the *Open* field to display a list of files and directories in the "current" directory. The current directory is determined as follows:

1. ***Current File***: If a file is open in the editor, the directory where the file is stored is the current directory. If multiple files are open, the file that is currently displayed is the current file.
2. ***HOME Variable***: If no files are open, Komodo checks if the system has a defined `HOME` environment variable. If so, the directory specified in that variable's value is the current directory.
3. ***Filesystem Root***: If neither of the above conditions is met, the system's root directory is used ("C:\" is the default on Windows, and "/" is the default on Unix).

To narrow the results in the list box, enter one or more characters in the name of the file or directory you wish to open or view. (The search is case−sensitive.) Alternatively, use the arrow keys to navigate the drop−down list. The contents of the ***Open*** field are updated as you move up and down the list with the arrow keys.

Note that you can use standard directory navigation syntax to change directories. For example, if the current directory is */home/fred/tmp/foo*, change to the */home/fred/tmp/bar* directory by entering *../bar*.

To open a file, press 'Enter' when the file name is displayed in the ***Open*** field. To continue navigating the directory structure, append a frontslash or backslash to the directory name displayed in the ***Open*** field; the drop−down list is updated with the list of files and directories that exist under the current location.

To open multiple files, specify a wildcard ("*" for a file name segment, "?" for a specific character). For example, to open all the files with the extension ".tcl" in the directory */home/fred/tmp*, enter */home/fred/tmp/*.tcl*.

At any time, press the 'Escape' key to return focus to the Komodo editor.

## Opening Remote Files

Komodo can open files located on remote machines, providing that the remote machine is configured for FTP access. To quickly access frequently used FTP servers, create an entry in the Server Preferences (***Edit/Preferences/Servers***).

To open a file located on a remote FTP server, select ***File/Open/Remote File***.

### Connecting to an FTP Server

- ***Pre−Configured Server Connection***: If FTP servers have been configured in Komodo's Preferences, select the name of the configuration from the ***Server*** drop−down list. Access the Server Configuration dialog box by clicking the ***Accounts*** button to the right of the ***Server*** field.
- ***Manual Server Connection***: Enter the FTP address (in the format "ftp.server.com") in the ***Server*** field. Press 'Enter'. You are prompted to enter a name and password for the FTP server. If the server is configured for anonymous access, select ***Anonymous login***. To store the login name

and password for the server, click **Remember these values**.

## Navigating the Remote File System

After establishing a connection to the remote FTP server, a list of files and directories is displayed. These files and directories exist under the directory specified in the **Look in** field. Double−click a directory (indicated by a file folder icon) to navigate the directory structure. Use the navigation buttons in the top right corner of the dialog box to navigate the remote filesystem.

To open a single file, double−click the filename. To open multiple files, hold down the 'Ctrl' key while clicking multiple files, then click **Open**.

The buttons in the top right corner of the dialog box perform various file and directory manipulation functions. Hover your mouse pointer over the buttons for a description of their functions. To delete or rename a file on the remote server, right−click the filename and, with the left mouse button, select the desired command on the context menu.

## Storing Remote Files in Projects or the Toolbox

When remote files are added to a project or the Toolbox, or to container components within a project or the Toolbox, they are accessed and organized in the same manner as local files (double−clicking a file opens it in the editor; right−clicking a file accesses file option, etc). Remote files can only be added to projects or the Toolbox by connecting to the remote server.

Login authentication is processed when the remote file is opened, or when it is added to the Toolbox or a project. For the duration of the Komodo session, the user name and password are remembered; closing and opening the file uses the same authentication information as the first time the file was opened. If authentication fails, you are prompted to enter a user name and password.

After closing and re−opening Komodo, the authentication information stored in the Servers preference is checked again. Thus, changes to the user name or password stored in the server configuration do not take effect until Komodo is closed and re−opened. If no authentication information is stored in the server configuration, you are prompted to supply it when you open the file.

See Adding Components to Projects or Adding Components to the Toolbox for instructions on adding remote files to the Toolbox or to projects.

To open a remote file, double−click the file name, use the assigned key binding, or right−click the file and select **Open File**.

# Switching Between Files

To switch between open files in the editor:

- *Key Binding*: Use the associated key binding.
- *Editor Tabs*: Click the tab with the desired filename.
- *Window Menu*: On the *Window* menu, select *Next File* or *Previous File* to move from left to right (or right to left) across the file tabs. Alternatively, select the desired file from the list of files currently open in the editor.
- *Project or Toolbox*: Double−click the filename.

If more files are opened than can be displayed by file tabs, click the right and left arrow buttons located in the top right corner of the editor to view the tabs of all open files.

To re−order the position of the file tabs, drag and drop the tabs into the desired positions.

For more information about working with the editor tabs, see Editor Tab Display in the editor documentation.

# Comparing Files

Komodo includes a "diff" mechanism used to compare files. To compare two files using Komodo's "diff" window:

1. Select *Tools/Compare Files*.
2. By default, the path and file name of the file currently displayed in the editor is the first file for comparison. As desired, alter this selection by entering an alternate path and file, or browse for the desired file using *Browse* button. Use the same mechanism to specify the second file.
3. Click *Compare Files*. The contents of both files are displayed in the "diff" window.

If the file is stored in a project or the Toolbox, this function can also be invoked by right−clicking the file and selecting *Compare File With*.

The unique characteristics of each file are displayed in different colors (red and blue by default); common characteristics are displayed in a third color (black by default). To configure custom colors for the "diff" window, alter the *Language−Specific Coloring* setting for the *Other/Diff* language in the Fonts and Colors preference.

The following key bindings are available in the "diff" window:

- *F8*: Jump to the next change.
- *F7*: Jump to the previous change.
- *F9*: Jump to corresponding line. Opens and/or shifts focus to the original file in the Editor Pane. If viewing a diff in an editor tab, right−click and select *Jump to Corresponding Line* (or select

*Code|Jump to Corresponding Line*) to shift focus to the editor tab containing the source code. Selecting this option opens the source code tab in the Editor Pane if it is not already open and/or shifts focus to the original file in the Editor Pane. (If viewing a diff in an editor tab, right−click and select *Jump to Corresponding Line*.)
- *Esc*: close the window

# Refreshing File Status

The *Refresh Status* option checks the read/write disk status for the component. If the file is of a language for which "code intelligence" is supported and enabled (as configured in the Code Intelligence Preferences), *Refresh Status* also updates the code intelligence database with the contents of the file.

If the component is stored in a source code control system, *Refresh Status* also checks the repository status of the file. Komodo determines whether a file is contained in an SCC repository by the following methods:

- *Perforce*: analysis of the client configuration
- *CVS*: analysis of the CVS control directories

To refresh the file status of the current file, right−click the file tab or right−click within the editor and select *Refresh Status*. The same option is available on the right−click context menu of files in projects or within the Toolbox.

# Source Code Control

Komodo provides source code control support for files stored in CVS or Perforce repositories. Source code control support (including SCC configuration, status icons and specific commands) is described in detail in the Source Code Control section of the documentation. To access source code control commands:

- *Editor Context Menu*: Right−click a file in the editor and select *Source Control*.
- *File Tab Context Menu*: Right−click a file tab above the editor and select *Source Control*.
- *Toolbox or Project Context Menu*: Right−click a file in the Toolbox or Project Manager and select *Source Control*.
- *Toolbox or Project Menu*: If a file is currently selected in the Toolbox or Project Manager, use the menu to access source code control commands for the selected file.

# File Properties and Settings

In addition to the Komodo's default preferences, some preferences can also be configured on a per–file basis. These settings override the default preferences. To access the Properties and Settings dialog box for a file, do one of the following:

- *Edit Menu*: On the *Edit* menu, click *Current File Settings*.
- *Editor Context Menu*: Right–click in the editor and select *Properties and Settings* from the context menu.
- *File Tab Context Menu*: Right–click the tab above the editor that displays the filename, and select *Properties and Settings*.

## Properties Tab

The *Properties* tab in the Properties and Settings dialog box displays general information about the file, such as the directory where it is stored, the size and creation and modification dates. The following file characteristics can be modified on this tab:

- *Attributes*: Toggle the file's status between writable and read–only.
- *Language*: This field displays the current language association (which affects language–specific options like syntax coloring and AutoComplete) for the current file. To change the language association, select another language from the drop–down list. To set the language association to the Komodo default (as configured in the File Association Preference, click *Reset*.
- *Encoding*: Use this field to set the International Encoding for the current file.
- *Line Endings*: Use this field to set the desired line endings for the current file. By default, Komodo preserves the line endings contained in a file when the file is opened. (Default line endings for new files are configured in the New Files preference.) If you select *Preserve existing line endings*, new lines are assigned the end–of–line character selected in the drop–down list, but existing lines are not be altered.

## Source Control Tab

If Komodo is configured to work in conjunction with a Source Code Control system, the *Source Code Control* tab displays the current SCC status and settings.

## Editing Tab

The options on this tab are a subset of the General Editor, Smart Editing and Code Intelligence preferences. Refer to those sections of the Preferences documentation for information about individual

options.

## Indentation Tab

The options on this tab are a subset of the Indentation Preferences. Refer to that section of the Preferences documentation for information about individual options.

## Preview Tab

This option configures the behavior of the Preview in Browser function. When the Preview in Browser function is invoked, you are prompted to specify the file or URL used to preview the current file. (For example, when previewing a CSS (cascading style sheet) file, specify an HTML file to use for the preview.) The Preview in Browser dialog box has an option for remembering the specification. If that option is enabled, the specification is displayed in the *Preview* tab on the Properties and Settings dialog box. Click *Change* to alter the preview file.

# Printing Files

To print the file that is currently displayed in the editor, use one of the following methods. These methods invoke the standard system dialog box for printer selection and configuration. Advanced print functions are described below.

- *File|Print|Print*: Invoke the print function from the *File* menu.
- *Standard Toolbar*: On the Standard Toolbar, click the Print button.
- *Editor Context Menu*: Right−click the file and select *Print*.

Printing style is configured on the Printing page in Komodo's Preferences. Alternatively, select *File|Print|Print Settings* to display the Printing preferences page.

To display a preview of the printed output, select *File|Print|Print Preview*.

Select *File|Print|Print Preview* contains features for setting the scale and orientation of a print job. Use the arrow buttons to move forward or backward in a multi−page print job, or enter a specific page number in the field provided. Click the Page Setup button to access the complete set of print features in the Page Setup dialog box.

To print a selection of text rather than the entire file, select the desired text in the editor, then select *File|Print|Print Selected Text*.

## Page Setup

Manage the format of print jobs using the options available in the Page Setup dialog box. Select *File|Page Setup* to access these options.

*Format and Options*

- *Orientation*: Select whether the printed output should have a portrait or landscape orientation.
- *Scale*: If the *Shrink To Fit Page Width* check box is not selected, use this field to manually enter a percentage.
- *Shrink To Fit Page Width*: Select this check box to make the print job fit the paper size selected for the default printer.
- *Print Background (colors & images)*: Select this check box to include background colors and graphics (e.g., on a web page) in a print job.

*Margins and Header/Footer*

- *Margins*: Use the fields provided to enter the size of the margins in inches.
- *Headers and Footers*: Use the drop−down lists to select the type of information that appears in the headers and/or footers, and to determine their position on the page. The top row of lists contains the header options, and the bottom row contains the footer options. Choose from options such as "Title", "URL" and "Page #". Select the "Custom" option from any of the drop−down lists to enter custom header information. To print without headers and footers, select the "blank" option in each of the drop−down lists.

## Print to HTML File

To generate an HTML file from the file currently active in the editor:

1. On the File menu, click *Print to HTML File*. You are prompted to name the output file.
2. Enter the file location in the field provided. Click *OK*. The HTML file opens in the editor.

To print a selection of text to an HTML file (rather than the entire file), select the desired text in the editor, then select *File|Print|Print to HTML File*.

# Saving Files

Komodo is "intelligent" about saving files. For example, Komodo prompts to save unsaved files on close. Attempting to save changes to a file that is set to read−only displays a dialog box where you are given the option to change the status or to "force" the save (which makes the file writable, saves the changes, then sets the file back to read−only). In addition, Komodo can be configured to automatically save a backup copy of files open in the editor. To configure Komodo's save functionality, use the Save Options

preference page.

To save a file with its current name, do one of the following:

- *Key Binding*: Use the associated key binding.
- *File/Save*: Use the *File/Save* menu option to save the file that is currently displayed in the editor. To save the file to a different name, select *File/Save As*. To save all open files, select *File/Save All*
- *Standard Toolbar*: Click the Save button on the Standard toolbar. To save all open files, click the Save All button.
- *File Tab Context Menu*: Right−click the file tab and select *Save*. To save the file to a different name, select *File/Save As*.

## Saving Files Remotely

To save a copy of the current file to a remote FTP server, select *File/Save Remotely As*. The Remote File dialog box is displayed. When editing files located on a remote server (including remote files stored in a project or the Toolbox), saving the file automatically saves it to the remote location.

## Show Unsaved Changes

Before saving a file, view the changes in the file since it was last saved by using the *Show Unsaved Changes* option. To invoke this option, right−click within the editor (or on the file tab above the editor) and select *Show Unsaved Changes*. An external window displays the differences between the current version of the file and the disk version (e.g., the version that was last saved).

The unique characteristics of each file are displayed in different colors (red and blue by default); common characteristics are displayed in a third color (black by default). To configure custom colors for the "diff" window, alter the *Language−Specific Coloring* setting for the *Other/Diff* language in the Fonts and Colors preference.

# Reverting Files

To abandon changes made to a file since it was last saved, but leave the file open in the editor, select *File/Revert*.

# Closing Files

To close one or more files, use one of the following methods:

- *File Menu*: Select *File|Close* or *Window|Close*. To close all open files, select *Windows|Close All*.
- *Key Binding*: Use the associated key binding.
- *File Tab Context Menu*: Right−click the file tab and select *Close*.
- *Editor*: Click the "x" in the top right corner of the editor to close the current file.

# Searching

Komodo provides a variety of methods for searching and replacing text, both within files open in the editor and within files on the filesystem. While Komodo has standard search and replace functionality, it also features unique search mechanisms like searching for the work under the cursor, incremental searching and function search.

## Searching for Strings

### Searching Within Open Files: Find Dialog

The Find dialog box is used to search for words or phrases in the current document. To open the Find dialog box, from the *Edit* menu, select *Find*, or use the associated [key binding](#).

Enter the text you wish to find in the *Find what* field.

The following search options can be configured:

- *Match Case*: To find matches of the search string regardless of case, select *No*. To find matches only when the search string matches the case of the occurrence, select *Yes*. To find exact case matches only when the search string contains mixed case, select *Yes, if search string contains capital letters*. For example, if you enter "mystring" in the *Find what* field and select *Yes, if search string contains capital letters*, both "mystring" and "myString" will match. However, if you enter "myString" in the *Find what* field, only "myString" will match.
- *Use*: Choose *Plain Text* to exactly match the search string. *Regular Expressions* will interpret the search string as a Python regular expression, and perform the search accordingly; *Wildcards* will interpret asterisk and question mark characters as wildcards. The option specified (Plain Text, Regular Expressions (Python), Wildcards) becomes the default search type used for the [Open/Find Toolbar](#).
- *Match whole word*: If this box is checked, matches in the document will only be found if whitespace occurs on either side of the search string. For example, if the search string is "edit" and "Match whole word" is selected, only occurrences of the word "edit" will be found as matches. However, if "Match whole word" is not selected, "editing", "editor", "editorial" and "edited" will also be found as matches.
- *Search up*: This performs the search from the cursor position to the top of the file, rather than from the cursor position to the bottom of the file. (The default functionality is to search down from the cursor position.)
- *Display results in Find Results 2*: Komodo supports two *Find Results* tabs in the [Bottom Pane](#). Check this box to display the search results in the second, rather than the first, *Find Results* tab. If this box is not checked, the first *Find Results* tab is used to display the results of the search.

The Open/Find toolbar uses the settings from the Find in Files and Find dialog boxes as default search parameters. For example, if you configure the *Find* dialog box to search using wildcards, the *Open/Find* toolbar will also perform wildcard searches.

Specify where Komodo should search for the text. In the "Search in" section, select one of the following:

- *Current document*: This searches the document that is currently in focus in the editor for occurrences of the search string.
- *Selection only*: will only search the highlighted area of the document that is in focus in the editor. (If you search a selected section, the highlighting of the selected section will be temporarily turned off in order to display highlighted results of the Find function.)
- *All open documents*: Search for the string in all documents currently open in the editor.

Invoke the search by clicking the desired search command button:

- *Find Next*: Finds consecutive occurrences of the search string in your file or selection. As matches are found, the text will be highlighted. The Find dialog box remains in focus. To keep the focus in your file, close the Find dialog box, then use the associated key binding (by default, press 'F3' to Find Next, or 'Shift'+'F3' to Find Previous).
- *Find All*: Locates all occurrences of the search string in your file or selection. The matches will be displayed in the Find Results tab in the Bottom Pane.
- *Mark All*: Inserts a bookmark on each line that contains the search string. To move the editing cursor to a bookmarked line, press *F2*.

To display the results of a previous search, click the relevant *Find Results* tab in the Bottom Pane. If the Bottom Pane is not displayed, select *View/Tabs/Find Results*.


## Replacing Within Open Files: Replace Dialog

The Replace dialog box is used to search for and replace words or phrases in the current document. To open the Replace dialog box, from the *Edit* menu, select *Replace*, or use the associated key binding.

Enter what to find and replace with:

- *Find what*: Enter the search string you want to find.
- *Replace with*: Enter the replacement characters.

The following replace options can be configured:

- *Match Case*: To find matches of the search string regardless of case, select *Never*. To find matches only when the search string matches the case of the occurrence, select *Always*. To find exact case matches only when the search string contains upper–case letters, select "If Search String Contains Capital Letters". For example, if you enter "function" in the *Find what* field and select "If Search String Contains Capital Letters", both "function" and "Function" will match.

However, if you enter "Function" in the *Find what* field, only "Function" will match.
- *Use*: `Plain Text` will exactly match the search string; *Regular Expressions* will interpret the search string as a Python regular expression, and perform the search accordingly; *Wildcards* will interpret asterisk and question mark characters as wildcards.
- *Match whole word*: If this box is checked, matches in the document will only be found if whitespace occurs on either side of the search string. For example, if the search string is "edit" and "Match whole word" is selected, only occurrences of the word "edit" will be found as matches. However, if "Match whole word" is not selected, "editing", "editor", "editorial" and "edited" will also be found as matches.
- *Search up*: Performs the search from the cursor position to the top of the file, rather than from the cursor position to the bottom of the file.
- *Show "Replace All" Results*: Displays the number of replacements in the status bar at the bottom left of the Komodo Workspace. Details of each change, complete with the line number, will appear on the *Find Results* tab in the Bottom Pane.
- *Display in Find Results 2*: Komodo supports two *Find Results* tabs in the Bottom Pane. Check this box to display the search results in the second, rather than the first, *Find Results* tab. If this box is not checked, the first *Find Results* tab is used to display the results of the search.

Specify where Komodo should replace the specified text. In the "Replace in" section, select one of the following:

- *Current document*: Searches the document that is currently in focus in the editor for occurrences of the search string.
- *Selection only*: Only searches the highlighted area of the document that is in focus in the editor. (If you search a selected section, the highlighting of the selected section will be temporarily turned off in order to display highlighted results of the Find function.)
- *All open documents*: Searches for the string in all documents currently open in the editor.

Invoke the search by clicking the desired search command button:

- *Find Next*: Finds consecutive occurrences of the search string in your file or selection. As matches are found, the text is highlighted. Click *Replace* to replace the highlighted text with the replacement string.
- *Replace*: Highlights the next occurrence of the search string; if you click *Replace* again, the highlighted text will be replaced with the replacement string and the next occurrence of the search string will be highlighted.
- *Replace All*: Replaces all occurrences of the search string in the document or selection without prompting for confirmation. All replacements will be displayed on the Find Results tab of the Bottom Pane.

## Searching for the Word Under the Cursor

When the editing cursor is within (or adjacent to) a word, you can quickly search for other occurrences of the same word within the current document. If you are using the default key binding scheme, press

'Ctrl'+'F3' to select the word; continue pressing 'Ctrl'+'F3' to step through each occurrence in the document.


## Incremental Search

Incremental search is used to look through the current file in the Editor Pane for a group of incrementing characters. That is, as you continue to type in search characters, Komodo will find the next occurrence of the search string. After all the search characters have been entered, you can move through each occurrence of the search string within the current file.

*To start an incremental search* select *Edit|Incremental Search*, or use the associated key binding. (If you are using the default key binding scheme, the key binding is 'Ctrl'+'I'.) The status bar (in the bottom left corner of the Komodo workspace) will display the text "Incremental Search:". Begin typing the characters you want to find; as you type characters, the editing cursor will move to the first match beneath the current cursor position within the current file, and the search string will be displayed in the status bar.

*To change the search string* based on the characters surrounding the editing cursor, use the associated key binding. If the default key binding scheme is in effect, the key combination is 'Shift'+'Right Arrow' (to add one or more characters to the right of the editing cursor) or 'Shift'+'Left Arrow' (to remove one or more characters to the left of the editing cursor).

For example, if you entered "fo" as the search string, and the next occurrence of these characters was in the word "foo", you could use the 'Shift'+'Right Arrow' key combination to extend the search string to "foo". Conversely, you could use the 'Shift'+'Left Arrow' key combination to reduce the search string to "f".

*To search through the file for the search string* press 'Ctrl'+'I' to find subsequent occurrences of the search string within the current file. Continue pressing 'Ctrl'+'I' to cycle through all occurrences. When the search reaches the bottom of the file, it will continue from the top of the file. 'Shift'+'Ctrl'+'I' will search backwards from the current cursor positon.

*To cancel the incremental search* press any key except the key bindings assigned to the incremental search functions.


## Searching All Files: Find in Files Dialog

The Find in Files dialog box complements the Find dialog box by providing the ability to search for text in files that are not currently opened in Komodo. Select *Edit|Find in Files* (or use the associated key binding) to open the new dialog box.

The Find in Files dialog box consists of the following options:

- *Find what*: Enter the search string.
- *Match Case*: To find matches of the search string regardless of case, select *No*. To find matches only when the search string matches the case of the occurrence, select *Yes*. To find exact case matches only when the search string contains mixed case, select *Yes, if search string contains capital letters*. For example, if you enter "mystring" in the *Find what* field and select *Yes, if search string contains capital letters*, both "mystring" and "myString" will match. However, if you enter "myString" in the *Find what* field, only "myString" will match.
- *Match whole word*: If this box is checked, matches in the document will only be found if whitespace occurs on either side of the search string. For example, if the search string is "edit" and "Match whole word" is selected, only occurrences of the word "edit" will be found as matches. However, if "Match whole word" is not selected, "editing", "editor", "editorial" and "edited" will also be found as matches.
- *Use*: Choose `Plain Text` to exactly match the search string. *Regular Expressions* will interpret the search string as a Python regular expression and perform the search accordingly; *Wildcards* interpret asterisk and question mark characters as wildcards.
- *Search in*: By default, the *Search in* field is populated with a period, which indicates that the "current" directory will be searched. (If no file is open, the current directory is the value specified in the `HOME` environment variable; if the `HOME` variable is not defined, the current directory is "C:\" on Windows and "/" on Unix platforms. If one or more files are open, the location of the file that is displayed in the Editor Pane is the current directory.) To search directories other than the current, either specify an absolute path, or specify a relative path from the current directory. For example, if the current directory is */home/fred/tmp/foo*, you could search the */home/fred/tmp/bar* directory by entering *../bar*. Manually enter one or more directories; multiple directories are separated by semicolons. Alternatively, click the browse button to the right of the field to display a directory browser dialog box. (Directories already specified in the *Search in* field will be displayed in this dialog box.) Use the directory browser to navigate the filesystem, specify one or more search directories, and/or alter the order in which the directories are searched.
- *Search in subfolders*: If this box is checked, subdirectories beneath the directories specified in the *Search in* field are also searched.
- *Include*: To specify one or more file types that should be searched, enter the file extensions preceded by a wildcard. For example, to search files with ".pl" and ".tcl" extensions, enter `*.pl;*.tcl`. (Note that multiple file extensions are separated by semicolons.) If this field is blank, all file types are searched.
- *Exclude*: To specify one or more file types that should be excluded from the search, enter the file extensions preceded by a wildcard. For example, to exclude files with ".exe" and ".doc" extensions, enter `*.exe;*.doc`. (Note that multiple file extensions are separated by semicolons.) If this field is blank, all file types are searched.
- *Display results in Find Results 2*: Komodo supports two *Find Results* tabs in the Bottom Pane. Select this check box to display the search results in the second, rather than the first, *Find Results* tab. If this box is not checked, the first *Find Results* tab is used to display the results of the search.

After clicking *Find All*, the *Find Results 1* or *Find Results 2* tab (depending on the setting of the *Display results in Find Results 2* check box) are displayed in Bottom Pane. Depending on the number of files that are being searched, it may take some time to generate results. (The search status is displayed on the top line of the *Find Results* tab. The results include the file in which the search string is found, the

line number in the file where the search string occurs, and the context surrounding the search string. Double–click a search result (or click the arrow button at the top right of the pane) to open the file in the editor and place the editing cursor on the selected occurrence.

## Fast Search: Open/Find Toolbar

The Open/Find toolbar provides quick access for opening files and finding strings. Find strings in files currently displayed in the editor or in files not currently open in Komodo but located on the filesystem. The toolbar is displayed by default; to close or open the toolbar, select *View|Toolbars|Open/Find*.

The *Open/Find* toolbar also provides fast access to find functionality. The *Find* and *in* fields are used to perform searches on files in Komodo or elsewhere in the filesystem. Both fields have a list of the most recently entered strings and file specifications. Use the *in* field browse button, located to the right of the *in* field, to populate this field with the directory, folders, or files you want to search.

Enter the string that you wish to search in the *Find* field. The *Find* toolbar uses the settings from the Find in Files and Find dialog boxes as default search parameters. For example, if you configure the *Find* dialog box to search using wildcards, the *Find* toolbar will also perform wildcard searches.

Specify the files that should be searched in the *in* field. The *in* field uses the same logic as the *Open* field to determine the current directory. (If no file is open, the current directory is the value specified in the HOME environment variable; if the HOME variable is not defined, the current directory is "C:\" on Windows and "/" on Linux and Solaris. If one or more files are open, the location of the file that is displayed in the Editor Pane is the current directory.)

To search directories other than the current, either specify an absolute path, or specify a relative path from the current directory. For example, if the current directory is */home/fred/tmp/foo*, you could search the */home/fred/tmp/bar* directory by entering *../bar*. Alternatively, use the *in* field browse button to locate and populate this field with the directory, folders, or files you want to search.

The *in* field accepts wildcards; use "*" for a file name segment and "?" for a specific character. Separate multiple directories with semicolons. If nothing is specified in the *in* field, the search will be performed against the file that is currently displayed in the Editor Pane (if applicable), and the next occurrence of the search string is highlighted.

For example, to search for occurrences of the string "error" in all files located in the directory */tmp/output*, enter *error* in the *Find* field and */tmp/output/\** in the *in* field.

If files are specified in the *in* field, matches will be displayed in the *Find Results tab*.

At any time, press the *Escape* key to return focus to the Komodo editor. The *Find* field can be accessed via the associated key binding.

For example, to search for occurrences of the string "debug" in all files located in the directory */tmp/log*

on a machine running Unix, enter *debug* in the *Find* field. Enter */tmp/log/\** in the *in* field, or use the "browse" button to locate this directory on your filesystem. For more information on searching using the *Open/Find Toolbar*, see Finding Strings.

## Find Results Tabs

The *Find Results 1* tab (located in the Bottom Pane) displays all matches that result when the *Find All* function in the Find dialog box is used. The Find Results 1 tab displays the line number on which the match occurred and the line that contains the match. Double−click a line in the Find Results tab to display the line in the Editor Pane.

The *Find All* function in the Find and Replace dialog box, and the Find in Files dialog box (*Edit/Find in Files*) both make use of the *Find Results* tab, which displays the result of the search on the Command Output tab. Komodo includes a second output tab that can be specified by checking the *Display results in Find Results 2* check box. By using both tabs, search results are not overwritten every time a new search is invoked.

# Finding Functions: Function Search

The Function Search looks through the current document for the following constructs:

- *Perl programs*: `sub` and `package` statements.
- *Python programs*: `class` and `def` statements.
- *PHP programs*: `class` and `function` statements.
- *Tcl programs*: `proc` statements.

## Moving Between Functions

To search forward from the current cursor position, select *Code/Find Next Function*, or use the associated key binding. To search backwards from the current cursor position, select *Code/Find Previous Function*, or use the associated key binding.

## Displaying a List of Functions

To find all instances of functions within the current document, select *Code/Find All Functions*, or use the associated key binding. The list of functions in the current document will be displayed on the *Find Results* tab located in the Bottom Pane of the Komodo Workspace. Double−click a specific construct on

the *Find Results* tab to highlight the relevant line in the Editor Pane.

---

# Editing

Komodo's editor provides support for writing programs in multiple languages, especially Perl, Python, Tcl, XSLT and PHP. Features of the Komodo editor are described below.

Editor functions related to whitespace (tabs, smart tabs, indentation) and smart editing (background syntax checking, AutoComplete, CallTips) can be customized to suit your preferences. Preferences can be configured to apply to all files, or to apply only to the file that is currently active in the editor. For more information, see Customizing Preferences.

Right−click in the Editor Pane for quick access to common editor functions. Use the left mouse button to select items from the context menu. In addition to standard Cut, Copy, Paste, Select All, and Print commands, the following options are also available:

- *Add as Snippet in Toolbox*: This option is only available if code has been selected in the Editor Pane. It is used to store the selected code in the Toolbox.
- *Disable/Enable Breakpoint*: Select to disable or enable a breakpoint on the line where the cursor is positioned.
- *Add/Edit Breakpoint*: Select to add or remove a breakpoint on the line where the cursor is positioned.
- *Add/Edit Spawnpoint*: Select to add or remove a spawnpoint on the line where the cursor is positioned.
- *Jump to Corresponding Line*: When comparing the changes made to a ".diff" file, select this option to shift focus from the current line in the editor tab containing the diff to the corresponding line in the tab containing the source code. This option is only available if Komodo is configured to display diffs in an open editor tab. See Configuring Source Code Control for more information.
- *Toggle Bookmark*: Select to insert or remove a bookmark on the line where the cursor is positioned.
- *Show Unsaved Changes*: Select to view, in a separate window, changes made to a file since the last time it was saved. In the new window, press *F8* to jump to the next change, *F7* to jump to the previous change, *F9* jump to a highlighted change in the original file, and *Esc* to close the window.
- *Refresh Status*: Select to refresh a file's status. See Customizing File Settings for more information.
- *Source Control*: Select to Run Source Code Control commands on the current file.
- *Properties and Settings*: Select to adjust the Current File Settings.
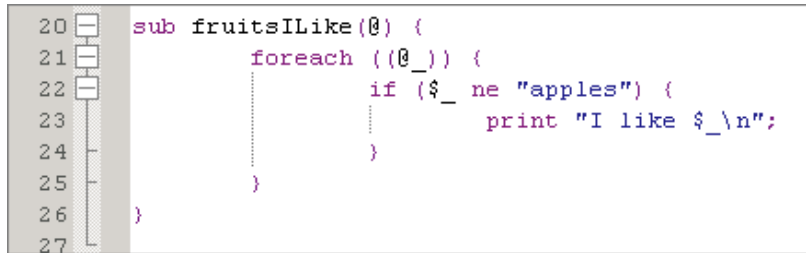
***Feature Showcases***

- previewing css in a browser
- storing a code fragment for reuse

# Language Support

## Syntax Coloring and Indentation

The Komodo editor is language–sensitive. When you open a file in a supported language, Komodo will color the syntax, format indentation, and provide indentation guides.

```
20    sub fruitsILike(@) {
21            foreach ((@_)) {
22                    if ($_ ne "apples") {
23                            print "I like $_\n";
24                    }
25            }
26    }
27
```

More Information:

- Customizing Fonts and Colors
- File Associations
- Viewing the Current File as Another Language
- Customizing Indentation

## Background Syntax Checking

As you write code in the editor, Komodo periodically checks for syntax errors. Syntax errors are underlined with a red wavy line; syntax warnings are underlined with a green wavy line. Note that Komodo uses the language interpreter's own error–checking functions. There may be slight differences in the way that syntax errors are detected and displayed, depending on the version of the interpreter.

At the bottom of the Komodo workspace, the syntax checking icon displays the syntax status of the current file displayed in the Editor Pane:

A blue check–mark over the syntax checking icon ( ) indicates that the language interpreter does not detect any warnings or errors in the program.

A red x over the syntax checking icon ( ) indicates that the interpreter detects one or more errors. To see the number of errors and warnings contained in the program, hover your mouse pointer over the syntax checking icon. A pop–up tool tip will display the total number of errors and warnings. If the syntax analysis is not yet complete, the tool tip will read "in progress".

To move the editing cursor to the line containing the error or warning, double–click the syntax checking icon. If there are multiple errors or warnings, each time you double–click the icon, the editing cursor will move to the next error.

You can view the error message from the interpreter by using one of the following methods:

- Hover the mouse pointer over the error. The interpreter error will be displayed in a pop−up tool tip.
- If the editing cursor is on the same line as the error, the interpreter error will be displayed on the status bar in the bottom left corner of the Komodo workspace.

Komodo supports background syntax checking for the following languages:

| | | |
|---|---|---|
| | | * requires ActivePerl build |
| • HTML | • Python | 623 or higher |
| • JavaScript | • Tcl | |
| • Perl * | • XML | ** requires PHP version 4.05 |
| • PHP ** | • XSLT | or greater |

More Information:

- Background Syntax Checking

# AutoComplete

AutoComplete presents a pop−up list of choices when it encounters functions, methods and properties for which there is a pre−defined set of options. AutoComplete functionality varies according to the language; for example, Python AutoComplete lists methods and properties, while XML AutoComplete lists element and attribute names and values. As you type additional characters, the items in the pop−up list will be reduced to those that match the characters you have entered.
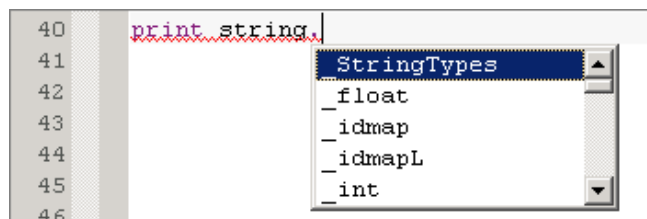


Use the arrow keys to scroll the list; use the Tab key to insert an item from the pop−up list into your document; use the Esc key to close the pop−up list.

AutoComplete functionality is described below for each supported language.

More Information:

- Enabling and Disabling AutoComplete

## PHP AutoComplete

AutoComplete for PHP is available for the following language elements:

- *Classes and Methods*: Classes in the current file, and classes in any included files, are displayed upon entry of the keyword "new". Methods in the class are displayed when you enter "–>".
- *Functions*: Standard PHP functions, and functions defined within the script (and within any included files) are displayed after you have typed four characters that match one or more function names.
- *Variables*: Variables declared in the current file, or in any included files, are displayed when you enter the symbol "$" followed by a letter. For variables declared within the current file, only those variables declared above the current line will be included in the pop–up list.

To use an item from the AutoComplete pop–up list, scroll the list to select the item, then press Tab to insert.

### *Customizing PHP AutoComplete*

PHP AutoComplete and CallTips use the built–in function definitions contained in the file *phpfunctips–<version>.txt* (located under *<komodo_installdir>/PHP/*). Specify the PHP <version> in [Configuring PHP](#) preferences. Manually alter the *phpfunctips–<version>.txt* file to customize or extend PHP AutoComplete and CallTips in Komodo. Review the existing *phpfunctips–<version>.txt* file for examples before editing.

## Python AutoComplete

AutoComplete for Python requires a [Code Intelligence database](#) to accurately list all methods and properties in a pop–up list. The Code Intelligence database contains data scanned from your Python installation and custom directories (if applicable). The database is built using a combination of two wizards. See [Building the Code Intelligence Database](#) for more information. Note that Perl, PHP, Tcl, XML, and XSLT do not require the Code Intelligence database to view AutoComplete pop–up lists.

## Perl AutoComplete

AutoComplete for Perl lists methods for modules and variables that have been used elsewhere in the same program.

## Tcl AutoComplete

AutoComplete for Tcl lists all valid methods for a command.

To use AutoComplete in Tcl files:

1. Begin typing "string is alpha". After you enter "str", a pop−up window lists methods that begin with str.
2. Move up and down through the list using the arrow keys.
3. Press Tab to select the desired method name.
4. If you keep typing, when you get to "is", the pop−up window lists a reduced list of valid methods.
5. Move up and down through the list using the arrow keys.
6. Press Tab to select the desired method name.

### *Customizing Tcl AutoComplete*

Tcl AutoComplete and CallTips use the built−in function definitions contained in Tcl *.tip* files (located under *<komodo_installdir>/tcl/*.tip*). Edit the *.tip* files to customize or extend Tcl AutoComplete and CallTips in Komodo. Review the existing *.tip* files for examples before editing.

## XML AutoComplete

AutoComplete for XML lists all valid elements for the file, attributes for each element, and suggested attribute values.

To use AutoComplete in XML files:

1. Enter "<".
   A pop−up window lists valid elements in the file.
2. Move up and down through the list using the arrow keys.
3. Press Tab to select the desired element name.
4. Press Space to insert a space after the element.
   A pop−up window lists valid attributes for that element, if attributes exist for that element.
5. Move up and down through the list using the arrow keys.
6. Press Tab to select the desired attribute name.
7. Enter "="".
   A pop−up window lists suggested attribute values, if suggestions for that attribute are available.
8. Move up and down through the list using the arrow keys.
9. Press Tab to select the desired value.
10. To end the tag, enter "">".

Autocomplete also helps with closing XML tags:

- Enter "</".
  A pop−up window lists the most recently opened tag. Note that the option includes the closing angle bracket ">".

**XSLT AutoComplete**

AutoComplete for XSLT lists all valid elements for the xsl namespace, attributes for the element, and suggested attribute values.

To use AutoComplete in XSLT files:

1. Enter "`<xsl:`".
   A pop−up window lists valid elements in the xsl namespace.
2. Move up and down through the list using the arrow keys.
3. Press Tab to select the desired element name.
4. Press Space to insert a space after the element.
   A pop−up window lists valid attributes for that element, if attributes exist for that element.
5. Move up and down through the list using the arrow keys.
6. Press Tab to select the desired attribute name.
7. Enter "`=`"".
   A pop−up window lists suggested attribute values, if suggestions for that attribute are available.
8. Move up and down through the list using the arrow keys.
9. Press Tab to select the desired value.

# CallTips

CallTips present you with a reference list of the parameters or arguments for a given function or method. To use CallTips:

1. Enter a function or method name followed by an open parenthesis "(".
2. Use the pop−up as a reference for the parameters and arguments suitable for the function or method.
3. Close the parenthesis.



```
import string
print string.split(|
                    split(s, sep=None, maxsplit=-1)
                    split(s [,sep [,maxsplit]]) -> l
```

To clear the CallTip, press the Esc key.

CallTips are available for the following languages:

- *PHP*: To customize PHP CallTips, see Customizing PHP AutoComplete.

- *Tcl*: To customize Tcl CallTips, see Customizing Tcl AutoComplete.
- *XSLT*

- *Python*: Python CallTip functionality requires a Code Intelligence database to accurately list all methods and properties. The Code Intelligence database contains data scanned from your Python installation and custom directories (if applicable). The database is built using a combination of two wizards. See Building the Code Intelligence Database for more information. Note that PHP, Tcl, and XSLT do not require the Code Intelligence database.

More Information:

- Enabling and Disabling CallTips

## Viewing the Current File as Another Language

Komodo's syntax coloring, background syntax checking, and indentation are language–specific. However, Komodo provides the option to view a file as another language. This is useful when you open, for example, a Perl file that has no extension. You can select the Perl language option, then edit the file as a regular Perl file. Komodo's File Associations do not allow you to set a language association with a file that doesn't have an extension.

To view the current file as another language:

1. On the *View* menu, select *View as Language*.
2. From the list, select the desired language.

If you have opened a file that does not have a file association specified in the Preferences dialog box, Komodo displays the file as text. You can select to view the file as another language, but Komodo does not remember again. If you will be working with a new type of file, it is recommended that you specify a file association.

For example, if you open a DocBook (*.docb) file in Komodo, it does not have XML syntax coloring. Specify a file association to tell Komodo to open *.docb files as XML files. For more information on specifying file associations, see Customizing File Associations.

If you choose to view a file in a different language and then save the file, the original language will not be restored when you re–open the file. If you are unsure of the original language, you can select *View/View As Language/Reset to best guess*. Komodo will ignore the user preference, and analyze the file in an attempt to determine its language.

## Commenting Blocks of Code

The commenting function is used to convert a single line or a group of lines into a comment, with the syntax appropriate for the file's language. Komodo supports commenting for the following languages:

More Information:

- [Commenting and Un−commenting Lines or Blocks of Code](#)

# Manipulating Code

## Automatically Repeating Keystrokes

To have Komodo repeat a key sequence a specified number of times:

1. Select *Code|Repeat Next Keystroke N Times*. The status bar at the bottom of the Komodo workspace prompts you for the number of times the keystroke(s) will be repeated.
2. Type a number using only numeric characters.
3. Enter the keystroke(s). The results of the sequence are displayed in the Editor Pane the specified number of times.

## Indenting and Un−indenting Lines of Code

To indent a single line or a selected block of code:

- *Single Line*: Position the cursor at the start of the text on the desired line. Press *Tab*, or select *Code|Increase Line Indent*.
- *Multiple Lines*: Select the desired lines by clicking and dragging in the Editor Pane. Press *Tab*, or select *Code|Increase Line Indent*.

To un−indent a single line or a selected block of code:

- *Single Line*: Position the cursor at the start of the text on the desired line. Select *Code|Decrease Line Indent* or use the associated [key binding](#).
- *Multiple Lines*: Select the desired lines by clicking and dragging in the Editor Pane. Select *Code|Decrease Line Indent*, or use the associated [key binding](#).

Specify the number of spaces per tab in the Indentation Editor Preferences (*Edit|Preferences|Editor|Indentation*).

## Reflowing Paragraphs

To reformat a section of code so that it is left−aligned and displays within the [Edge line column](#), select the section of code to be reflowed, and then select *Code|Reflow Paragraph*. Alternatively, use the

associated [key binding](#).

## Joining Lines

To cause two lines of code to display on the same line, position the cursor in the first of the two lines, and select *Code/Join Lines*. The second line is joined with the first line.

## Converting between Uppercase and Lowercase

To convert a selection of text from uppercase to lowercase (or vice–versa), from the *Code* menu, select *Make Uppercase* or *Make Lowercase*
, or use the associated [key binding](#).

## Transposing Characters

To reverse the position of the two characters to the left of the editing cursor, use the associated [key binding](#).

## Literal Characters

To insert literal characters into the editor, select *Code/Enter Next Character as Raw Literal*, and then enter the key or key combination representing the literal character. (Alternatively, use the associated [key binding](#).) For example, to insert a form feed, enter 'Ctrl'+'L'. The following characters are common:

- *Ctrl+L*: Form Feed (shown as "FF")
- *Esc*: Escape character (shown as "ESC")
- *Return or Ctrl+M*: Carriage Return (shown as "CR")
- *Ctrl+J*: Line Feed (shown as "LF")
- *Tab* or *Ctrl+I*: Tab (shown as "––––>")

## Commenting and Un–commenting Lines or Blocks of Code

To comment a single line of code, place the cursor on the desired line, then, from the *Code* menu, select *Comment Region*. Alternatively, use the associated [key binding](#).

To un−comment a line of code, place the cursor is on the desired line, then, from the ***Code*** menu, select
***Uncomment Region***. Alternatively, use the associated key binding.

To comment a block of code, select the lines you wish to comment by clicking and dragging the mouse
in the Editor Pane. Then, from the ***Code*** menu, select ***Comment Region***. Alternatively, use the associated
key binding.

To un−comment a line of code, place your cursor is on the desired line, then, from the ***Code*** menu, select
***Uncomment Region***, or use the associated key binding.


## Cleaning Line Endings

If a file contains line endings for more than one platform, you can replace the unwanted line endings with
the line endings specified in file's Properties and Settings dialog box.

1. On the ***View*** menu, click ***View EOL Markers*** to show line endings.
2. Select the line(s) for which you want to replace the endings.
3. On the ***Code*** menu, click ***Clean Line Endings***. The line endings are replaced with the line
   endings specified in the file's settings.


## Tabifying and Untabifying Regions

"Tabifying" a region converts leading spaces to tabs. If you select a line of code that has some leading
spaces and you choose to tabify the region, you convert all the leading spaces into Tab characters. The
Tabify region dialog box sets the ratio of space characters to Tab characters. If you select 8, then each 8
space characters will be represented as 1 Tab character.


To tabify a region:

1. From the ***Code*** menu, select ***Tabify Region***.
2. In the dialog box, set the number of spaces, from 1 to 16, to apply to a tab.
   Click ***OK*** or press ***Enter***.

To untabify a region:

1. From the ***Code*** menu, select ***Untabify Region***.
2. In the dialog box, set the number of spaces, from 1 to 16, to apply to a tab.
3. Click ***OK*** or press ***Enter***.

To illustrate tabifying, follow this procedure:

1. Open the sample_project.kpf.
2. Open perl_sample.pl.
3. Turn on the Line Numbers.
4. Turn on the Whitespace characters.
5. Find the following line: `$sum += $prices[$i];` There are four leading spaces on this line. You can tabify this line and convert each space character into one Tab character.
6. Tabify this line. Set the number of spaces to 1. This means each space character will be converted to one Tab character.
7. Now this line has four Tab characters, represented as right arrows, preceding `print $sum += $prices[$i];`. This causes the line to be indented too far.
8. Untabify this line. Set the number of spaces to 1. This returns the line to the original state.

Now look at another line with 8 leading spaces.

1. Open python_sample.py
2. Find the following line: `print "element %s is a string" % element`. There are 8 leading spaces on this line.
3. Tabify this line. Set the number of spaces to 8. This means the 8 spaces will be converted to one Tab character.
4. Now this line has one Tab character, represented as a right arrow, preceding `print "element %s is a string" % element;`. This does not change the line's indentation.
5. Untabify this line. Set the number of spaces to 8. This returns the line to the original state.

You can set the width of Tab characters in the Preferences dialog box. The default value is 8.

## Selecting Columns

Select columns of text in Komodo by pressing the 'Alt' key and then dragging with the mouse. This feature is particularly useful when you want to easily move code and data that is arranged in columns. Once the column of text has been selected, use the keyboard or the Edit menu to delete it or move it to another location.

## Completing Words

The Komodo editor maintains an index of words in the current file. Rather than re-entering words that already exist in the current file, use the **Complete Word** feature to finish words.

Enter one or more characters, then select **Code/Complete Word**, or use the associated key binding. Words are completed based on the most recent occurrence in the current file. For example, if you type "pr", Komodo searches backward from the insertion point to find the first instance of a word that begins with "pr". Continue pressing the spacebar while holding down the 'Ctrl' key to cycle through all possible completions for the word. The **Complete Word** feature is case sensitive.

## Selecting Blocks of Code

Quickly select blocks of code using Komodo's **Select Block** function (**Code/Select Block**, or use the associated key binding). This function uses the Code Folding logic.

When the **Select Block** function is invoked, Komodo analyzes the cursor position relevant to the blocks of code in the document. If the cursor is within a block, the entire block will be selected. (If the cursor is within a nested block, only the current sub–block will be selected, not the block that contains the entire nested structure.) If the cursor is not inside a block, the entire document will be selected.

# Editor Display Characteristics

## Toggling Whitespace On and Off

Whitespace is any space in a file not taken up by text. Line breaks, spaces, and tabs are considered whitespace.

To toggle whitespace on and off, select **View/View Whitespace**, or use the associated key binding.

To set a default for whitespace display, see Customizing Editor Features for more information.

## Toggling Indentation Guides On and Off

Indentation guides display vertical lines in the Editor Pane that indicate the number of whitespace indents. The width of indentation guides is determined by the value in the Indentation Width preference. See Customizing Indentation for more information.

To toggle indentation guides on and off, select **View/View Indentation Guides**, or use the associated key binding.

## Toggling Line Numbers On and Off

Line numbers can help orient you when working in a long file.

To toggle line numbers on and off, select **View/View Line Numbers**, or use the associated key binding.

To set this option globally, see Customizing General Editor Features for more information.

## Toggling EOL (end of line) Markers On and Off

End–of–line markers indicate where and how a line ends, such as by a hard return or another key. If you use Enter to end a line, the EOL marker could be CR or CR+LF.

To toggle EOL markers on and off, select *View*/*View EOL markers*, or use the associated key binding.

To set this option globally, see Customizing General Editor Features for more information.

## Increasing and Decreasing the Code Font Size

To increase the font size in the Editor Pane, select *View*/*Font*, and then *Increase* or *Decrease*. Alternatively, use the associated key binding. Repeat until the font size is appropriate. The size specification applies to all files open in the Editor Pane.

When you save a file, the new font size is saved.

## Toggling Fixed and Non–Fixed Width Fonts

In Komodo, you can use fixed width or non–fixed width fonts for editing. You can also toggle between these settings. The default font is non–fixed width. Note that this setting does not persist. If you toggle to a different setting, the next time you open the file it will restore the width specified on the Fonts tab of the Fonts and Colors page in Komodo Preferences.

To toggle between fixed and non–fixed width font:

1. On the *View* menu, select *Font*, then *Toggle Fixed/Proportional Fonts*. This changes the font to fixed width.
2. Repeat to reverse.

## Folding and Unfolding Code

Code folding symbols appear in the left margin of the Editor Pane immediately left of the line of code that is or can be folded. Minus signs indicate the beginning of a block of code that can be collapsed or folded. Plus signs indicate the beginning of a block of code that can be expanded or unfolded. This line of code is also underlined.

Either specific code blocks or all code blocks can be folded.

To collapse or fold *a single block of code*:

- Click the minus sign immediately to the left of a block of code

  or
- On the *View* menu, select *Fold*, then *Collapse*

  or
- Use the associated key binding.

To collapse or fold *all foldable blocks* of code:

- On the *View* menu, select *Fold*, then *Collapse All*

  All foldable blocks of code collapse and the minus signs all become plus signs.

To expand or unfold *a single block of code*:

- Click the plus sign immediately to the left of a block of code

  or
- On the *View* menu, select *Fold*, then *Expand*

  or
- Use the associated key binding.

To expand or unfold all foldable blocks of code:

- On the *View* menu, select *Fold*, then *Expand All*

  All foldable blocks of code expand and the plus signs all become minus signs.

# Navigating Within Files

## Moving to a Specific Line

While editing, you can move to a specific line number as follows:

1. On the *View* menu, select *Goto Line*.
2. In the dialog box, enter the line number, or, to move backward or forward from the current line enter "+" or "−" in front of the number. For example, enter "+5" to move five lines ahead.
3. Click *Goto Line* or press *Enter*.

## Setting and Moving to Bookmarks and Marks

Bookmarks are points of interest in a file. Komodo displays blue triangles on the left margin beside bookmarked lines. Marks, which are derived from the Emacs editor, are similar to bookmarks. The key difference is that marks have no graphical representation in Komodo. Marks make it possible to create an

invisible reminder of previously visited locations in a file.

*Bookmarks*

- *To set or unset a bookmark*: Position the editing cursor on the line of interest. Select *Code|Marks|Toggle Bookmark* or use the associated key binding to bookmark the line. If the line is already bookmarked, the bookmark will be removed.
- *To move to the next bookmark*: Select *Code|Marks|Next Bookmark* or use the associated key binding.
- *To move to the previous bookmark*: Select *Code|Marks|Previous Bookmark* or use the associated key binding.
- *To clear all bookmarks*: Select *Code|Marks|Remove All Bookmarks* or use the associated key binding.

*Marks*

- *To set a mark*: Position the cursor on the line of interest. Select *Code|Marks|Set Transient Mark*. The status bar at the bottom of the Komodo workspace indicates that a transient mark is set at the current line. If the default Emacs key binding scheme is in effect, execute this command using 'Ctrl'+'Space'.
- *To move from a position to a mark*: Select *Code|Marks|Exchange Position and Mark* to move from the cursor location (or "position") back to the associated mark. Conversely, if the cursor is located at the mark, selecting this option will move the cursor back to the previous position. If the default Emacs key binding scheme is in effect, execute this command using 'Ctrl'+'X', 'Ctrl'+'X'.
- *To move to the previous mark*: Select *Code|Marks|Move to Previous Mark* to move from the current mark to the previous mark or from the current cursor location to the previous mark. If the default Emacs key binding scheme is in effect, execute this command using 'Ctrl'+'U', 'Ctrl'+'Space'.

## Matching Braces

Use the *Matching Brace* functions to quickly jump between opening and closing braces and parentheses. Notice that when the editing cursor is adjacent to a brace or parenthesis, the brace is displayed in bold red. The associated closing or opening brace is also displayed in bold red.

To jump to the matching brace, select *Code|Jump to Matching Brace*, or use the associated key binding. To select the braces and the contents they contain, select *Code|Select to Matching Brace*.

### Detecting Changed Files

Komodo can be configured to monitor the status of files that are opened in the editor. If the file is changed on disk, you will be prompted to reload the latest version under the following circumstances:

- when you change between tabs in the editor
- when you switch back to Komodo from another application
- when you save a file

Use Komodo's Preferences to enable or disable this function.

# Preview in Browser

You can configure Komodo to preview a variety of file types in your default browser, or in the Editor Pane. The Preview in Browser feature is particularly useful when working with HTML or XML files.

The browser preview will be displayed in a separate window, in the Editor Pane, or in a split view of the Editor Pane, depending on which preference has been set.

The context menu in the Editor Pane is only available when the "source" tab is in focus. If Komodo does not support previewing of a specific file type, the Preview in Browser option will not be available from either the toolbar or the View menu.

To preview a file with the Preview in Browser feature:

1. Open the file in the Komodo Editor Pane. Or, if the file is already open, make sure it is the selected tab in the Editor Pane.
2. Select *View/Preview in Browser*. A dialog box will appear, prompting you to choose which file to preview.
3. If you want to preview the current file, select *Preview with this file*, or, if you want to preview using another file that includes the current file (e.g., use an HTML file to preview a CSS file), select *Preview with another file or URL*, then click *Browse* to navigate to the desired file location. If you do not want to be prompted each time you preview a specific file, select *Remember this selection for this file*. If you later decide that you want to specify a different preview selection, use the option on the Preview tab of the Current File Settings dialog box.
4. Click *Preview*. The file will be displayed in the Editor Pane or in a separate window, depending on which preference has been set.

# Editor Tab Display

Use the following commands on the *Window* menu to manage the way previews and tab groups are

displayed in the Editor Pane:

- *Move to Other Tab Group*: Splits the Editor Pane (if not already split) and moves the active file to the other tab group.
- *Split View*: Splits the Editor Pane (if not already split) and displays the active file in both tab groups.
- *Rotate Tab Groups*: If two tab groups are displayed, this option switches between a horizontal and vertical split.

If displayed in the Editor Pane, previews include a toolbar with basic Web browser functionality, including (from left to right) "Back", "Forward" "Reload", and "Stop" buttons.

---

# Working with Folders

Use folders to group items within projects or within the Toolbox. Project folders are virtual; that is, they do not correspond to directories on the filesystem. When you Import from File System, the virtual project or Toolbox folders are created with the same name as the imported directories. However, modifying the name of the folder within Komodo has no effect on the directory's name in the file system.

Folders are "containers"; that is, any component that can be added to a project or the Toolbox can be stored in a folder. See Adding Components to Projects or Adding Components to the Toolbox for instructions on adding folders to the Toolbox or to projects.

Double−click a folder to display its contents; double−click again to collapse a folder and hide its contents. Alternatively, click the plus and minus icons.

## Folder Options

To access options for the selected folder, do one of the following:

- ***Toolbox or Project/component_name/option***: When a component is selected in the Project Manager or Toolbox, use the ***Project*** or ***Toolbox*** drop−down menus to access the list of options. The name of the component that is currently selected in a project or the Toolbox is displayed in the drop−down menu.
- ***Context Menu***: Right−click an existing component in a project or the Toolbox and select the desired option.

The following sections describe each of the options available for folders.

## Import from File System

This option imports the files and directories from a local or network filesystem into a folder. Depending on the configuration of the options below, file and directory references are created in the same hierarchical structure as the filesystem. (Alternatively, all the file references in a recursive directory structure can be located at the same level within a folder.)

Use the following options on the Import from File System dialog box to configure the import:

- ***Directory to import from***: Specify the directory from which you want to import files. Use the ***Browse*** button to navigate the file system.
- ***Files to include***: Specify the filenames to include. Use wildcards ("*" and "?") to specify groups of files. Separate multiple file specifications with semicolons. If the field is left blank, all files in the specified directory are imported.

- *Files and directories to exclude*: Specify the file and directory names to exclude. Use wildcards ("*" and "?") to specify groups of files. Separate multiple file specifications with semicolons. If the field is left blank, no files in the specified directory are excluded.
- *Import Subdirectories Recursively*: Check this box to import directories (and files contained in those directories) located beneath the directory specified in the *Directory to import from* field. This box must be checked in order to specify the "Import Directory Structure" option as the *Type of folder structure to create*.
- *Type of folder structure to create*:
  - *Import directory structure*: If the *Import Subdirectories Recursively* box is checked and this option is selected, Komodo creates folders within the project that represent imported directories. Thus, the directory structure is preserved within the project.
  - *Make a folder per language*: If this option is selected, imported files are organized into folders according to the language indicated by file pattern in the filename. File associations are configured in the Komodo Preferences. Each folder is named after the associated language, for example, "Perl files", "XML files", etc. Files that don't correspond to a known file pattern are stored in a folder called "Other files".
  - *Make one flat list*: If this option is selected, all the imported files are placed directly under the project or folder from which the *Import from File System* command was invoked.

After importing from the file system, you are prompted to confirm the addition of the files that match the specified criteria. You may remove one or more files from the import. Click *OK* to proceed with the import.

After using the *Import from File System* feature, if you attempt to re−import the same file system location into the same project, only files that are new since the last import are imported.


# Export Contents to Package

Folders can be archived and distributed among multiple Komodo users via Packages. Packages are compressed archive files that contain the folder from which the *Export Package* option was invoked, as well as the folder's contents. Packages are stored in files with a ".kpz" extension, and can be opened by any archiving utility that supports `libz` (for example WinZip). The *Export Package* option differs from the *Export to Project* option in that a copy of filesystem−based components (such as files and dialog projects) is included in the archive. Conversely, *Export to Project* creates a project with a reference to the component's original location and does not create a copy of the component.

To export the contents of a folder to a package:

1. In the Project Manager or Toolbox, right−click the folder containing the item(s) to be exported to a package and select *Export Package*.
2. In the Package Export Wizard, enter the *Package Name* and *Export Location*. Click *Next*.
3. Click *Finish*.

# Import Contents from Package

Exported packages can only be imported into "container" objects in Komodo, such as projects, the Toolbox, and folders within projects and the Toolbox.

To import the contents of a package to a folder:

1. In the Project Manager or Toolbox, right−click the folder and select *Import Package*.
2. In the Package Import Wizard, enter the name of the package and the location on disk where files (as opposed to internal components like snippets and run commands) will be extracted. Click *Next*.
3. Click *Finish*.

# Refresh Folder Contents Status

The *Refresh Status* option checks read/write disk status for filesystem−based components (such as files and dialogs) contained within the folder. If a file is of a language for which "code intelligence" is supported and enabled (as configured in the Code Intelligence Preferences), *Refresh Status* will also update the code intelligence database with the contents of the file.

If the component is stored in a source code control system, *Refresh Status* also checks the repository status of the component. Komodo determines whether a file is contained in an SCC repository by the following methods:

- *Perforce*: analysis of the client configuration
- *CVS*: analysis of the CVS control directories

# Adding Components to Folders

Use this option to add components to the selected folder. All components can be added to folders, in the same manner they are added to projects. Refer to the individual component documentation, or the topics Adding Components to Projects or Adding Components to the Toolbox for more information.

Use the cut/copy/paste options to remove folders from a project or the Toolbox, or to move folders between the project and the Toolbox, between projects, or between folders.

## Exporting Contents as Project File

When this option is invoked, a new project file is created that contains the folder (and its contents) from which the option is invoked. You are prompted to provide the name of the new project file and the directory where it will be stored. To open the new project file, select *File|Open|Project*.

## Renaming Folders

To change the name of a folder, select this option and enter a new name.

## Source Control on Folder Contents

*Source Control on Contents* refreshes the source code control status of all the files contained within the folder. A subset of Komodo's Source Code Control functions can be performed. See Source Code Control for a description of specific options.

## Deleting Folders

To remove a folder from a project or the Toolbox, select this option. Komodo folders are virtual; deleting a folder has no effect on the disk structure or contents.

---

# Snippets

Snippets are frequently used strings that can be quickly inserted into the current document. For example, repetitive sections of code or standard comments can be stored within a snippet. Snippets have advanced properties; they support the use of Interpolation Shortcuts, can be assigned to Key Bindings, and allow for the specification of indentation context and cursor position.

Snippets are stored in the Project Manager or the Toolbox.

*Feature Showcases*

- a snippet that prompts for input
- a snippet containing a code fragment

## Creating Snippets

To create a code snippet, select the desired block of text in the Editor Pane. Then drag and drop the selected section onto the Toolbox tab or into a project on the Projects tab.

Alternatively, select the desired text, then right−click and select *Add as Snippet in the Toolbox*.

Alternatively, right−click a folder in the Toolbox or on a project or folder name on the *Projects* tab, and select *New Snippet*. If you use this method, you must manually enter the contents of the snippet; text selected in the Editor Pane is not automatically added to the Snippet dialog box.

## Configuring Snippets

To configure snippet properties, right−click the snippet on either the Toolbox tab or the Projects tab, and select *Properties*. The following configuration properties are available:

- *Snippet Name*: Enter the text that should display in the Project Manager or Toolbox for this code snippet. If the snippet was created by dragging a text selection from the Editor Pane, the snippet is named after the text in the snippet.
- *Snippet Contents*: If the snippet was created by dragging a text selection from the Editor Pane, the contents of the selected text are displayed in the Snippet Contents field. Otherwise, enter the contents of the snippet manually. Add or edit snippet content as desired.
- *Snippet Shortcuts*: Interpolate shortcuts into snippets by clicking the arrow button to the right of the Snippets Contents field, and selecting a shortcut from the drop−down menu. For a complete list of interpolation shortcut options and syntax, see Interpolation Shortcuts. (Interpolation shortcuts in snippets are not executed when the snippet is inserted in the Editor Pane via dragging and dropping.)
- *Maintain selected text or cursor position after insertion*: Within the snippet contents field, either select a portion of the snippet (by dragging the mouse pointer over the desired selection) or position the editing cursor within the string. If this check box is selected, when the snippet is inserted into the Editor Pane, the selected text or the cursor position is displayed in the same manner.

- *Maintain indentation context after insertion*: If the snippet is inserted into the Editor Pane when the editing cursor is in an indented position, select this check box to use the indentation point as an indentation "prefix". The indentation structure of the snippet is preserved at the position of insertion.

# Using Snippets

To insert the contents of a snippet at the current cursor position in the Editor Pane, double−click it, or right−click the snippet and select *Insert Snippet*.

Although you can also drag and drop snippets onto the Editor Pane, the cursor position and indentation check box options explained above in Configuring Snippets will only take effect if the snippet is added using the double−click or *Insert Snippet* method.

# Snippet Options

To access options for the selected snippet, do one of the following:

- *Toolbox/snippet_name/option* or *Project/snippet_name/option*: When a snippet is selected in the Project Manager or Toolbox tab, use the *Project* or *Toolbox* drop−down menus to access the list of options. The name of the snippet currently selected in a project or the Toolbox is displayed on the drop−down menu.
- *Context Menu*: Right−click a snippet in a project or the Toolbox and select the desired option.

The following options are available:

- *Insert Snippet*: Use this option to insert the snippet at the current cursor position in the editor, as described above in Using Snippets.
- *Cut/Copy/Paste*: These options are used to remove the snippet from a project or the Toolbox, or to move snippets between the project and the Toolbox (and vice versa).
- *Export as Project File*: When this option is invoked, a new project file is created that contains the snippet from which the option is invoked. You are prompted to provide the name of the new project file and the directory where it will be stored. To open the new project file, select *File/Open/Project*.
- *Export Package*: Snippets can be archived and distributed among multiple Komodo users via Packages. Packages are compressed archive files that contain the snippet from which the *Export Package* option was invoked. Packages are stored in files with a ".kpz" extension, and can be opened by any archiving utility that supports `libz` (for example WinZip). The *Export Package* option differs from the *Export as Project File* option in that copies of filesystem−based components (such as files and dialog projects) are included in the archive. Conversely, *Export as Project File* creates a project with a reference to the component's original location and does not create copies of the components. When *Export Package* is invoked, you are prompted for a name

and file location for the package. Exported packages can only be imported into "container" objects in Komodo, such as projects, the Toolbox, and folders within projects and the Toolbox. See Toolbox – Exporting and Importing Toolbox Contents, Projects – Importing and Exporting Projects via Packages, or Folders – Import Contents from Package for more information.

- *Rename*: To change the name of a snippet, select this option and enter a new name.
- *Delete*: To remove a snippet from a project or the Toolbox, select this option. The snippet is permanently deleted.

## Snippet Properties

Snippet properties are used to alter or rename snippets (as described in Configuring Snippets, above). The Properties dialog box is also used to assign a custom icon to a snippet or to assign a custom key binding. To access the Properties dialog box, right–click the snippet and select *Properties*.

### Assigning Custom Icons to Snippets

The default snippet icons can be replaced with custom icons. Komodo includes more than 600 icons; alternatively, select a custom image stored on a local or network drive (use 16x16–pixel images for best results).

To assign a custom icon to a snippet:

1. On the Projects tab or Toolbox tab, right–click the desired snippet and select *Properties*. Alternatively, click the icon in the Projects tab or Toolbox tab, then select *Projects|snippet_name|Properties* or *Toolboxsnippet_name|Properties*.
2. In the Properties dialog box, click *Change Icon*.
3. In the Pick an Icon dialog box, select a new icon and click *OK*. Alternatively, click *Choose Other*, and browse to the desired image file.
4. In the Properties dialog box for the snippet, click *OK*. The custom icon is displayed next to the snippet.

To revert to the default icon for a selected snippet:

1. On the Projects tab or Toolbox tab, right–click the desired snippet and select *Properties*.
2. Click *Reset*, then click *OK*. The default icon is displayed next to the snippet.

### Snippet Key Bindings

To assign a key binding to a snippet, right–click the snippet on either the Toolbox tab or the Projects tab, and select *Properties*. Select the *Key Bindings* tab, and configure the desired binding. See Key Bindings

[for Custom Components](#) for more information.

# Macros

A macro consists of keystroke sequences that are recorded, saved, and executed either manually, via a key binding, or with an event trigger. When the macro is invoked, the recorded keystrokes and instructions execute. Only keystrokes (not mouse movements) are recorded within macros. Note that, with the exception of the Find dialog box, macros that call external dialog boxes are not supported. Macros are stored in projects or the Toolbox. Custom key bindings can be assigned to macros.

The *Macros* toolbar provides quick access for recording, running, and saving macros. To open or close the toolbar, select *View/Toolbars/Macros*. Alternatively, select *Tools/Macros*.

# Creating Macros

Macros can be created via recording keystrokes, or by programming macro commands in the Macros properties dialog. For information about programming macros, refer to the Macro API.

## Recording Macros

Recording is a simple method for creating a macro. Recording a macro requires typing a series of keystrokes in the Editor Pane. To record a macro:

1. Select *Tools/Macros/Start Recording*. The Komodo status bar displays "Recording Macro".
2. In the Editor Pane, enter the keystrokes to store in the macro. While entering keystrokes, pause recording by selecting *Tools/Macros/Pause Recording*. Select *Start Recording* when ready to resume macro creation.
3. To end macro recording, select *Tools/Macros/Stop Recording*. The status bar displays "Macro Recorded".

Alternatively, use the Macros Toolbar to invoke the commands.

## Saving Recorded Macros

To save the most recent macro:

1. Select *Tools/Macros/Save to Toolbox*, or click *Macro: Save to Toolbox* on the Macro Toolbar.
2. Give the new macro a unique name in the *Enter name for new macro* field. A reference to the macro is automatically added to the Toolbox.

## Programming Macros

Use the "New Macro" Properties dialog box to program macros in either Python or JavaScript. Additionally, use this dialog box to specify macro key bindings and Komodo triggers that invoke the macro automatically.

To add a macro:

1. Select *Toolbox|Add|New Macro...* or *Project|Add|New Macro...*. Alternatively, use the Add buttons within the Project or Toolbox tab, or right−click a project or folder name and select *Add*.
2. On the *Macro* tab, configure the following options:
    - ◆ *New Macro*: Enter the name of the macro (displayed in the Toolbox and Project Manager) in the field.
    - ◆ *Change Icon*: Click to select a custom icon to associate with this macro.
    - ◆ *Reset*: Clears the icon choice.
    - ◆ *Language*: Specify the language (Python or JavaScript) in which to program the macro.
3. Program the macro in the *Language* editor field.
4. Click *OK*.

Refer to the Macro API for information about programming macros.

# Running Macros

To run the most recently recorded macro, select *Tools|Macros|Execute Last Macro*, or use the associated key binding. If the Macro Toolbar is open (*View|Toolbars|Macro*), click *Macro: Run Last Macro*.

To run a macro that has been saved to a file and assigned to a project or to the Toolbox, double−click the macro, or use the key binding assigned to the macro. Alternatively, right−click the macro in the Project Manager or the Toolbox and select *Execute Macro* from the context menu.

# Specifying Macro Triggers

Macros can be configured to execute based on certain Komodo events or triggers. When an event occurs (for example, a file is opened in Komodo), the macro is triggered and then executes. A macro that triggers "editor" functions or modifies open files should run in the foreground to block user access to the editor. Macros running in the foreground run and "block" until they return. This prevents the user from moving the cursor and disrupting the macro currently in progress.

*Macro Return Values*

Use the Macros Properties dialog box to specify a macro return value. Entering `return 1;` is

syntactically valid as a true value in either Python or JavaScript. Macros that return true and invoke *before* a Komodo event can interrupt the process under execution. For example, a macro can prevent a file from being saved if true is returned. If a macro returns "None" in JavaScript, or "Null" in Python, it evaluates to false and does not interrupt the Komodo event in progress.

*Note*: Be sure to enable macro triggers via the Projects and Workspace Preferences in Komodo's preferences. In the ***Triggering Macros*** area, select ***Enable triggering of macros on Komodo events***, and then click ***OK***.

To add a trigger to a macro:

1. Right−click the macro in the Toolbox and select ***Properties***.
2. Select the ***Triggers*** tab on the Macro Properties dialog box to access the following options:
     ♦ ***Macro should trigger on a Komodo event***: Select the check box to access the trigger events.
     ♦ Select the event you want to trigger the macro:
          ◊ ***At the tail end of the Komodo startup process***
          ◊ ***After a file is opened***
          ◊ ***Before a file is saved***
          ◊ ***After a file is saved***
          ◊ ***Before a file is closed***
          ◊ ***After a file is closed***
          ◊ ***Before Komodo shuts down (as part of File/Exit)***
     ♦ ***Rank***: Enter a numerical rank for the macro. For example, if three macros all invoke "After a file is opened", a macro executes first (100), second (101), or third (102). The default is 100 to provide room for macros to run before the default (1−99). Note that if two macros trigger on the same event with the same rank, both execute in indeterminate order.
3. Click ***Apply***.

## Running Macros in the Background

If a macro is not associated with a Komodo event, it can run either in the foreground or in the background. Depending on the use case, some macros should run in the background while others are more suitable for running in the foreground. Macros that invoke and do not affect the current file are best run in the background to minimize interference with Komodo responsiveness. Macros that perform "editor" functions or modify open files should always run in the foreground to "block" and prevent user interference. This prevents the user from moving the cursor and disrupting the macro currently in progress. Macros that run in the background are run in threads in Python, or in a timeout in JavaScript.

To run a macro in the background:

1. Right−click the macro in the Toolbox and select ***Properties***.
2. Select the ***Run in Background*** option.

3. Click *Apply*.

# Storing Macros in Projects or the Toolbox

Macros are added to the Toolbox via the *Tools/Macros/Save to Toolbox* option. However, you can manually add macros to the Toolbox, or to a project in the Project Manager.

To add a new macro to a project or the Toolbox:

1. Right−click the name of the project, or the name of a folder within a project or the Toolbox, and select *Add/New Macro* from the context menu. Alternatively, select *Project/Add/New Macro* or *Toolbox/Add/New Macro* from the drop−down menu, or select *New Macro* from the Add button in the Project or Toolbox tab.
2. Follow the instructions in the Programming Macros section.
3. Click *OK*.

Alternatively, existing macros can be dragged and dropped between the Toolbox and the Project Manager.

## Macro Options

To access macro options, right−click the macro name in a project or the Toolbox and select the desired option.

- *Execute Macro*: Use this option to run the selected macro.
- *Cut/Copy/Paste*: These options are used to remove the macro from a project or the Toolbox, or to move macros between the project and the Toolbox (and vice versa).
- *Export as Project File*: When this option is invoked, a new project file is created that contains the macro from which the option is invoked. You are prompted to provide the name of the new project file and the directory where it will be stored. To open the new project file, select *File/Open/Project*.
- *Export Package*: Macros can be archived and distributed among multiple Komodo users via "packages". Packages are compressed archive files that contain the macro from which the *Export Package* option was invoked. Packages are stored in files with a ".kpz" extension, and can be opened by any archiving utility that supports `libz` (for example WinZip). The *Export Package* option differs from the *Export as Project File* option in that copies of filesystem−based components (such as files and dialog projects) are included in the archive. Conversely, *Export as Project File* creates a project with a reference to the component's original location and does not create copies of the components. When *Export Package* is invoked, you are prompted for a name and file location for the package. Exported packages can only be imported into "container" objects in Komodo, such as projects, the Toolbox, and folders within projects and the Toolbox. See Toolbox − Exporting and Importing Toolbox Contents, Projects − Importing and Exporting

Projects via Packages, Folders – Import Contents from Package for more information.
- *Rename*: To change the name of a macro, select this option and enter a new name.
- *Delete*: To remove a macro from a project or the Toolbox, select this option. The macro is permanently deleted.

## Assigning Custom Icons to Macros

The default macro icon can be replaced with custom icons. Komodo includes more than 600 icons; alternatively, select a custom image stored on a local or network drive (use 16x16–pixel images for best results).

To assign a custom icon to a macro:

1. In the project or in the Toolbox, right–click the desired macro and select *Properties*. Alternatively, click the macro in the Projects tab or Toolbox tab, then select *Projects/macro_name/Properties* or *Toolboxmacro_name/Properties*.
2. In the Properties dialog box, click *Change Icon*.
3. In the Pick an Icon dialog box, select a new icon and click *OK*. Alternatively, click *Choose Other*, and browse to the desired image file.
4. In the properties dialog box for the macro, click *OK*. The custom icon is displayed next to the macro.

To revert to the default icon for a selected macro:

1. On the Projects tab or Toolbox tab, right–click the desired macro and select *Properties*.
2. Click *Reset*, then click *OK*. The default icon is displayed next to the macro.

## Assigning Key Bindings to Macros

Custom key bindings can be assigned to macros stored in the Toolbox or in a Project. Use the *Key Binding* tab in the macro's Properties to specify the keystrokes that invoke the macro. See Key Bindings for Custom Components for more information.

# Macro API

## Introduction to the Komodo Macro API

Komodo macros can be written in either JavaScript or Python. As much as possible, the API calls are the same regardless of the language. Exceptions are noted where appropriate.

In both Python and JavaScript, there is a top–level **komodo** object that contains both variables and utility functions. These are:

- The editor object for manipulation of code buffers.
- The document object for manipulation of documents in memory.
- The file type, corresponding to files on disk.
- The doCommand(...) function to execute Komodo "commands".
- The findPart(...) function to find other components (snippets, run commands, other macros, etc).
- The interpolate(...) function for evaluation of interpolation codes.
- The getWordUnderCursor() function to retrieve the word under the editing cursor.

### Warning

The macro system is a powerful mechanism by which Komodo users can execute arbitrary code inside the Komodo process. It is easy for novices to write macros that can significantly disrupt Komodo's behavior, leading to instability and data loss. We encourage users to experiment with macros when *not* working with important files.

Of particular note:

- Macros that never terminate (for example, due to infinite loops) can hang Komodo.
- Macros that modify the buffer should never be run in the background, as multiple threads accessing the editor object could cause crashes.
- Macros that modify the editor object should be written with care, to avoid data loss

.

### Feedback

Komodo 3.0 is the first release of Komodo with a published macro API. We expect the API to change in response to customer feedback. As such, the API described below may change in future releases (we will of course try to minimize backwards–incompatible changes). Questions and feedback on the API are encouraged via the komodo–discuss mailing list.

## The editor Object

The `komodo.editor` object corresponds to the main text editing widget that contains and manipulates

files in the Editor Pane. It is a thin wrapper around the Scintilla widget, an open–source component written by Neil Hodgson ([www.scintilla.org](http://www.scintilla.org)).

The Scintilla API is large, complex and subject to change. This document only contains the calls most relevant to Komodo, and notes some common patterns of use relevant to changing the *editor* widget.

## editor Attributes

*int currentPos*
> The location (in character units) of the caret.

*int anchor*
> The location (in character units) of the selection anchor.

*string text*
> The contents of the buffer.

*string selText*
> The contents of the selection (readonly).

*long scrollWidth*
> The width of the scroll area (in pixels).

*long xOffset*
> The horizontal scroll position (in pixels) of the start of the text view.

*boolean viewEOL*
> Whether to show end–of–line markers or not.

*long viewWS*
> Whether to show whitespace characters (0: no, 1: yes).

*long eOLMode*
> The characters that are inserted when the user presses 'Enter': either 'CRLF' (0 – the default on Windows), 'CR' (1) or 'LF' (2 – the default on Unix).

*long tabWidth*
> The size of a tab as a multiple of the size of a space character.

*long indent*
> The size of indentation in terms of the width of a space.

*boolean useTabs*
> Whether indentation should be created out of a mixture of tabs and spaces (1) or be based purely on spaces (0).

*boolean indentationGuides*
> Whether to show indentation guides or not.

*readonly long firstVisibleLine*
> The line number of the first visible line in the text view.

*long lineCount*
> The number of lines in the text view.

*long textLength*
> The length of the current buffer in characters.

*long targetStart*
> The start of the target region; see replaceTarget.

*long targetEnd*
> The end of the target region; see replaceTarget.

*long linesOnScreen*
> The number of complete lines visible on the screen.

## komodo.editor Methods

*void emptyUndoBuffer()*
> Empty the undo buffer.

*void undo()*
> Undo the last action.

*void cut()*
> Cut current selection (`komodo.doCommand('cmdCut')` is the preferred method).

*void copy()*
> Copy current current selection.

*void paste()*
> Replace current selection with the clipboard contents.

*void clear()*
> Clear current selection.

*long replaceTarget(in long length, in string text)*
> Replace XXX.

*string getTextRange(in long min, in long max)*
> Return a range of characters from the current buffer.

*void insertText(in long pos, in string text)*
> Insert text at a specified position.

*void colourise(in long start, in long end)*
> Force the re–coloring of the specified range.

*wchar getWCharAt(in long pos)*
> Get the (Unicode) character at the specified position.

*void addText(in long length, in string text)*
> Add text to the end of the current buffer.

*void selectAll()*
> Select the entire buffer.

*void gotoLine(in long line)*
> Jump to the specified line.

*void gotoPos(in long pos)*
> Jump to the specified position in the buffer.

*void deleteBack()*
> Delete the character to the left of the cursor.

*void newLine()*
> Add a newline (note: this is a less 'smart' newline than can be obtained using `komodo.doCommand('cmd_newlineExtra')`.

*void redo()*
> Redo the last action.

*boolean canRedo()*
> There is an action that can be redone.

*void beginUndoAction()*
> Begin an undo block (see note).

*void endUndoAction()*
> End an undo block (see note).

*long getColumn(in long pos)*
> Get the column (0–based) of the specified position.

*long getLineEndPosition(in long line)*
> Get the position corresponding to the last character on the specified line (not including EOL characters).

*void setSel(in long start, in long end)*
> Make selection start at `start` and end at `end`.

*long lineFromPosition(in long pos)*
> Get the line number (0–indexed) from character position `pos`.

*long positionFromLine(in long line)*
> Get character position which begins the specified line.

*void lineScroll(in long columns, in long lines)*
> This will attempt to scroll the display by the number of columns and lines that you specify. Positive line values increase the line number at the top of the screen (i.e. they move the text upwards as far as the user is concerned). Negative line values do the reverse.

*void scrollCaret()*
> If the current position (this is the caret if there is no selection) is not visible, the view is scrolled to make it visible.

*long lineLength(in long line)*
> Return the length of the current line.

*void replaceSel(string)*
> Replace current selection with the text in the *string*.

*void lineDown()*
> Move cursor down one line.

*void lineDownExtend()*
> Extend selection down one line.

*void lineUp()*
> Move cursor up one line.

*void lineUpExtend()*
> Extend selection up one line.

*void charLeft()*
> Move cursor one character to the left.

*void charLeftExtend()*
> Extend selection one character to the left.

*void charRight()*
> Move cursor one character to the right.

*void charRightExtend()*
> Extend selection one character to the right.

*void wordLeft()*
> Move cursor one word to the left.

*void wordLeftExtend()*
> Extend selection one word to the left.

*void wordRight()*
> Move cursor one word to the right.

*void wordRightExtend()*

Extend selection one word to the right.

*void home( )*

Move cursor to the Home position.

*void homeExtend( )*

Extend selection to the Home position.

*void lineEnd( )*

Move cursor to the end of the line.

*void lineEndExtend( )*

Extend selection to the end of the line.

*void documentStart( )*

Move cursor to the start of the document.

*void documentStartExtend( )*

Extend selection to the start of the document.

*void documentEnd( )*

Move cursor to the end of the document.

*void documentEndExtend( )*

Extend selection to the end of the document.

*void pageUp( )*

Page up.

*void pageUpExtend( )*

Extend selection one page up.

*void pageDown( )*

Page down.

*void pageDownExtend( )*

Extend selection one page down.

*void editToggleOvertype( )*

Toggle overtype mode.

*void vCHome( )*

Move cursor to the first non−whitespace character on a line or, if none, the beginning of a line.

*void vCHomeExtend( )*

Extend the selection to the first non−whitespace character on a line or, if none, the beginning of a line.

*void zoomIn( )*

Increase font size.

*void zoomOut( )*

Decrease font size.

*void delWordLeft( )*

Delete word to the left of the cursor.

*void delWordRight( )*

Delete word to the right of the cursor.

*void lineCopy( )*

Copy line at the cursor.

*void lineCut( )*

Cut line at the cursor.

*void lineDelete( )*

Delete line at the cursor.

*void lineTranspose( )*

Transpose current line and previous line.

*void lineDuplicate( )*

Duplicate current line.

*void lowerCase( )*

Convert selection to lower case.

*void upperCase( )*

Convert selection to upper case.

*void lineScrollDown( )*

Scroll display down one line.

*void lineScrollUp( )*

Scroll display up one line.

*void deleteBackNotLine( )*

Delete last character except if at beginning of line.

*void homeDisplay( )*

Move cursor to Home position for the current display line (as opposed to the buffer line when word wrap is enabled).

*void homeDisplayExtend( )*

Extend selection to the Home position for the current display line (as opposed to the buffer line when word wrap is enabled).

*void lineEndDisplay( )*

Move cursor to end of the current display line (as opposed to the buffer line when word wrap is enabled).

*void lineEndDisplayExtend( )*

Extend selection to the end of the current display line (as opposed to the buffer line when word wrap is enabled).

*void wordPartLeft( )*

Move cursor a word segment to the left. Word segments are marked by capitalisation (aCamelCaseIdentifier) or underscores (an_under_bar_ident).

*void wordPartLeftExtend( )*

Extend selection a word segment (as described in void wordPartLeft()) to the left.

*void wordPartRight( )*

Move cursor a word segment (as described in void wordPartLeft()) to the right.

*void wordPartRightExtend( )*

Extend selection a word segment (as described in void wordPartLeft()) to the right.

*void delLineLeft( )*

Delete to beginning of line.

*void delLineRight( )*

Delete to end of line.

*void paraDown( )*

Move cursor one paragraph down.

*void paraDownExtend( )*

Extend selection one paragraph down.

*void paraUp( )*

Move cursor one paragraph up.

*void paraUpExtend( )*

Extend selection one paragraph up.

## editor Object Notes

*Invalid Parameters*: The Scintilla API assumes that users of the API do their own error−checking. Passing arguments that are out of bounds or otherwise erroneous can result in Komodo crashing.

*The Undo Stack*: Scintilla manages the "undo" stack. To treat a sequence of operations as a single operation for the sake of Undo/Redo, wrap these operations in a <u>beginUndoAction</u> / <u>endUndoAction</u> pair. The endUndoAction must be called even in the case of an exception in the code. Otherwise, the undo stack will be corrupted and might lose data.

For example, for JavaScript:

```
komodo.editor.beginUndoAction()
try {
    ...  // do your sequence here
} finally {
    komodo.editor.endUndoAction()
}
```

For Python:

```
komodo.editor.beginUndoAction()
try:
    ...  # do your sequence here
finally:
    komodo.editor.endUndoAction()
```

# The document Object

The `komodo.document` object refers to the current document. A document contains the contents of the file being edited. These contents will be different than those of the file on disk if the file is unsaved or "dirty".

## document Attributes

*string baseName*
>       The basename of the document (e.g. `"myfile.txt"`).

*string displayPath*
>       The display path of the document (e.g. `"C:\Code\myfile.txt"`).

*file*
>       The <u>file</u> object corresponding to the document (null if the document is unsaved).

*string buffer*
>       The contents of the document (Unicode string).

*boolean isDirty*
>       Whether there are unsaved changes to the document.

*boolean isUntitled*
>       Whether the document has never been saved.

*string language*

The language that this document is viewed as ("Python", "Perl", etc.

# The file Object

The `file` object is an attribute of document objects, and corresponds to a wrapper object around the file object.

## `file` attributes

*string URI*
    The URI to the file (e.g. "file:///C:/Code/myfile.txt").
*string displayPath*
    The display path of the file (e.g. "C:\Code\myfile.txt"), or the URI if the URI is not of the `file://` scheme.
*string baseName*
    The base name of the file (e.g. "myfile.txt").
*string dirName*
    The directory of the file (e.g. "C:\Code").

# The komodo.doCommand Function

*Signature*:

    komodo.doCommand(commandId)

Execute the internal Komodo command specified by `commandId`.

Command IDs and their corresponding functions are available in the Command ID reference.

Most editor–related commands require that the Editor Pane be in focus. To ensure focus before invoking `doCommand`, set the focus explicitly as follows:

    komodo.view.setFocus()

# The komodo.findPart Function

*Signature*:

    komodo.findPart(type, name, where) -> part

Find a "part" (the internal name for a component such as a snippet, another macro, a run command, etc) in the Toolbox or a project.

- `type`: The type of component to search for. It can be one of:
    - `"snippet"`

- ♦ `"command"`
- ♦ `"macro"`
- ♦ `"file"`
- ♦ `"folder"`
- ♦ `"dialog"`
- ♦ `"URL"`
- ♦ `"template"`
- ♦ `"DirectoryShortcut"`
- name: The component's name.
- where: A string corresponding to the component container that should be searched. Supported values are:

  *`"toolbox"`*
   search in the Toolbox

  *`"shared toolbox"`*
   search in the Shared Toolbox (if enabled)

  *`"toolboxes"`*
   search in both the Toolbox and the Shared Toolbox

  *`"container"`*
   search the project or Toolbox that contains the current macro

  *`"*"`*
   search all of the above

# The komodo.interpolate Function

***Signature***:

```
komodo.interpolate(s, bracketed=False) -> string
```

Evaluate interpolation shortcuts in the given string.

- s: The string to interpolate.
- bracketed: An optional boolean value indicating whether plain (e.g. `%F`) or bracketed (e.g. `[[%F]]`) syntax is being used. If not specified, plain interpolation is used.

Some interpolation shortcuts cannot be used within Python macros. These include `%P` and `%ask`, and the `:orask` modifier on other shortcuts. A `ValueError` is raised if they are used.

# The komodo.getWordUnderCursor Function

***Signature***:

```
komodo.getWordUnderCursor() -> string
```

This function returns the word under the cursor in the current buffer.

# Komodo Command Id List

## Breakpoint Manager

| | |
|---|---|
| Add Breakpoint | cmd_breakpoint_add |
| Add Tcl Spawnpoint | cmd_spawnpoint_add |
| Delete All Breakpoints | cmd_breakpoint_delete_all |
| Delete Breakpoint | cmd_breakpoint_delete |
| Edit Breakpoint | cmd_breakpoint_properties |
| Enable/Disable All Breakpoints | cmd_breakpoint_toggle_all |
| Enable/Disable Breakpoint | cmd_breakpoint_toggle |
| Show Breakpoint | cmd_breakpoint_goto |

## Code Browser

| | |
|---|---|
| Find Symbol | cmd_codeBrowserFindSymbol |
| Go to Definition | cmd_codeBrowserGoToDefinition |
| Locate current scope... | cmd_codeBrowserLocate |

## Code Intelligence

| | |
|---|---|
| Find Symbol | cmd_codeIntelFindSymbol |

## Debugger

| | |
|---|---|
| Add Watch Variable | cmd_dbgAddVariable |
| Add/Edit Breakpoint... | cmd_dbgBreakpointAddOrEdit |
| Add/Edit Spawnpoint... | cmd_dbgSpawnpointAddOrEdit |
| Break Now | cmd_dbgBreakNow |
| Clear All Breakpoints | cmd_dbgBreakpointClearAllInURI |
| Delete Selected Variable | cmd_dbgDeleteVariable |
| Detach | cmd_dbgDetach |
| Disable/Enable Breakpoint | cmd_dbgBreakpointToggle |
| Edit Selected Variable Name | cmd_dbgWatchedVariable |
| Edit Selected Variable Value | cmd_dbgEditVariable |
| Interactive Debugger Shell Clear Buffer | cmd_dbgInteractiveClearBuffer |
| Interactive Debugger Shell | cmd_dbgInteractive |

| | |
|---|---|
| Listen for Remote Debugger | cmd_debuggerListener |
| Load HTML Preview | cmd_dbgViewAsHTML |
| Make Selection a Watched Variable | cmd_dbgMakeWatchedVariable |
| New Session | cmd_dbgNewSession |
| Run Script | cmd_dbgRun |
| Run to Cursor | cmd_dbgStepCursor |
| Show Current Statement | cmd_dbgShowCurrentStatement |
| Show Hidden Variables | cmd_dbgShowHiddenVars |
| Start Default Interactive Shell | cmd_startInteractiveShell |
| Start Perl Interactive Shell | cmd_startPerlInteractiveShell |
| Start Python Interactive Shell | cmd_startPythonInteractiveShell |
| Start Tcl Interactive Shell | cmd_startTclInteractiveShell |
| Start | cmd_dbgGo |
| Start/Find/Hide Default Interactive Shell | cmd_toggleInteractiveShell |
| Step Forward | cmd_dbgStepForward |
| Step In | cmd_dbgStepIn |
| Step Out | cmd_dbgStepOut |
| Step Over | cmd_dbgStepOver |
| Stop | cmd_dbgStop |

## Editor

| | |
|---|---|
| Add as Snippet to Toolbox | cmd_makeSnippetFromSelection |
| Back | cmd_back |
| Backspace | cmd_backSmart |
| Beginning of Line (first char/first column) | cmd_home |
| Cancel AutoComplete | cmd_cancel |
| Compare with file on disk | cmd_showUnsavedChanges |
| Copy | cmd_copy |
| Current File Settings | cmd_editPrefsCurrent |
| Cut Line | cmd_lineCut |
| Cut Region | cmd_cutRegion |
| Cut rest of line | cmd_killLine |
| Cut | cmd_cut |

| | |
|---|---|
| Decrease Line Indent | cmd_dedent |
| Delete Line | cmd_lineDelete |
| Delete Word Left | cmd_deleteWordLeft |
| Delete Word Right | cmd_deleteWordRight |
| Delete | cmd_delete |
| End of Line | cmd_end |
| Exchange Current Point and Mark | cmd_transientMarkExchangeWithPoint |
| Go to Beginning of word | cmd_beginningOfWord |
| Go to End of Document | cmd_documentEnd |
| Go to End of word | cmd_endOfWord |
| Go to Line... | cmd_gotoLine |
| Go to Next Bookmark | cmd_bookmarkGotoNext |
| Go to Next Line | cmd_lineNext |
| Go to Previous Bookmark | cmd_bookmarkGotoPrevious |
| Go to Previous Line | cmd_linePrevious |
| Go to Top of Document | cmd_documentHome |
| Increase Line Indent | cmd_indent |
| Insert Newline (align with current line) | cmd_newlineSame |
| Insert Newline (continue comments) | cmd_newlineExtra |
| Insert Newline (no favors) | cmd_newlineBare |
| Insert Newline | cmd_newline |
| Insert Next Key as Literal Character | cmd_rawKey |
| Join current and next lines | cmd_join |
| Move Back Part of Word | cmd_wordPartRight |
| Move Forward Part of Word | cmd_wordPartLeft |
| Move Left One Character | cmd_left |
| Move One Character Right | cmd_right |
| Move One Word Left | cmd_wordLeft |
| Move One Word Right | cmd_wordRight |
| Move to previous mark in mark ring | cmd_transientMarkMoveBack |
| Page Down | cmd_pageDown |
| Page Up | cmd_pageUp |
| Paste and Select | cmd_pasteAndSelect |

| | |
|---|---|
| Paste | cmd_paste |
| Redo | cmd_redo |
| Reflow paragraph(s) | cmd_editReflow |
| Remove All Bookmarks | cmd_bookmarkRemoveAll |
| Repeat next keystroke N times | cmd_repeatNextCommandBy |
| Reset language to best guess | cmd_viewAsGuessedLanguage |
| Scroll One Line Down | cmd_lineScrollDown |
| Scroll One Line Up | cmd_lineScrollUp |
| Scroll current line to center of screen | cmd_editCenterVertically |
| Scroll current line to top of screen | cmd_editMoveCurrentLineToTop |
| Select All | cmd_editSelectAll |
| Select Next Character | cmd_selectCharNext |
| Select Next Part of Word | cmd_wordPartRightExtend |
| Select Next Word | cmd_selectWordRight |
| Select Page Down | cmd_selectPageDown |
| Select Page Up | cmd_selectPageUp |
| Select Previous Character | cmd_selectCharPrevious |
| Select Previous Part of Word | cmd_wordPartLeftExtend |
| Select Previous Word | cmd_selectWordLeft |
| Select to Beginning of Line (first char/first column) | cmd_selectHome |
| Select to Beginning of word | cmd_beginningOfWordExtend |
| Select to End of Document | cmd_selectDocumentEnd |
| Select to End of Line | cmd_selectEnd |
| Select to End of word | cmd_endOfWordExtend |
| Select to Next Line | cmd_selectLineNext |
| Select to Previous Line | cmd_selectLinePrevious |
| Select to Top of Document | cmd_selectDocumentHome |
| Set Mark | cmd_transientMarkSet |
| Toggle Bookmark | cmd_bookmarkToggle |
| Toggle Overtype/Insert Mode | cmd_toggleOvertype |
| Transpose Current and Previous Characters | cmd_transpose |
| Transpose Current and Previous Lines | cmd_lineTranspose |
| Transpose Current and Previous Words | cmd_transposeWords |

| | |
|---|---|
| Trigger preceding AutoComplete list or CallTip | cmd_triggerPrecedingCompletion |
| Undo | cmd_undo |
| Zoom Font Size Down | cmd_fontZoomOut |
| Zoom Font Size Up | cmd_fontZoomIn |

# Find

| | |
|---|---|
| Find Next Result | cmd_findNextResult |
| Find Next Selected | cmd_findNextSelected |
| Find Next | cmd_findNext |
| Find Previous | cmd_findPrevious |
| Find in Files... | cmd_findInFiles |
| Find... | cmd_find |
| Incremental Search Backwards | cmd_startIncrementalSearchBackwards |
| Incremental Search | cmd_startIncrementalSearch |
| Replace... | cmd_replace |

# General

| | |
|---|---|
| Close All Windows | cmd_closeAll |
| Close Project | cmd_closeProject |
| Close Window | cmd_bufferClose |
| Edit Properties | cmd_editProperties |
| Export Selection as HTML... | cmd_exportHTMLSelection |
| Export as HTML... | cmd_exportHTML |
| Import From File System into Current Project | cmd_importFromFS_Project |
| Least Recently Viewed File | cmd_bufferNextLeastRecent |
| Most Recently Viewed File | cmd_bufferNextMostRecent |
| New File (default type) | cmd_new |
| New File... | cmd_newTemplate |
| New Project... | cmd_newProject |
| Next File | cmd_bufferNext |
| Open File... | cmd_open |
| Open Project... | cmd_openProject |
| Open Remote File... | cmd_open_remote |

| | |
|---|---|
| Open Template... | cmd_openTemplate |
| Open URL... | cmd_openURL |
| Preferences... | cmd_editPrefs |
| Previous File | cmd_bufferPrevious |
| Print Preview | cmd_printPreview |
| Print Selection... | cmd_printSelection |
| Print Setup... | cmd_printSetup |
| Print... | cmd_print |
| Quit | cmd_quit |
| Refresh Status | cmd_refreshStatus |
| Revert Project | cmd_revertProject |
| Revert | cmd_revert |
| Save All | cmd_saveAll |
| Save As Remote... | cmd_saveAs_remote |
| Save As Template | cmd_saveAsTemplate |
| Save As... | cmd_saveAs |
| Save Project As... | cmd_saveProjectAs |
| Save Project | cmd_saveProject |
| Save | cmd_save |

## Help

| | |
|---|---|
| About Komodo | cmd_helpAbout |
| ActiveState Programmer Network | cmd_helpASPN |
| ActiveState Support | cmd_helpSupport |
| ActiveState Website | cmd_helpActiveState |
| Alternate Help on Selection | cmd_helpLanguageAlternate |
| Help... | cmd_helpHelp |
| Komodo Bug Database | cmd_helpViewBugs |
| Language−Specific Help on Selection | cmd_helpLanguage |
| PHP Mailing Lists | cmd_helpPHPMailingLists |
| PHP Reference (Web) | cmd_helpPHPRef_Web |
| Perl Mailing Lists | cmd_helpPerlMailingLists |
| Perl Reference (Local) | cmd_helpPerlRef_Local |

| | |
|---|---|
| Perl Reference (Web) | cmd_helpPerlRef_Web |
| Python Mailing Lists | cmd_helpPythonMailingLists |
| Python Reference (Local) | cmd_helpPythonRef_Local |
| Python Reference (Web) | cmd_helpPythonRef_Web |
| Show Keybindings... | cmd_helpShowKeybindings |
| Software Updates | cmd_helpUpdates |
| Start Page | cmd_openStartPage |
| Tcl Mailing Lists | cmd_helpTclMailingLists |
| Tcl Reference (Local) | cmd_helpTclRef_Local |
| Tcl Reference (Web) | cmd_helpTclRef_Web |
| Tip of the Day... | cmd_helpTipOfTheDay |
| XSLT Mailing Lists | cmd_helpXSLTMailingLists |

## Macro

| | |
|---|---|
| End Recording | cmd_stopMacroMode |
| Execute Last Macro | cmd_executeLastMacro |
| Pause Recording | cmd_pauseMacroMode |
| Save to Toolbox | cmd_saveMacroToToolbox |
| Start/Resume Recording | cmd_startMacroMode |

## Projects/Toolbox

| | |
|---|---|
| Add New Folder... | cmd_addFolderPart |
| Add Remote File... | cmd_addRemoteFilePart |

## Source Code

| | |
|---|---|
| Clean Line Endings | cmd_cleanLineEndings |
| Collapse All Folds | cmd_foldCollapseAll |
| Collapse Fold | cmd_foldCollapse |
| Comment Region | cmd_comment |
| Complete Word (backwards) | cmd_completeWordBack |
| Complete Word | cmd_completeWord |
| Convert Selection to Lower Case | cmd_convertLowerCase |
| Convert Selection to Upper Case | cmd_convertUpperCase |
| Expand All Folds | cmd_foldExpandAll |

| | |
|---|---|
| Expand Fold | cmd_foldExpand |
| Find All Functions | cmd_findAllFunctions |
| Find Next Function | cmd_findNextFunction |
| Find Previous Function | cmd_findPreviousFunction |
| Jump to Matching Brace | cmd_jumpToMatchingBrace |
| Jump to corresponding line | cmd_jumpToCorrespondingLine |
| Match Braces | cmd_braceMatch |
| Next Syntax Error/Warning | cmd_nextLintResult |
| Select Block | cmd_blockSelect |
| Select to Matching Brace | cmd_selectToMatchingBrace |
| Tabify Region... | cmd_tabify |
| Un–comment Region | cmd_uncomment |
| Un–tabify Region... | cmd_untabify |

## Source Control

| | |
|---|---|
| Add File | cmd_SCCadd |
| Add Folder | cmd_SCCadd_folder |
| Commit Changes in Folder | cmd_SCCcommit_folder |
| Commit Changes | cmd_SCCcommit |
| Compare Files in Folder | cmd_SCCdiff_folder |
| Compare | cmd_SCCdiff |
| Edit | cmd_SCCedit |
| Remove File | cmd_SCCremove |
| Revert Changes in Folder | cmd_SCCrevert_folder |
| Revert Changes | cmd_SCCrevert |
| Update Folder | cmd_SCCupdate_folder |
| Update | cmd_SCCupdate |

## Toolbox

| | |
|---|---|
| Export Package | cmd_toolboxExportPackage |

## Tools

| | |
|---|---|
| Build Standalone Perl Application... | cmd_toolsBuildPerlAppCommand |
| Run Command... | cmd_toolsRunCommand |

| | |
|---|---|
| Rx Toolkit | cmd_toolsRx |
| Visual Package Manager | cmd_VPM |
| Watch File... | cmd_toolsWatchFile |

# User Interface

| | |
|---|---|
| Browser Preview | cmd_browserPreview |
| Focus on Bottom Pane | cmd_focusBottomPane |
| Focus on Editor | cmd_focusEditor |
| Focus on Projects Pane | cmd_focusProjectPane |
| Focus on Toolbox Pane | cmd_focusToolboxPane |
| Move tab to another tab window | cmd_movetab |
| Show/Hide Code Browser Tab | cmd_viewCodeBrowser |
| Show/Hide Debug Toolbar | cmd_viewdebugtoolbar |
| Show/Hide End of Lines | cmd_viewEOL |
| Show/Hide Macros Toolbar | cmd_viewmacrotoolbar |
| Show/Hide Open/Find Toolbar | cmd_viewfindtoolbar |
| Show/Hide Output Pane | cmd_viewBottomPane |
| Show/Hide Projects Pane | cmd_viewLeftPane |
| Show/Hide Projects Tab | cmd_viewProjects |
| Show/Hide SCC Toolbar | cmd_viewscctoolbar |
| Show/Hide Standard Toolbar | cmd_viewedittoolbar |
| Show/Hide Toolbar Button Text | cmd_toggleButtonText |
| Show/Hide Toolbox Pane | cmd_viewRightPane |
| Show/Hide Toolbox Tab | cmd_viewToolbox |
| Show/Hide Tools Toolbar | cmd_viewtoolstoolbar |
| Show/Hide Whitespace | cmd_viewWhitespace |
| Show/Hide Workspace Toolbar | cmd_viewworkspacetoolbar |
| Split view of tab | cmd_splittab |
| Toggle splitter orientation | cmd_rotateSplitter |
| Use Fixed/Proportional Font | cmd_fontFixed |
| View Source | cmd_focusSource |
| View/Hide Indentation Guides | cmd_viewIndentationGuides |
| View/Hide Line Numbers | cmd_viewLineNumbers |

| Word–wrap long lines | cmd_wordWrap |

# Templates

Templates are files that contain the basic structure for new files. For example, a Perl template for a specific project might contain the standard shebang line, appropriate copyright statements, and `use` statements calling a standard set of modules.

Komodo includes a number of pre−configured templates for multiple languages, and provides support for the creation and use of custom templates. Templates can contain [Interpolation Shortcuts](#) (variables that get replaced by context− or time−specific information, such as the complete path of the current file or the date). Templates can be stored in a [project](#) or the [Toolbox](#) for quick access. Templates can be shared via the [Common Data Directory](#).

*Feature Showcase*

- store a [template in a project](#)

## Creating New Files from Templates

The *New File* menu option, invoked via *File|New|New File*, or via the associated [key binding](#), provides access to numerous templates for creating new files. These templates consist of standard code that is generally included in programs of the selected type. For example, the Perl template creates a file with a ".pl" extension that contains the line "`use strict;`"; the XSLT stylesheet template creates a file with an ".xsl" extension and an xml version and xsl stylesheet declaration.

To create a new file from a template file, in the New File dialog box, select the category in the Left Pane, then double−click the template name in the Right Pane. A new file is created in the Komodo editor using the code stored in the template.

When using the *New File* button on the Standard Toolbar, the template (if any) associated with the language in the [new files preferences](#) is used to create the new file.

Alternatively, templates can be stored in a [project](#) or the [Toolbox](#), and can be associated with a key binding.

## Creating Custom Templates

Custom templates are created in the Komodo editor; on saving the file, it is identified as a template.

1. *Create Template File*: In Komodo, create a file with the desired contents for the template.
2. *Save as Template*: Select *File|Save As Template*.

Custom templates are stored in the "My Templates" folder (as displayed in the New File dialog box). To organize templates, click the *Open Template Folder* button. This displays the selected "Category" (which corresponds to a disk location) in the operating system's file explorer application. Create new directories and move template files between directories to organize templates.

Optionally, create shortcuts (on Windows) or symlinks (on Linux and Solaris) in the "My Templates" directory structure that point to files located elsewhere on the system. Files should have an extension that allows Komodo to correctly detect the language (according to the settings configured in Komodo's File Associations preference).

If you create a directory alongside `My Templates` with the same name as a template group that already exists in Komodo (such as `Common` or `Web`), the contents of the two directories are merged. If files of the same name exist in both directories, the file in the directory at the same level as `My Templates` is used.

For example:

```
templates\
  My Templates\    <--directory
    MyCGI.pl         <--file in the My Templates directory
    TestSystem.tcl   <--file in the My Templates directory
    Corporate        <--shortcut/symlink to corporate templates
  Common\          <--directory
    Python.py        <--file; takes precedence over the Python.py template
    MyCGI.pl         <--file; displays in the Common folder
```

To edit an existing template, select *File|Open|Template File*. The directory containing the `My Templates` directory (and any "merge" directories, as described above) is displayed. Select the desired template and make the desired changes.

## Using Interpolation Shortcuts in Custom Templates

Interpolation shortcuts can be used in templates. When a new file is generated from a template file containing interpolation shortcuts, the shortcut is converted to its relevant value.

For example, if a template file contains the interpolation shortcut `[[%date:%d/%m/%Y %H:%M:%S]]`, when a new file is created from that template, the interpolation shortcut is converted to the current date and time in the following format: `27/06/2004 11:10:21`.

Interpolation shortcuts within templates use the bracketed syntax. Refer to the interpolation shortcut section for a complete description of the code, syntax and usage.

# Storing Templates in a Project or the Toolbox

Use the Komodo Project Manager or the Toolbox to store frequently used templates. In addition to the general methods described in Adding Components to Projects and Adding Components to the Toolbox, templates can be added to the Toolbox via the New File dialog box. Select *Add to Toolbox* to add the selected template to the Toolbox.

## Template Options

Right−click a template to access the following options:

- *Open Template*: Use this option to create a new file from the selected template.
- *Cut/Copy/Paste*: These options are used to remove the template from a project or the Toolbox, or to move templates between the project and the Toolbox (and vice versa).
- *Export as Project File*: When this option is invoked, a new project file is created that contains the template from which the option is invoked. You are prompted to provide the name of the new project file and the directory where it will be stored. To open the new project file, select *File|Open|Project*.
- *Export Package*: Templates can be archived and distributed among multiple Komodo users via "packages". Packages are compressed archive files that contain the template from which the *Export Package* option was invoked. Packages are stored in files with a ".kpz" extension, and can be opened by any archiving utility that supports `libz` (for example WinZip). The *Export Package* option differs from the *Export as Project File* option in that copies of filesystem−based components (such as files and dialog projects) are included in the archive. Conversely, *Export as Project File* creates a project with a reference to the component's original location and does not create copies of the components. When *Export Package* is invoked, you are prompted for a name and file location for the package. Exported packages can only be imported into "container" objects in Komodo, such as projects, the Toolbox, and folders within projects and the Toolbox. See Toolbox – Exporting and Importing Toolbox Contents, Projects – Importing and Exporting Projects via Packages, or Folders – Import Contents from Package for more information.
- *Rename*: To change the name of a template (as displayed in the Project Manager or Toolbox), select this option and enter a new name.
- *Delete*: To remove a template from a project or the Toolbox, select this option. The reference to the template is deleted; the file containing the template is not deleted.
- *Properties*: Changes the title or the disk location of the template. In addition, the icon and key binding assigned to the template are accessed via the Properties dialog box. See below for instructions on altering the template's icon or key binding.

## Assigning Custom Icons to Templates

The default template icon can be replaced with custom icons. Komodo includes more than 600 icons; alternatively, select a custom image stored on a local or network drive (use 16x16−pixel images for best results).

To assign a custom icon to a template:

1. On the *Projects* or *Toolbox* tab, right−click the desired template and select *Properties*. Alternatively, click the template in the *Projects* or *Toolbox* tab, then select *Projects|template_name|Properties* or *Toolbox|template_name|Properties*.

2. In the Properties dialog box, click ***Change Icon***.
3. In the Pick an Icon dialog box, select a new icon and click ***OK***. Alternatively, click ***Choose Other***, and browse to the desired image file.
4. In the Properties dialog box for the template, click ***OK***. The custom icon is displayed next to the template.

To revert to the default icon for a selected template:

1. On the ***Projects*** or ***Toolbox*** tab, right−click the desired template and select ***Properties***.
2. Click ***Reset***, then click ***OK***. The default icon is displayed next to the template.

## Template Key Bindings

Custom key bindings can be assigned to templates stored in the Toolbox or in a Project. Use the ***Key Binding*** tab in the template's Properties to specify the keystrokes that invoke the template. See Key Bindings for Custom Components for more information.

# Open Shortcuts

Open shortcuts are references stored in projects or the Toolbox that point to directories on a local or network drive. When an open shortcut is invoked, the standard *Open File* dialog box is displayed with the contents of the directory. Open shortcuts make it quick to access deep directories that are frequently used.

Refer to Adding Components to Projects or Adding Components to the Toolbox for instructions on adding components to projects or the Toolbox. In addition, open shortcuts can be created by right–clicking a file in a project or the Toolbox and selecting *Make "Open..." Shortcut*.

## Open Shortcut Options

To access options for the selected open shortcut, do one of the following:

- *Toolbox/shortcut_name/option* or *Project/shortcut_name/option*: When an open shortcut is selected in the Project Manager or Toolbox, use the *Project* or *Toolbox* drop–down menus to access the list of options. The name of the open shortcut that is currently selected in a project or the Toolbox is displayed on the drop–down menu.
- *Context Menu*: Right–click the open shortcut in a project or the Toolbox and select the desired option.

The following options are available:

- *Open (Shortcut)*: Use this option to display the contents of the folder specified in the open shortcut.
- *Cut/Copy/Paste*: These options are used to remove components from a project or the Toolbox, or to move components between the Project Manager and the Toolbox.
- *Export as Project File*: When this option is invoked, a new project file is created that contains the open shortcut from which the option is invoked. You are prompted to provide the name of the new project file and the directory where it will be stored. To open the new project file, select *File/Open/Project*.
- *Export Package*: Open shortcuts can be archived and distributed among multiple Komodo users via "packages". Packages are compressed archive files that contain the open shortcut from which the *Export Package* option was invoked. Packages are stored in files with a ".kpz" extension, and can be opened by any archiving utility that supports `libz` (for example WinZip). The *Export Package* option differs from the *Export as Project File* option in that copies of filesystem–based components (such as files and dialog projects) are included in the archive. Conversely, *Export as Project File* creates a project with a reference to the component's original location and does not create copies of the components. When *Export Package* is invoked, you are prompted for a name and file location for the package. Exported packages can only be imported into "container" objects in Komodo, such as projects, the Toolbox, and folders within projects and the Toolbox. See Toolbox – Exporting and Importing Toolbox Contents, Importing and Exporting Projects via Packages, or Folders – Import Contents from Package for more

information.

- *Rename*: To change the name of an open shortcut, select this option and enter a new name.
- *Delete*: To remove an open shortcut from a project or the Toolbox, select this option. The open shortcut is permanently deleted.

# Open Shortcut Properties

Open shortcut properties are used to alter the destination of the shortcut or to change the open shortcut's name. The Properties dialog box is also used to assign a custom icon to an open shortcut or to assign a custom key binding. To access the Properties dialog box, right−click the open shortcut and select *Properties*.

### Assigning Custom Icons to Open Shortcuts

The default open shortcut icon can be replaced with custom icons. Komodo includes more than 600 icons; alternatively, select a custom image stored on a local or network drive (use 16x16−pixel images for best results).

To assign a custom icon to an open shortcut:

1. On the *Projects* tab or *Toolbox* tab, right−click the desired open shortcut and select *Properties*. Alternatively, click the open shortcut in the *Projects* tab or *Toolbox* tab, then select *Projects/shortcut_name/Properties* or *Toolbox/shortcut_name/Properties*.
2. In the Properties dialog box, click *Change Icon*.
3. In the Pick an Icon dialog box, select a new icon and click *OK*. Alternatively, click *Choose Other*, and browse to the desired image file.
4. In the Properties dialog box for the open shortcut, click *OK*. The custom icon is displayed next to the open shortcut.

To revert to the default icon for a selected open shortcut:

1. On the *Projects* tab or *Toolbox* tab, right−click the desired open shortcut and select *Properties*.
2. Click *Reset*, then click *OK*. The default icon is displayed next to the open shortcut.

### Open Shortcut Key Bindings

Custom key bindings can be assigned to open shortcuts stored in the Toolbox or in a Project. Use the *Key Binding* tab in the open shortcut's Properties to specify the keystrokes that invoke the open shortcut. See Key Bindings for Custom Components for more information.

# URL Shortcuts

URLs are components within a project or the Toolbox that are used to store frequently used URL addresses. These URL shortcuts can be opened in an external browser (as specified in the Web and Browser preference) or displayed in a new tab within Komodo.

Refer to Adding Components to Projects or Adding Components to the Toolbox for instructions on adding components to projects or the Toolbox. In addition, URL shortcuts can be created by dragging a URL from a browser address bar or from a Komodo tab onto a project or Toolbox. URLs must be preceded by "http://".

To open a URL, double−click the URL name (which opens the URL in an external browser), use the assigned key binding, or right−click the URL and select *Open URL in browser* or *Open URL in tab*.

# URL Shortcut Options

To access options for the selected URL shortcut, do one of the following:

- *Toolbox|shortcut_name|option* or *Project|shortcut_name|option*: When a URL shortcut is selected in the Project Manager or Toolbox, use the *Project* or *Toolbox* drop−down menus to access the list of options. The name of the URL shortcut that is currently selected in a project or the Toolbox is displayed on the drop−down menu.
- *Context Menu*: Right−click the URL shortcut in a project or the Toolbox and select the desired option.

The following options are available:

- *Open URL in browser*: Use this option to launch the default web browser (as specified in the Web and Browser preference) and display the stored URL.
- *Open URL in tab*: Use this option to display the stored URL in a tab in the Komodo Editor Pane.
- *Cut/Copy/Paste*: These options are used to remove the URL shortcut from a project or the Toolbox, or to move URL shortcuts between the project and the Toolbox (and vice versa).
- *Export as Project File*: When this option is invoked, a new project file is created that contains the URL shortcut from which the option is invoked. You are prompted to provide the name of the new project file and the directory where it will be stored. To open the new project file, select *File|Open|Project*.
- *Export Package*: URLs can be archived and distributed among multiple Komodo users via "packages". Packages are compressed archive files that contain the URL shortcut from which the *Export Package* option was invoked. Packages are stored in files with a ".kpz" extension, and can be opened by any archiving utility that supports `libz` (for example WinZip). The *Export Package* option differs from the *Export as Project File* option in that copies of filesystem−based components (such as files and dialog projects) are included in the archive. Conversely, *Export as Project File* creates a project with a reference to the component's original location and does not create copies of the components. When *Export Package* is invoked, you are prompted for a name and file location for the package. Exported packages can only be imported into "container"

objects in Komodo, such as projects, the Toolbox, and folders within projects and the Toolbox. See Toolbox – Exporting and Importing Toolbox Contents, Projects – Importing and Exporting Projects via Packages, or Folders – Import Contents from Package for more information.

- *Rename*: To change the name of a URL shortcut, select this option and enter a new name.
- *Delete*: To remove a URL shortcut from a project or the Toolbox, select this option. The URL shortcut is permanently deleted.

# URL Shortcut Properties

URL shortcut properties are used to alter the address of the URL or to change the URL shortcut's name. The Properties dialog box is also used to assign a custom icon to a URL shortcut or to assign a custom key binding. To access the Properties dialog box, right–click the URL shortcut and select *Properties*.

## Assigning Custom Icons to URL Shortcuts

The default URL shortcut icon can be replaced with custom icons. Komodo includes more than 600 icons; alternatively, select a custom image stored on a local or network drive (use 16x16–pixel images for best results).

To assign a custom icon to a URL shortcut:

1. On the *Projects* tab or *Toolbox* tab, right–click the desired URL shortcut and select *Properties*. Alternatively, click the URL shortcut in the *Projects* tab or *Toolbox* tab, then select *Projects/macro_name/Properties* or *Toolbox/macro_name/Properties*.
2. In the Properties dialog box, click *Change Icon*.
3. In the Pick an Icon dialog box, select a new icon and click *OK*. Alternatively, click *Choose Other*, and browse to the desired image file.
4. In the Properties dialog box for the URL shortcut, click *OK*. The custom icon is displayed next to the URL shortcut.

To revert to the default icon for a selected URL shortcut:

1. On the *Projects* tab or *Toolbox* tab, right–click the desired URL shortcut and select *Properties*.
2. Click *Reset*, then click *OK*. The default icon is displayed next to the URL shortcut.

## URL Shortcut Key Bindings

Custom key bindings can be assigned to URL shortcuts stored in the Toolbox or in a Project. Use the *Key Binding* tab in the URL shortcut's Properties to specify the keystrokes that invoke the URL shortcut. See Key Bindings for Custom Components for more information.

# Run Commands

Run commands are operating system commands run from within Komodo. Use the Run Command dialog box to interact with the system command line or shell while editing or debugging files in Komodo. Besides making it easy to run simple and complex custom commands from within Komodo, the Run Command dialog box can insert the results of shell commands into a document in the Editor Pane, or pass the contents of a document to the system command line or shell.

To view examples of run commands, see the "Samples" folder in the Toolbox.

Run commands can be stored for re−use in a project or the Toolbox, where they can be assigned key bindings.

Access the last ten commands executed in the Run Command dialog box by selecting *Tools/Recent Commands*, or using the 'Alt'+'T'+'M', *number* default keyboard shortcut (where "number" represents the sequence number of the previous command). The prefixes [i], [l] and [il] indicate that *Insert output*, *Pass selection as input* or both were selected with the original command.

*Tutorial*

- Run Commands

*Feature Showcase*

- Google Run Command

## Creating Run Commands

To create a run command, select *Tools/Run Command*. The Run Command dialog box is displayed. Alternatively, invoke the Run Command dialog box from the Project Manager or the Toolbox by selecting *Add/New Command* from the *Project* or *Toolbox* menu.

The Run Command dialog box can be toggled between its "simple" or "advanced" form by clicking the *More/Less* button.

## Simple Run Commands

This section describes the components of the Run Command dialog box that are displayed when the advanced commands are hidden (via the *More/Less* button. See Advanced Run Commands for information about the advanced fields.

- *Run*: Enter the command to run.
- *Interpolation Shortcut*: Click the arrow button to the right of the *Run* field to access a drop−down list of interpolation shortcuts. When an interpolation shortcut is selected, it is inserted at the current cursor position in the *Run* field. Windows users should enclose shortcuts (except for the `%(browser)` shortcut) in quotation marks to ensure that spaces in filenames or file paths are interpreted correctly by the system shell.
- *Pass selection as input*: If this check box is selected, the text currently highlighted in the editor is passed to the command in the *Run* field. For example, if the *Run* field contains `grep`

`myvar`, each line containing "myvar" in the text selected in the editor is returned.

- *Insert output*: If this check box is selected, the results of the command are inserted at the cursor position in the current document.
- *Add to Toolbox*: If this check box is selected, the command is saved in the Toolbox.

## Advanced Run Commands

Click the *More* button in the Run Command dialog box to display advanced options. The following options are available:

- *Start in*: Enter the directory where the command should be run, or click the *Browse* button to navigate the filesystem. Click the arrow button to the right of the *Start in* field to select interpolation shortcuts pertinent to the *Start in* setting. Interpolation shortcuts are inserted at the current cursor position in the *Run* field.
- *Run in*: Specify the environment in which the command should be run. The options are:
  - *Command Output Tab*: The command is run in Komodo's Bottom Pane.
  - *New Console*: The command is run in a new shell or command window.
  - *No Console (GUI Application)*: The command launches the specified application without displaying output in a shell or on the *Command Output* tab.
- *Do not open output pane*: If this check box is selected, the Bottom Pane containing the *Command Output* tab does not automatically open when the command is run. To manually view the Bottom Pane, select *View*/*Command Output*. (This option is only accessible if the *Run in* field is set to *Command Output* tab.)
- *Parse output with*: If this check box is selected, the field to the right is used to enter a regular expression that parses the output. (This option is only accessible if the *Run in* field is set to *Command Output* tab.)
- *Show parsed output as list*: If output parsing is configured, select *Show parsed output as list* to display the output in list format on the *Command Output* tab. (This option is only accessible if the *Run in* field is set to *Command Output* tab.)
- *Environment Variables*: Use the *Environment Variables* section of the dialog box to configure new environment variables or change the value of existing environment variables for the duration of the run. To add or alter an environment variable, click *New* and configure the following values:
  - *Variable Name*: Enter a name for the variable.
  - *Variable Value*: Enter a value for the variable.
  - *Interpolation Shortcut*: Click the arrow button to the right of the *Variable Value* field to insert an interpolation shortcut pertinent to the *Variable Value* setting. The interpolation shortcut is inserted at the current cursor position in the *Variable Value* field.
  - *Add Path*: Click this button to insert a directory as the variable value.
- *Save advanced options as defaults*: If this check box is selected, the current settings are stored as the defaults for the Run Command dialog box.

## Command Output Tab

By default, the commands run in the ***Command Output*** tab on Komodo's Bottom Pane. (Use the ***Run in*** field to run the command in a new shell window, or to run a graphical application without a console.)

If the command prompts for input, enter it directly on the ***Command Output*** tab. Output written to "stderr" (standard error output) is displayed in red at the top of the tab. Click the ***Close*** button at the top right of the ***Command Output*** tab to terminate a running command. Click the ***Toggle Raw/Parsed Output View*** button to jump from parsed results to raw output and vice versa. (Parsing is enabled and configured via the ***Parse output with*** field.)

# Storing Run Commands in a Project or the Toolbox

To add a run command to the Toolbox, select ***Add to Toolbox*** in the Run Command dialog box. Run commands can also be added to projects or the Toolbox via the methods described in Storing Run Commands in a Project or the Toolbox.

To run a command stored in the Toolbox or in a project, double–click the run command's name, use the assigned key binding, or right–click the run command and select ***Run***.

To access run command options for the selected run command, right–click the run command's name. The options are as follows:

- ***Run Command***: Execute the stored run command.
- ***Cut/Copy/Paste***: These options are used to remove the run command from a project or the Toolbox, or to move a run command between a project and the Toolbox (and vice versa).
- ***Export as Project File***: When this option is invoked, a new project file is created that contains the run command from which the option is invoked. You are prompted to provide the name of the new project file and the directory where it is stored. To open the new project file, select ***File|Open|Project***.
- ***Export Package***: Run commands can be archived and distributed among multiple Komodo users via "packages". Packages are compressed archive files that contain the run command from which the ***Export Package*** option was invoked. Packages are stored in files with a ".kpz" extension, and can be opened by any archiving utility that supports `libz` (for example WinZip). The ***Export Package*** option differs from the ***Export as Project File*** option in that copies of filesystem–based components (such as files and dialog projects) are included in the archive. Conversely, ***Export as Project File*** creates a project with a reference to the component's original location and does not create copies of the components. When ***Export Package*** is invoked, you are prompted for a name and file location for the package. Exported packages can only be imported into "container" objects in Komodo, such as projects, the Toolbox, and folders within projects and the Toolbox. See Toolbox – Exporting and Importing Toolbox Contents, Projects – Importing and Exporting Projects via Packages, or Folders – Import Contents from Package for more information.
- ***Rename***: To change the name of a run command, select this option and enter a new name.

- *Delete*: To remove a run command from a project or the Toolbox, select this option. The run command is permanently deleted.

# Run Command Properties

To access the properties of a run command stored in a project or the Toolbox, right−click the run command and select *Properties*. The Properties dialog box contains all the elements of the Run Command dialog box, and is therefore used for editing stored run commands. In addition, the Properties dialog box is used to assign a custom icon to the run command, and to assign a custom key binding.

### Assigning Custom Icons to Run Commands

The default run command icon can be replaced with custom icons. Komodo includes more than 600 icons; alternatively, select a custom image stored on a local or network drive (use 16x16−pixel images for best results).

To assign a custom icon to a run command:

1. On the *Projects* tab or *Toolbox* tab, right−click the desired run command and select *Properties*. Alternatively, click the run command in the *Projects* tab or *Toolbox* tab, then select *Projects/runcommand_name/Properties* or *Toolboxruncommand_name/Properties*.
2. In the Properties dialog box, click *Change Icon*.
3. In the Pick an Icon dialog box, select a new icon and click *OK*. Alternatively, click *Choose Other*, and browse to the desired image file.
4. In the properties dialog box for the run command, click *OK*. The custom icon is displayed next to the run command.

To revert to the default icon for a selected run command:

1. On the *Projects* tab or *Toolbox* tab, right−click the desired run command and select *Properties*.
2. Click *Reset*, then click *OK*. The default icon is displayed next to the run command.

### Run Command Key Bindings

Custom key bindings can be assigned to run commands stored in the Toolbox or in a Project. Use the *Key Binding* tab in the run command's Properties to specify the keystrokes that invoke the run command. See Key Bindings for Custom Components for more information.

# Custom Toolbars and Menus

Custom toolbars and menus are used to extend Komodo's default menus and toolbars with menus and toolbars containing custom items. Any component that can be stored in a project or the Toolbox can be stored in a custom toolbar or menu. A custom toolbar might contain frequently used run commands and snippets; a custom menu might contain folders with the contents of a local filesystem.

Custom toolbars and menus are created and configured within the Project Manager and/or the Toolbox. Custom toolbars and menus contained in projects are only displayed when the projects are open in the Project Manager; custom toolbars and menus contained in the Toolbox are always displayed.

## Creating Custom Toolbars and Menus

1. On the **Project** or **Toolbox** menu, select **Add|New Custom Toolbar** or **Add|New Custom Menu**. Enter a name for the new component.
2. Copy and paste or drag and drop the item(s) to be included on the toolbar or menu onto the icon created in the previous step. Alternatively, right−click the custom menu or toolbar name and select **Add**. For a complete description of adding components to containers (such as projects and custom menus and toolbars), see the Projects section of the documentation. New toolbars are displayed alongside the default Komodo toolbars and can be accessed via the **View|Toolbars** menu. New menus are displayed to the right of the default Komodo menus.

## Custom Menu and Toolbar Options

To access options for a custom menu or toolbar, do one of the following:

- **Toolbox|menuortoolbar_name|option** or **Project|menuortoolbar_name|option**: When a custom menu or toolbar is selected in the Project Manager or Toolbox, use the **Project** or **Toolbox** menus to access the list of options. The name of the custom menu or toolbar that is currently selected in a project or the Toolbox is displayed on the menu.
- **Context Menu**: Right−click the custom menu or toolbar in a project or the Toolbox and select the desired option.

The following options are available:

- **Export as Project File**: When this option is invoked, a new project file is created that contains the custom menu or toolbar from which the option is invoked. You are prompted to provide the name of the new project file and the directory where it will be stored. To open the new project file, select **File|Open|Project**.
- **Export Package**: Snippets can be archived and distributed among multiple Komodo users via Packages. Packages are compressed archive files that contain the snippet from which the **Export**

*Package* option was invoked. Packages are stored in files with a ".kpz" extension, and can be opened by any archiving utility that supports `libz` (for example WinZip). The *Export Package* option differs from the *Export as Project File* option in that copies of filesystem−based components (such as files and dialog projects) are included in the archive. Conversely, *Export as Project File* creates a project with a reference to the component's original location and does not create copies of the components. When *Export Package* is invoked, you are prompted for a name and file location for the package. Exported packages can only be imported into "container" objects in Komodo, such as projects, the Toolbox, and folders within projects and the Toolbox. See Toolbox − Exporting and Importing Toolbox Contents, Projects − Importing and Exporting Projects via Packages, or Folders − Import Contents from Package for more information.

- *Rename*: To change the name of a custom menu or toolbar, select this option and enter a new name.
- *Cut/Copy/Paste*: These options are used to remove the custom menu or toolbar from a project or the Toolbox, or to move custom menu or toolbars between the project and the Toolbox (and vice versa).
- *Add*: Use this option to add components to the selected custom menu or toolbar. All components can be added to custom menu or toolbars in the same manner they are added to projects. Refer to the individual component documentation, or the project documentation for more information.
- *Delete*: To remove a custom menu or toolbar from a project or the Toolbox, select this option. The custom menu or toolbar is permanently deleted.

## Custom Menu and Toolbar Properties

Custom menu or toolbar properties are used to alter the name of the custom menu or toolbar, or change the custom menu or toolbar's display order. To access the Properties dialog box, right−click the custom menu or toolbar and select *Properties*.

Custom menus are displayed between the default *Tools* and *Window* menus. If multiple custom menus are in effect, the display order depends on the menu's *Priority* setting. New menus have a default priority of 100; alter the priority of custom menus to control the left−to−right order of display.

Custom toolbars are displayed to the right of default Komodo toolbars. If necessary, a new row is created for their display. If multiple custom toolbars are in effect, the display order depends on the toolbar's *Priority* setting. New toolbars have a default priority of 100; alter the priority of custom toolbars to control the left−to−right order of display.

To assign a letter to be used in combination with the 'Alt' key for menu access, enter a shortcut letter in the *Menu Access Key* field. If the letter is already assigned to a Komodo core function, you are prompted to enter a different letter.

# Debugging Programs

The Komodo debugger is a tool for analyzing programs on a line–by–line basis, monitoring and altering variables, and watching output as it is generated. Debugging features include:

- breakpoint and spawnpoint control
- remote debugging
- stepping
- watching variables
- viewing the call stack
- sending input
- adding command–line arguments
- interactive shell

The sections that follow contain general information about the debugger that is applicable to each language. Komodo provides debugging support for Perl, Python, PHP, XSLT and Tcl. For information about configuring languages and language–specific debugger functions, see:

- Debugging Perl
- Debugging Python
- Debugging PHP
- Debugging XSLT
- Debugging Tcl

*Notes*

- Perl Debugging on Windows 98/Me requires ActivePerl Build 623 or higher. Restart your machine after you install ActivePerl to be sure that your system's PATH variable is updated with the location of Perl.
- Be sure you meet the software prerequisites for debugging, as described in the Installation Guide.

## Starting the Debugger

To start the debugger, do one of the following:

- *Debug Toolbar*: Select *Go/Continue* or *Step In*.
- *Keyboard*: Use the associated key binding.
- *Debug Menu*: Click *Go/Continue* or *Step In*.

By default, the Debugging Options dialog box is displayed (unless the debugger preference has been configured to start without displaying the dialog box). To override the debugger preference, hold down the 'Ctrl' key while invoking the key binding for starting the debug session. (Select *Help/List Key Bindings* to view the current key bindings; use the key bindings preference to configure custom key

*Tutorials*

- Perl tutorial
- Python tutorial
- PHP tutorial
- Tcl tutorial
- XSLT tutorial

*Feature Showcases*

- conditional breakpoints
- XSLT debug view

bindings.) Alternatively, the 'Ctrl' key can be used to suppress the display of the ***Debugging Options*** dialog box.

If multiple files are open in the Editor Pane, the program that is currently displayed is debugged. If no breakpoints are set, ***Go/Continue*** causes the debugger to run to the end without stopping. ***Step In*** moves through the program one line at a time.

If the Bottom Pane is hidden, Komodo automatically shows it.

To run a program without debugging, do one of the following:

- ***Debug Menu***: Select ***Run without debugging***.
- ***Keyboard***: Use the associated key binding.

To run a program to the current cursor position, do one of the following:

- ***Debug Menu***: Select ***Run to Cursor***.
- ***Keyboard***: Use the associated key binding.

## Multi–Session Debugging

Komodo supports the concurrent debugging of multiple applications, or multi–session debugging. With multi–session debugging, Komodo debugs more than one project at a time, regardless of the supported languages used in the programs being debugged.

When debugging multiple sessions, each session has a unique ***Debug*** tab (located in the Bottom Pane) for controlling the debug actions specific to that process. A ***Debug*** tab is created each time a new debugger session is started. To close a ***Debug*** tab, click the X button at the top right corner.

To start multiple debugging sessions, do one of the following:

- ***Debug Menu***: Click ***Start New Session***.
- ***Keyboard***: Use the associated key binding.

# Debugging Options

When the debugger is invoked, the Debugging Options dialog box is displayed. Use this to configure the system environment, command–line arguments, CGI environment, and other debugging options.

Not all of the debugging options described below apply to all languages. The available tabs and fields depend on the interpreter used to debug the file. The interpreter is determined by the File Associations configured for the active file in the Editor Pane.

To suppress the display of the Debugging Options dialog box, hold down the 'Ctrl' key while clicking the desired debugging button on the Debug Toolbar, or use the desired keyboard shortcut. Change the default display by selecting *Skip debug options dialog* from *Edit|Preferences|Debugger*.

## Global Options

These options are displayed regardless of which configuration tabs are available.

- *Simulate CGI Environment*: Select this check box to display two additional CGI option tabs – [CGI Environment](#) and [CGI Input](#).
- *Debug in separate console*: Select this check box to display the debug process in a separate console window rather than the *Output* tab. As applicable, the console window displays program output and prompts for program input.

## General Tab

- *Interpreter Arguments*: As required, enter command line options and arguments for the interpreter in this field. Use the *Shortcut* button to the right of the input field to select common language–specific options.
- *Script*: Enter the name of the script to be debugged. By default, this field contains the full path and name of the program displayed in the Editor pane. When manually specifying a script, UNC or SMB paths (which identify remote machines on a local area network via the "\\" prefix) are not supported. Instead, map the network share to a drive letter (Windows) or mount the share on the filesystem (Unix).
- *Script Arguments*: As required, enter arguments for the script in this field as they would appear on the command line. Multiple Arguments must be separated with spaces. If the *Simulate CGI Environment* box is selected, and CGI Input variables of the type *GET* are set, the contents of this field are ignored.
- *Directory*: Specify the directory to start the program in. If unset, the program starts in the directory where it resides.
- *Select the input XML file*: (XSLT only) Specify the name and location of the input XML file.
- *Select the interpreter to use for debugging*: (PHP and Tcl only) For Tcl programs, select *tclsh* or the *wish* interpreter, depending on whether you are debugging a console or a GUI application. For PHP programs, select the CLI (Command Line Interface) or CGI (Common Gateway Interface) interpreter. These selections reference the interpreters configured under *Edit|Preferences...* in the [Language Configuration](#) section.
- *Select the directory that contains the php.ini file*: (PHP only) If more than one version of PHP exists on the system, specify the directory that contains the php.ini file you wish to use.
- *Disable Output Buffering (PHP only)*: Output from the PHP interpreter is not buffered (it is displayed as it occurs) if this option is enabled. This option has no effect when *Simulate CGI Environment* is selected. To disable output buffering in CGI emulation mode, comment out the `output_buffering` setting in *php.ini* with a ";" character, or set it to "`off`".

- *Enable Implicit Flush (PHP only)*: The PHP output layer flushes itself automatically after every output block. If this option is not enabled, output is buffered by the operating system and is flushed periodically by the operating system, or when the application is finished. This option has no effect when *Simulate CGI Environment* is selected.

## Environment Tab

The *Environment* tab displays all environment variables set on the system. Use this tab to add new variables or change the values of existing variables for the duration of the debug session. The *Default Environment Variables* pane displays environment variables that have been declared on your system. The *User Environment Variables* pane displays environment variables set in the saved configuration which override the *Default Environment Variables*.

Change variables by adding a new variable with the same name and a new value. These changes have no effect outside of the Komodo debugger and are stored in each saved configuration.

- *To Add New Variables*: Click *New* and enter the *Variable Name* and *Variable Value* in the Environment Variable dialog box. To add one or more directories to the *Variable Value* field, click *Add Path* and navigate to the desired directory.
- *To Edit Existing Variables*: Select the variable, click *Edit*, then change as desired. This creates a new variable with the same name and the desired value. (User Environment Variables take precedence over Default Environment Variables.)
- *To Delete a Variable*: Select the variable from the *User Environment Variables* pane and click *Delete*. *Default Environment Variables* cannot be deleted, but can be set to an empty value.

## CGI Environment Tab

The *CGI Environment* tab is only displayed if the *Simulate CGI Environment* check box is selected on the General tab. It displays CGI Environment Variables commonly configured on a web server. Use this tab to alter existing variables and add new variables. Variable changes have no effect outside of the Komodo debugger and are stored in each saved configuration.

- *To Add New Variables*: Click *New*, and enter the *Variable Name* and *Variable Value* in the Environment Variable dialog box. To add one or more directories to the *Variable Value* field, click *Add Path* and navigate to the desired directory.
- *To Edit Existing Variables*: Select the variable, click *Edit*, then change as desired. This creates a new variable with the same name and the desired value. (*User CGI Environment Variables* take precedence over *Default CGI Environment Variables*.)
- *To Delete a Variable*: Select the variable from the *User CGI Environment Variables* pane and click *Delete*.

## CGI Input Tab

The *CGI Input* tab is only displayed if the *Simulate CGI Environment* check box is selected on the *Global Options* tab. It is used to configure the CGI form type and variables for the purpose of simulating CGI input. Note that Komodo's CGI emulation does not generate HTTP Request Headers; rather, it executes the CGI directly by emulating a web server environment.

- *Request Method*: Select the request method that has been assigned to the form in the CGI program.
- *Post Type*: Select the format in which data is sent from the browser to the server.

Use the *Request Variable* section of the dialog box to create variables that are processed by your CGI program. These variables are displayed in the *Browser Arguments* section of the dialog box.

- *Type*: Specify the type of input associated with the variable (GET, POST, cookie or file).
- *Name*: Enter the variable name as specified in the CGI program.
- *Value*: Enter the value for the variable specified in the *Name* field. To provide a directory path and file, click the *Browse Files* button, select the desired file and click *Add*. (To accommodate file uploads, select *Multipart* as the form's *POST* method.)

*To alter variables:* Click on the desired variable in the *Browser Arguments* section of the dialog box, make changes in the *Type*, *Name* and *Value* fields, then click *Update*.

*To delete variables:* click on the desired variable in the *Browser Arguments* section of the dialog box, and click *Delete*.

## Storing Debug Configurations

Debugging options can be saved as "named configurations" in the *Debug Configuration* panel. To save the current configuration:

1. Click *New...*
2. Enter a unique *Configuration Name*.
3. Click *OK*.

Existing saved configurations can be selected from the drop−down list. If you wish to delete a saved configuration, select it from the list and click *Delete*.

If the file being debugged is part of a project that is currently open, these preferences are saved in that project. If not, this configuration is automatically saved as part of the file's Properties and Settings (although they cannot be altered via the file's *Properties* dialog box).

# Breakpoints and Tcl Spawnpoints

Breakpoints are set at lines in the program where you want program execution to pause. Enabled breakpoints appear as solid red circles in the left margin of the Editor pane and are also listed on the *Breakpoints* tab during debugging. Disabled breakpoints appear as white circles with a red outline. Double–clicking on an enabled or disabled breakpoint from the *Breakpoints* tab opens the associated file in the Editor Pane and shifts focus to the line number for that break location.

Spawnpoints are set at points in a Tcl script where you want an external application to execute (spawn). When a spawnpoint is encountered during the debugging process, Komodo configures the spawned application to start as a new debugger session. Both the initial and spawned debugger sessions run concurrently. Enabled spawnpoints appear as solid green arrows in the left margin of the Editor pane and are also listed on the *Breakpoints* tab during debugging. Disabled spawnpoints appear as white arrows with a green outline. Double–clicking an enabled or disabled spawnpoint from the *Breakpoints* tab opens the associated file in the Editor pane and shifts focus to the line number coinciding with that spawnpoint location.

## Breakpoint and Spawnpoint Management

Breakpoints and spawnpoints can be monitored and managed on the *Breakpoints* tab in the Bottom pane (displayed during debugging or invoked by selecting *View*/*Tabs*/*Command Output*). This tab lists all breakpoints and spawnpoints set in the program. Use the *Breakpoints* tab to:

- toggle breakpoints
- toggle spawnpoints
- go to source code
- set breakpoint properties

### Toggling Breakpoints

Breakpoints and Spawnpoints can be toggled between Enabled, Disabled and Deleted. To toggle a breakpoint, do one of the following:

- *Breakpoint Margin*: Click on the line you wish to break at once to enable a breakpoint, a second time to Disable it, and a third time to Delete it.
- *Debug Menu*: Click *Enable/Disable Breakpoint* once to enable a breakpoint on the current line, a second time to Disable it, and a third time to Delete it.
- *Keyboard:* Press '*F9*' once to enable a breakpoint on the current line, a second time to Disable it, and a third time to Delete it.

To create a new breakpoint in the *Breakpoints* tab:

1. Click the *New* button and then select *New Breakpoint* or right−click in the Breakpoints pane and select *Add/New Breakpoint* on the context menu.
2. The following Breakpoint Properties are required:
    ♦ Language: By default, the language of the program being debugged.
    ♦ *File*: The location of the file where the breakpoint is being set.
    ♦ *Line*: The line number in the file where the spawnpoint is to be set.
    ♦ *Enable*: Select the check box to enable the breakpoint.
3. Click *OK*.

To delete a breakpoint in the *Breakpoints* tab, do one of the following:

- Select the breakpoint and click the *Delete Breakpoint* button
- Right−click on the breakpoint and select *Delete*.

To clear or remove multiple breakpoints, do one of the following:

- *Debug Menu*: Click *Clear All Breakpoints*.
- *Breakpoint Tab*: Click the *Delete All Breakpoints* button.
- *Keyboard:*: Use the associated key binding.

To disable or enable all breakpoints:

- On the *Breakpoints* tab, click the *Disable/Enable All Breakpoints* button. All breakpoints are disabled if previously enabled, or enabled if previously disabled.

## Toggling Spawnpoints

To add a Tcl spawnpoint, use the *Breakpoints* tab:

1. Click the *New* button and then select *New Tcl Spawnpoint* or right−click in the Breakpoints list and select *Add/New Tcl Spawnpoint* on the context menu.
2. The following properties are configurable in the Spawnpoint Properties dialog box:
    ♦ *Language*: Tcl
    ♦ *File*: The location of the file where the spawnpoint is to be set (for example,
      `C:\tcl_tutorial\dlg_tcl_check.tcl`).
    ♦ *Line*: The line number in the file where the spawnpoint is to be set.
    ♦ *Enable*: Select the check box to enable the spawnpoint. Deselect the check box to disable the spawnpoint.
3. Click *OK*.

To delete a spawnpoint in the *Breakpoints* tab, do one of the following:

- Select the spawnpoint and click the *Delete Breakpoint* button
- Right−click on the spawnpoint and select *Delete*.

To clear or remove multiple spawnpoints, do one of the following:

- *Debug Menu*: Click *Clear All Breakpoints*.
- *Breakpoint Tab*: Click the *Delete All Breakpoints* button.
- *Keyboard:*: Use the associated [key binding](#).

To disable or enable all spawnpoints:

- On the *Breakpoints* tab, click the *Disable/Enable All Breakpoints* button. All spawnpoints are disabled if previously enabled, or enabled if previously disabled.

*Note:* Breakpoints and spawnpoints added or modified while a program is running are not necessarily updated in the breakpoint manager. To add breakpoints while debugging, interrupt the debugging session using the *Break* button to ensure that the new breakpoint is properly updated.

## Go to the Source Code

To open the source code in the Editor Pane at the line number where the breakpoint or spawnpoint is set, do one of the following:

- *Breakpoints Tab*: Double−click the breakpoint to view the associated source code.
- *Breakpoints Tab*: Select the desired breakpoint and click the *Go to the Source Code* button.

## Breakpoint Properties

When adding or editing a breakpoint in the *Breakpoint* tab, a *Breakpoint Properties* dialog box appears. This dialog box contains a tab for each available breakpoint type. Change the breakpoint type by switching to a different tab.

Each tab is split into two parts, separated by a horizontal line. The top section contains configuration items that are required; the bottom section contains configuration options that are optional. The last item on this tab is the Enable checkbox.

- *Language*: The language of the file where the breakpoint is to be set.
- *File*: The line number on which to break.
- *Condition*: Evaluate some code and break if it evaluates to true. For example, `0==0`, as the condition would always evaluate to true and cause the debugger to break. The condition should be specified in the syntax of the language selected in the *Language* drop−down list.
- *Watch*: Break when the value of the specified variable (or expression) is set or changed.
- *Function Call*: Break after the specified function is called.
- *Function Return*: Break after the specified function has finished executing.
- *Exception*: Break when the specified exception is caught.
- *Line*: Break on the specified line.

- *Hit Counts*: Break when the condition specified in any of the above has been met a certain number of times. Each time the debugger engine reaches a breakpoint, it adds one to the count. It then looks at the hit count setting to see if the evaluation allows for a break at that moment. There are two configuration items related to hit counts, the condition and the count. There are three types of conditions:
  - ♦ Break when hit count is greater than or equal to
  - ♦ Break when hit count is equal to
  - ♦ Break when hit count is a multiple of

For example:

1. Set a breakpoint on line 2 of a script.

```
for 1 in range(256):
print 'hello'
```

2. Define a hit condition 'Break when hit count is a multiple of'.
3. Enter the value 5. The debugger breaks every 5th time it passes the line with the print statement.

Not all breakpoint types are supported for all languages. The following table shows breakpoint support by language:

| *Type* | *Tcl* | *Perl* | *PHP* | *XSLT* | *Python* |
|---|---|---|---|---|---|
| Line Number | Yes | Yes | Yes | Yes | Yes |
| Function Call | No | Yes | Yes | Yes | Yes |
| Function Return | No | Yes | Yes | Yes | Yes |
| Exception | No | No | No | No | Yes |
| Conditional | Yes | Yes | No | Yes | Yes |
| Watch | Yes | Yes | No | No | Yes |

**Forcing a Break**

Use the *Break Now* function to stop debugging an application at the current execution point, and then continue debugging from that point. For example, use this control when debugging applications running long processes. To force a break while debugging an application, do one of the following:

- *Debug Menu*: Select *Break Now*.
- *Debug Toolbar*: Click the *Break Now* button.

# Remote Debugging

Remote debugging is the process of debugging programs locally while they execute on remote machines. This is useful for debugging applications in the environments where they would normally be run (e.g. CGI programs in a live web server setting), and for systems built using client/server architecture.

Komodo can be set to listen for remote debugger sessions continuously. Additionally, you can set remote debugger preferences and check listener status of the current listener configuration. For instructions on configuring specific languages for remote debugging, see:

- Debugging Perl
- Debugging Python
- Debugging PHP
- Debugging XSLT
- Debugging Tcl

## Listen for Remote Debugger

To toggle continuous listening for remote debugging, do one of the following:

- *Debug Menu*: select *Listen for Remote Debugger*.
- *Keyboard*: Use the associated key binding.

*Note:* A check mark appears when *Listen for Remote Debugger* is enabled. Otherwise, this feature is disabled.

## Check Listener Status

To check the status and current configuration of the Komodo debugger:

1. On the *Debug* menu, select *Listener Status*. The *Debugger Listener* status screen appears.
2. Click *OK* after reviewing listener status, or use the associated key binding.

## Multi−User Debugging

When multiple users are running Komodo session, configure Komodo's *Debugger Connection Options* to listen for debug connections on port "0" (see Set Debugger Preferences). The system provides

Komodo with a unique port each time Komodo is started, allowing multiple users on the system to debug applications simultaneously. In remote debugging, this requires the remote debugger application to be manually set to connect on the system−allocated port unless the *Debugger Proxy* is used.

## Debugger Proxy

Remote debugger processes can communicate with Komodo through the *DBGP proxy* (debugger protocol proxy). The proxy allows Komodo to use a system−allocated listener port for debugging without the user having to manually configure the same port number on the remote debugger. This is useful for running multiple remote debugging sessions and on networks where a remote debugging process can not connect to Komodo directly. The proxy can run on the local machine, the remote machine, or a separate machine.

A typical DBGP Proxy connection is established as follows:

1. Komodo contacts the DBGP proxy and identifies itself with:
2.     ♦ *Hostname or IP*: The hostname or IP address of the machine Komodo is running on. This is set to `localhost` or 127.0.0.1 if the debugger is running locally.
   ♦ *Port Number*: The port configured in *Preferences|Debugger* or the system−assigned port.
   ♦ *Proxy Key*: The *Proxy Key* configured in *Preferences|Debugger*. If unset, Komodo will use the USER or USERNAME environment variable value.
3. The DBGP Proxy stores this information.
4. The remote debugging process contacts the DBGP Proxy, providing an IDE Key which corresponds to the Proxy Key specified in Komodo. By default, this connection happens on port 9000 but can be configured to use another port (see language−specific debugging instructions and "−d" option below).
5. DBGP Proxy uses the IDE Key value to match the connection to the appropriate instance of Komodo.
6. The remote debugger connects to Komodo on the system−assigned or user−specified port.

To start the proxy on Windows:

```
cd <Komodo installation directory>\python\dbgp
dbgpProxy
```

To start the proxy on Linux or Solaris (requires Python 2.2 or later):

```
cd /<Komodo installation directory>/python/dbgp
python dbgpProxy.py
```

The following options are available:

- *−d <hostname:port>*: Listener port for debugger processes.
- *−i <hostname:port>*: Listener port for Komodo instances.

- *–l <log_level>*: Logging level. Logging is dumped to `stout` and can be set to CRITICAL, ERROR, WARN, INFO or DEBUG.

*Example*

If you are debugging scripts on a remote web server that cannot connect to Komodo directly because of a firewall, you can run dbgpProxy on an intermediary server (e.g. a gateway) which can connect to Komodo and the web server on specified ports. The three servers in this example are:

1. *workstation*: The machine running Komodo. The following preferences are set:
   - ♦ *Listen for Remote Debugger* is enabled.
   - ♦ *Enable Debugger Proxy* is selected.
   - ♦ *Listen for debug connections on port* is set to '0' (use system–assigned port)
   - ♦ *Proxy Listener Address* is set to `gateway:9001`
   - ♦ *Proxy Key* is set to "jdoe"
   *Debug/Listener Status* displays a system–assigned *Host Port* of 37016.

2. *gateway*: A gateway server with access to the internal and external networks. The proxy is running with the following options:

   ```
   dbgpProxy –i gateway:9001 –d gateway:9000
   ```
3. *webserver*: The machine running a Python CGI script called test.py.

The debugging process on 'webserver' is launched with the following command:

```
python dbgpClient.py –d gateway:9000 –k "jdoe" test.py
```

The remote debugger running on 'webserver' (dbgpClient.py in this case) connects to the proxy (dbgpProxy.py) running on 'gateway'. The proxy uses the *IDE Key* "jdoe" to connect the debugger process to the Komodo instance listening with a *Proxy Key* of "jdoe". The proxy continues to communicate with the remote debugger on port 9000, but routes the debugging session to Komodo on port 37016.

# Sending Input to the Program

When a program prompts for input, enter the desired input in the console window or *Output* tab (depending on the Debugging Options configuration), and press *Enter* to continue.

# Using Debugger Commands

## Debugger Command Description

This table lists common tasks and their Komodo commands.

| *To do this* | *Press this* |
| --- | --- |
| **Run** a program<br><br>The debugger runs until the program ends. | • **Debug Menu**: Select **Run Without Debugging**<br>• **Keyboard**: Press 'F7' |
| **Start** the debugger<br><br>The debugger runs until it encounters a breakpoint, or until the program ends. | • **Debug Menu**: Select **Go/Continue**<br>• **Keyboard**: Press 'F5'<br>• **Debug Toolbar**: Click the **Go/Continue** button |
| **Step In**<br><br>The debugger executes the next unit of code, and then stops at the subsequent line. | • **Debug Menu**: Select **Step In**<br>• **Keyboard**: Press 'F11'<br>• **Debug Toolbar**: Click the **Step In** button |
| **Step Over**<br><br>Like **Step In**, Step Over executes the next unit of code. However, if the next unit contains a function call, Step Over executes the entire function then stops at the first unit outside of the function. | • **Debug Menu**: Select **Step Over**<br>• **Keyboard**: Press 'F10'<br>• **Debug Toolbar**: Click the **Step Over** button |
| **Step Out**<br><br>The debugger executes the remainder of the current function and then stops at the first unit outside of the function. | • **Debug Menu**: Select **Step Out**<br>• **Keyboard**: Use the associated key binding<br>• **Debug Toolbar**: Click the **Step Out** button |
| **Run to Cursor**<br><br>The debugger runs until it reaches the line where the cursor is currently located. | • **Debug Menu**: Select **Run to Cursor**.<br>• **Keyboard**: Use the associated key binding. |
| | |

| | |
|---|---|
| *Break Now* <br><br> Pause debugging an application at the current execution point. *Go/Continue* continues debugging from that point. | • *Debug Menu*: Select *Break Now* <br> • *Debug Toolbar*: Click the *Break Now* button |
| *Stop* <br><br> Stop the debugging session. *Go/Continue* restarts debugging from the beginning of the program. | • *Debug Menu*: Select *Stop* <br> • *Keyboard*: Use the associated key binding. <br> • *Debug Toolbar*: Click the *Stop* button |
| *Toggle breakpoint* <br><br> Enables, disables, or deletes a breakpoint on the current line. | • *Debug Menu*: Select *Disable/Enable Breakpoint* <br> • *Keyboard*: Press 'F9' |
| *Show Current Statement* <br><br> Moves the editing cursor from any position in the file to the statement at which the debugger is stopped. | • *Debug Menu*: Select *Show Current Statement* <br> • *Keyboard*: Use the associated key binding |
| *Detach* <br><br> Stop the debugging process but continue application process execution. | • *Debug Menu*: Select *Detach* <br> • *Debug Toolbar*: Click the *Detach* button |

## Debugger Stepping Behavior

Instead of running to the end of a program or to the next breakpoint, the debugger can also step through code one statement at a time. The following Debug menu items and toolbar buttons control stepping behavior:

- *Step In*: Executes the current statement and pauses at the following statement.
- *Step Over*: Executes the current statement. If the line of code calls a function or method, the function or method is executed in the background and the debugger pauses at the statement that follows the original one.
- *Step Out*: When the debugger is within a function or method, *Step Out* will execute the code without stepping through the code line by line. The debugger will stop on the line of code following the function or method call in the calling program.

When stepping through a program which calls a function or method from an external program (e.g. a

module or package) the debugger steps into the external program at the point where the function or method is called, opening it in a new tab. Stepping continues in the external program until the function call is completed.

*Note*: Perl operators `sort`, `map`, and `grep` behave like other looping constructs with respect to stepping behavior in the debugger. When Komodo has stopped at one of these operators, ***Step Over*** stops at the first statement or expression used within the first argument of these operators.

For example, if the debugger steps over a statement containing a `foreach`, `while`, `map`, `grep`, or `sort` looping construct that evaluates its body five times, the debugger remains inside that loop for five iterations. When it steps over on the sixth iteration, the debugger exits the loop and stops at the next statement.

To skip execution of such looping constructs, set a breakpoint on the statement following the construct, and continue until Komodo reaches that breakpoint.

# Viewing the Debugging Session

When the Komodo debugger is started, the ***Debug*** tab opens in the Bottom Pane. This tab consolidates views of the debugger output, call stack, program variables (local and global), and watch variables. The ***Debug*** tab also contains a Debug Toolbar for stepping in, out, over, and running functions while debugging.

When debugging more than one session at a time (multi–session debugging), a ***Debug*** tab for each session is accessible in the Bottom Pane. The ***Debug*** tab selected is the session currently being debugged. To change to another debug session, select the ***Debug*** tab for that session (identified by the filename of the program). When a new session is started, a new ***Debug*** tab is created and Komodo automatically switches to that new session.

The ***Debug*** tab is divided into two sub–panes, which have tabs of their own. The right sub–pane contains the ***Output***, ***Call Stack***, and ***HTML Preview*** tabs. The left sub–pane contains variable tabs.

## Viewing Variables

The variables section of the ***Debug*** tab is divided into tabs that vary according to the language of the program being debugged. (Language variations are described below.) Variables with multiple values (such as arrays) are indicated by plus and minus symbols to the left of the variable name.

***To Expand or Collapse Variables***: Plus symbols indicate variables with multiple values that can be expanded; minus symbols indicate variables that can be collapsed. Click on the plus or minus symbol to expand or collapse the variable list.

*To Change Variable Values*: Double–click in the variable's *Value* field and enter the desired value. (The value of nodes in XML documents cannot be changed.)

## Python Variables and Objects

While debugging Python programs, variables and objects are displayed on the *Locals*, *Globals*, and *Code Objects* tabs:

- *Locals*: Displays variables referenced within the current function. If the program is currently outside of a function, all variables are displayed.
- *Globals*: Displays all used program variables.
- *Code Objects*: Displays an expandable tree view of all classes, functions, and their attributes.

During Python debugging sessions, click the *Show Hidden Variables* 👓 button to display special Python variables prefixed with double underscores, such as __doc__, __dict__, etc.

## PHP and Tcl Variables

While debugging PHP and Tcl programs, variables are displayed on the *Locals* and *Globals* tabs:

- *Locals*: Displays variables referenced within the current function. If the program is currently outside of a function, all variables are displayed.
- *Globals*: Displays all used program variables. *Note*: PHP "Super Globals" ($_POST, $_GET, etc.) are hidden by default. The *Show Hidden Variables* button will toggle them on and off.

## Perl Variables

While debugging Perl programs, *Argument* and *Special* tabs are displayed in addition to the *Locals* and *Globals* tabs listed above.

- *Argument*: Displays parameters for the current subroutine (i.e. @_).
- *Special*: Displays current Perl special variables (i.e. @ARGV, %ENV, @INC, $0, etc.)

## XSLT Variables

While debugging XSLT programs, data nodes and variables are displayed on the *Locals* and *Globals* tabs:

- *Locals*: Displays data nodes from the input XML document. Only nodes contained in the context of the template specified in the *Call Stack* are displayed.
- *Globals*: Displays `xsl:param` and `xsl:variable` elements declared at the top level of the program.

## Setting Watched Variables

The *Watch* variable tab monitors selected variables and expressions. Use the *Watch* variable tab to watch variables, or expressions based on variables, by typing expressions, dragging and dropping expressions from an editor, or selecting variables from one of the other variable tabs. Also, use the *Watch* tab to change the value of a variable or remove a variable from the *Watch* tab.

Watched variables can be added, manipulated and removed regardless of whether the debugger is currently running.

To watch one or more variables during program execution:

- Click the *Add* button on the *Watch* variables tab and type a variable name in the dialog box
- Select the variable in the editor (or any other drag−and−drop aware application), then drag and drop the variable into the *Watch* tab
- Right−click a variable in one of the other variable tabs and select *Add to Watch* from the context menu.

The *Watch* variable tab supports viewing the results of expressions made with watched variables. For example, in a Perl program with scalar variables `$base` and `$height` the following expression could be entered:

```
($base / 2) * $height
```

To enter arbitrary expressions on the *Watch* variable tab:

1. Click the *Add* button above the *Watch* variable tab.
2. Enter an arbitrary expression in the dialog box.
3. Click *OK*.

To change the values of variables:

- Double−click the variable on the *Watch* variable tab and specify a value. Currently, only the values of simple variables can be changed. For example, values of variables such as 'a.b[3]' (in Python) or '$a{b}−>[3]' (in Perl) cannot be changed.

  *Note*: This function is not available for arbitrary expressions.

- Double−click the variable on the *Locals* or *Globals* pane and specify a value in the dialog box.

To remove a variable from the *Watch* variable tab, select the variable and click *Delete* on the bottom right toolbar. Alternatively, right–click the desired variable and select *Remove Watch* on the context menu.

## Output Tab

The *Output* tab is used to view program output and to send input to the program being debugged. The following standard data streams are handled in the *Output* tab:

- `stderr`: program output
- `stderr`: errors
- `stdin`: program input (not supported for Perl or PHP during remote debugging)

When debugging Tcl and Python, if `stdin` is requested by the program, a red percent character is shown in the margin of the *Output* tab.

## HTML Preview Tab

If the program produces HTML output, select the *HTML* tab to preview the rendered output. Unlike the *Output* tab, the HTML preview is not constantly updated. Use the *Reload HTML View* button in the bottom–pane toolbar to update the preview.

## Viewing the Call Stack

The call stack is a data area or buffer used for storing requests that need to be handled by the program. Komodo's stack stores temporary data such as variables and parameters and operates as a push–down list. New data moves to the top of the stack and pushes the older data down in a "last–in, first–out" arrangement.

To view the call stack in a current debugging session, select the *Call Stack* tab in the right pane of the *Debug* tab.

There is one line in this tab per stack frame at any point in the execution of a program. The calling frame contains the information about a function call, including the filename, the line number, and any parameters or local variables.

# Watching Files

When debugging a program that writes output to another file, or when watching programs execute, you can watch the output or the log file using Komodo's File Watcher.

The *Watch File* tool shows a file as the file is being updated on disk. It has no relationship with variable viewing, except that, while debugging, it is often useful to watch variables change state and files change content.

To use the File Watcher:

1. On the *Tools* menu, select *Watch File*.
2. Browse to the desired file and click *OK*.
3. Run the program.

# Detaching the Debugger

Use the Detach control to stop the debugging process but continue running the application. When application execution is detached from the debugging process, output continues to print on the *Debug* tab until the application finishes running.

To detach application execution from the debugging process, do one of the following:

- *Debug Menu*: Select *Detach*.
- *Debug Toolbar*: Click the *Detach* button.

# Stopping the Debugger

To stop the Komodo debugger, do one of the following:

- *Debug Menu*: Select *Stop*.
- *Keyboard*: Use the associated key binding.
- *Debug Toolbar*: Click the *Stop* button.

The debug session ends.

# Debugging Perl

Komodo can debug Perl programs locally or remotely, including debugging in CGI environments. The instructions below describe how to configure Komodo and Perl for debugging. For general information about using the Komodo debugger, see Komodo Debugger Functions.

## Configuring the Perl Debugger

To specify which Perl interpreter Komodo uses to debug and run Perl programs:

1. On the *Edit* menu, click *Preferences*.
2. In the Preferences dialog box under *Languages*, click *Perl*. Komodo searches for Perl interpreters on your system and displays them in the drop–down list.
3. If the preferred interpreter is in this list, click to select it. If not, click *Browse* to locate it.
4. Click *OK*.

To start a local Perl debugging session:

On the *Debug* menu or Debug Toolbar, click *Go/Continue* ('F5') or *Step In* ('F11') to invoke the debugging session. See Komodo Debugger Functions for full instructions on using Komodo's debugging functionality.

## Debugging Perl Remotely

When debugging a Perl program remotely, the program is executed on the remote system and the debug output is sent to Komodo. Komodo controls the debugging session (e.g. stepping and breakpoints) once the session starts on the remote system.

Perl remote debugging works on any system that can run the version of *perl5db.pl* distributed with Komodo. ActivePerl and most other distributions of Perl (version 5.6 or greater) will work.

*Note*: If you have the ActiveState Perl Development Kit (PDK) installed, follow the instructions for PDK users to disable the PDK debugger before continuing.

To debug Perl programs remotely:

*Step One: Configure the Remote Machine*

1. Log in to the remote machine.

2. Copy Komodo's perl debugger and it's associated libraries to the remote machine by copying the entire *dbgp/perllib* sub−directory of the Komodo installation to the new machine, or download a package from the [Komodo Remote Debugging](#) page.

   *Note*: Do not copy *perl5db.pl* to the Perl "lib" directory on the remote machine, as this will overwrite the standard *perl5db.pl* file.

3. On the remote machine, set the `PERL5LIB` environment variable to the location of the new *perl5db.pl* and it's libraries. For example, if the remote machine is running Windows and *perllib* directory was copied to *C:\misc\perllib*, set the variable as follows:

   ```
   set PERL5LIB=C:\misc\perllib
   ```

   If the remote machine is running Linux and *perllib* was copied to the */usr/home/me/perl/vperl_debugger* directory, set the variable as follows:

   ```
   export
   PERL5LIB=/usr/home/me/perl/vperl_debugger/perllib
   ```

4. On the remote machine, set the `PERLDB_OPTS` and `DBGP_IDEKEY` variables. This tells the Perl interpreter on the remote machine where to connect to Komodo or the [DBGP Proxy](#) and how to identify itself.

   ```
   PERLDB_OPTS=RemotePort=<hostname>:<port>
   DBGP_IDEKEY=<ide_key>
   ```

   ♦ The port number must match the port number specified in *Edit/Preferences/Debugger*. Click *Debug/Listener Status* to check the current port.
   ♦ Replace `<hostname>` with the name or IP address of the machine running Komodo.
   ♦ If DBGP_IDEKEY is unset, the USER or USERNAME environment variable is used as the IDE Key.
   ♦ The variable definitions must be on one line.

For example:

*Windows 2000, NT, XP*

   ```
   set PERLDB_OPTS="RemotePort=127.0.0.1:9000"
   set DBGP_IDEKEY="jdoe"
   ```

*Windows Me*

Use the MSCONFIG utility (*Start/Run/MSCONFIG*). Select the *Environment* tab, and create a new variable with the Variable Name of `PERLDB_OPTS`, and the Variable Value of

```
       RemotePort=127.0.0.1:9000.
```

### *Unix Systems*

```
           export PERLDB_OPTS="RemotePort=127.0.0.1:9000"
           export DBGP_IDEKEY="jdoe"
```

### *Step Two: Listen for Remote Debugger*

In Komodo, on the *Debug* menu, click *Listen for Remote Debugger*.

### *Step Three: Start the Perl Program on the Remote Machine*

Start the debugging process using the "−d" flag:

```
       perl -d program_name.pl
```

A Perl *Debug* tab is displayed in Komodo.

### *Step Four: Debug the Perl Program using Komodo*

Use 'F11' to *Step In*, or 'F5' (*Go*) to run to the first breakpoint. See Komodo Debugger Functions for full instructions on using Komodo's debugging functionality.


# Disabling and Enabling the Perl Dev Kit (PDK) Debugger

If you have installed the ActiveState Perl Development Kit (PDK) on the remote machine, the system may be configured to use the PDK debugger when a Perl debug session (`perl -d`) is launched. To use Komodo's debugger, disable the PDK debugger on the remote machine first. If necessary, you can re−enable the PDK debugger on the remote machine later.


## Disabling the PDK Debugger on the Remote Machine

To disable the PDK debugger on the remote machine, perform one of the following three procedures:

Option 1:

At the command shell, enter:

```
       set PERL5DB="BEGIN { require 'c:\...\Komodo-3.0\dbgp\perllib-5.6\perl5db.pl'; }"
```

To re−enable the PDK debugger, set the `PERL5DB` variable to an empty string.

Option 2:

1. Right–click the *My Computer* icon and select *Properties*.
2. Click the *Advanced* tab.
3. Click *Environment Variables*.
4. In the *System variables* section, click *New*.
5. Set the *Variable Name* field to PERL5DB.
6. Set the *Variable Value* field to BEGIN { require 'perl5db.pl'}.
7. Click *OK* three times to exit.

*Note*: These changes take effect only in new DOS windows.

To re–enable the PDK debugger, delete the PERL5DB variable.

Option 3:

Change the registry setting for HKEY_LOCAL_MACHINE\SOFTWARE\Perl. Rename the variable PERL5DB to xPERL5DB.

*Warning:* This registry setting is semi–permanent and persists through machine restarts.

To re–enable the PDK debugger, rename the xPERL5DB registry variable back to PERL5DB. This change takes effect only in new DOS windows.


# Configuring Perl for CGI Debugging

Debugging CGI programs on live production servers can seriously impair performance. We recommend using a test server for CGI debugging. Instructions for configuring Microsoft IIS and Apache (Unix) servers are shown below; for other web servers, use these examples and the web server software documentation as a guide for modifying the server environment.

The settings and paths listed are examples only. Substitute these with the specific paths, hostnames and port numbers of your server as necessary


## Configuring a Microsoft IIS Web Server

- *Modify the Server's Environment Variables*: Right–click the *My Computer* icon on the desktop, and select *Properties*. On the *Advanced* tab, click the *Environment Variables* button. Add the following items to the *System Variables* pane:

  ```
  PERL5LIB="C:\Program Files\Komodo-x.x\dbgp\perllib-5.x"
  PERLDB_OPTS=RemotePort=<hostname>:<port>
  DBGP_IDEKEY="<ide_key>"
  ```

- *Modify the Internet Information Services Configuration*: Open the *Internet Information Services* manager. Select the *Home Directories* tab, and click the *Configuration* button. Add (or modify) an entry for Perl with the following characteristics:

```
Extension = .pl
Executable Path = c:\perl\bin\perl.exe –d "%s" %s
```
- *Restart the Server* You must restart the server in order for the above changes to take effect.

## Configuring an Apache Web Server

Ensure that Perl CGI scripts are operating correctly on the Apache server before proceeding with CGI debugger configuration. If you are running Apache under Windows, disable the `ScriptInterpreterSource` registry in the *httpd.conf* file.

Remote debugging works with a stand–alone Perl interpreter or with the *mod_perl* Apache module.

- *Modify the httpd.conf file*: The following values can be configured for a specific virtual host or all hosts. Add the following values in the appropriate sections:

```
SetEnv PERL5LIB "C:\Program Files\Komodo-x.x\dbgp\perllib-5.x"
SetEnv PERLDB_OPTS "RemotePort=<hostname>:<port>"
SetEnv DBGP_IDEKEY "<ide_key>"
```

*Note:* You must enable the `mod_env` Apache module (see httpd.apache.org/docs/mod/mod_env.html) for the SetEnv directive to function.

- *Modify the Perl Script*: Add the "–d" flag to the "shebang" line:

```
#!/perl/bin/perl –d
```

## Starting a CGI Debugging Session

After the configuration is complete, debug programs as follows:

- In Komodo, on the *Debug* menu, click *Listen for Remote Debugger*.
- Using a web browser, access your CGI script.
- A Perl *Debug* tab is displayed in Komodo. See Komodo Debugger Functions for full instructions on using Komodo's debugging functionality.

# Debugging Python

Komodo can be used to debug Python programs locally or remotely, including debugging in CGI environments. The instructions below describe how to configure Komodo and Python for debugging. For general information about using the Komodo debugger, see Komodo Debugger Functions.

## Configuring the Python Debugger

To specify which Python interpreter Komodo should use to debug and run Python programs locally:

1. On the *Edit* menu, click *Preferences*.
2. In the Preferences dialog box under *Languages*, click *Python*. Komodo searches for Python interpreters on your system and displays them in the drop–down list.
3. If the preferred interpreter is in this list, click to select the interpreter. If not, click *Browse* to locate it.
4. Click *OK*.

On the *Debug* menu or Debug Toolbar, click *Go/Continue* ('F5') or *Step In* ('F11') to invoke the debugging session. See Komodo Debugger Functions for full instructions on using Komodo's debugging functionality.

## Using the Python Remote Debugger

When debugging a Python program remotely, the program is executed on the remote machine, and the debug output is sent to Komodo. Komodo controls the debugging session once the session starts on the remote machine.

### Installing the Python Remote Debugger on the Remote Machine

To debug a Python program remotely, the Python debugger modules must be installed on the remote machine. These files can be found in the *python/dbgp* sub–directory of the Komodo installation and are also are also available for download from the Komodo Remote Debugging page.

To install the Python Remote Debugger:

1. Copy all files from the *python/dbgp* sub–directory of the local Komodo installation to a directory on the remote machine. The Python Remote Debugger requires the `logging` module. To verify that this module installed, run the following command on the remote machine:

   ```
   python -c "import logging"
   ```

If this command returns an "ImportError", copy the *python/dbgp/logging* sub–directory to the remote machine as well.

2. On the remote machine, add the directory containing the copied files to the `PYTHONPATH` environment variable. For example, on Windows, if you copied the files to a directory called *C:\debugger*, enter the following at the command line:

```
set PYTHONPATH=%PATH%;C:\debugger
```

# Invoking the Python Remote Debugger

Python remote debugging sessions can be started from the command line or from within the Python program itself. Both methods require the installation of the Python remote debugger modules on the remote machine (see Installing the Python Remote Debugger).

### Running dbgpClient.py from the Command Line

To start a Python remote debugging session from the command line:

1. In Komodo, On the *Debug* menu, click *Listen for Remote Debugger*.
2. Log in to the remote machine.
3. On the remote machine, run the *dbgpClient.py* program:

```
python dbgpClient.py −d <komodo_host:port> script.py [script args]
```

The following options are available:

- ♦ **−d**: Sets the hostname (or IP address) and port where Komodo or DBGP Proxy is running. See *Debug/Listener Status* in Komodo to check the current port setting.
- ♦ **−k**: Sets the `ide_key` used with DBGP Proxy.
- ♦ **−h**: Displays a complete list of options.

4. A Python *Debug* tab opens in Komodo. Use 'F11' to *Step In*, or 'F5' (*Go/Continue*) to run to the first breakpoint. See Komodo Debugger Functions for full instructions on using Komodo's debugging functionality.

The port number set with the "−d" switch must match the port number configured in Komodo under *Edit/Preferences/Debugger*. Click *Debug/Listener Status* to check your current settings.

If you are connecting to a DBGP Proxy, be sure to specify an `ide_key` value with the "−k" switch. For example:

```
python dbgpClient.py −d <komodo_host:port> −k <ide_key> script.py [script args]
```

Set the `ide_key` to the value listed in ***Debug/Listener Status/Proxy Key***. It is taken from the environment variable specified under ***Proxy Key*** in Komodo's Debugger Preferences (USER or USERNAME by default).

## Using dbgpClient Functions in Python Programs

To start a Python remote debugging session from within a Python program:

- Ensure that the Python remote debugging programs are installed on the remote computer as per the instructions in Installing the Python Remote Debugger.
- Import *dbgpClient.py* into your Python program (`import dbgpClient`) or import just the `brk()` function (`from dbgpClient import brk`).
- Set a hard breakpoint with `brk()` one line above where you want the program to break. The `brk()` function takes the following arguments:
  - ◆ ***host***: machine running Komodo or the DBGP Proxy (uses `localhost` if unspecified)
  - ◆ ***port***: port to connect on (uses 9000 if unspecified)
  - ◆ ***idekey***: key used to identify the debugging session to Komodo or the DBGP Proxy (uses the value of the USER or USERNAME environment variable if unspecified)
- The `brk()` function connects to the host and port specified, then breaks on the current line of the script. Under local debugging, calling `brk()` causes the debugger engine to break on the current line, since the engine is already connected to Komodo.

The following script uses this method to initiate debugging:

```
from dbgpClient import brk
def foo():
    print "hello ",
    brk(host='mybox', port=3210)
    print "world!"

foo()
```

This example initiates a debugging connection to Komodo running on a machine called "mybox" on port 3210.

## Just−in−Time Debugging

"Just−in−time debugging" allows the remote debugger to connect to Komodo if an uncaught exception occurs during execution. To prevent these uncaught exceptions from ending your program, add the following lines of code to the beginning of your script:

```
from dbgpClient import brkOnExcept
brkOnExcept(host='mybox', port=3210)
```

The script runs until an exception occurs, at which point a Python interactive shell starts in Komodo. The traceback appears in the *Output* tab, and the variables from the traceback appear in the *Variables* tabs. The *Call Stack* tab displays the call stack location of the exception.

The brkOnExcept() function takes the same arguments as brk(). As with brk(), brkOnExcept() attempts to connect to localhost on port 9000 with an idekey of USER or USERNAME if no arguments are specified.

*Caveats*

- Output from the debug process appears on both the remote machine and in Komodo.
- If dbgpClient.brk is executed when Komodo is not listening, the program runs without any breakpoint.

# CGI Debugging

To debug CGI applications written in Python:

- Configure Python to be used as the CGI (or embedded extension) for your Web server. For information on configuring Python, refer to the Python documentation.
- Follow the steps outlined in Using dbgpClient Functions in Python Programs to call the Python remote debugger from within the application. Start the remote application through a web browser instead of running it from the command line.

# Debugging PHP

Komodo can be used to debug PHP programs locally or remotely. Remote
PHP debugging encompases all types of PHP debugging not initiated from
within Komodo, including debugging PHP scripts running under a local web
server.

The instructions below describe how to configure Komodo and PHP for
debugging. For general information about using the Komodo debugger, see
Komodo Debugger Functions.

Komodo uses a PHP debugger extension called *Xdebug* that must be installed
for Komodo to debug PHP scripts. This can be done manually or with the
PHP Configuration Wizard. Pre−built binaries named "php_xdebug.dll" (for
Windows) or "xdebug.so" (for Linux and Solaris) are provided with Komodo
and are also available for download from the Komodo Remote Debugging
page.

See www.xdebug.org/install.php for instructions on compiling Xdebug from
source on other platforms.

## Installing PHP

PHP debugging in Komodo requires PHP version 4.3.1 or greater (including PHP 5). Download PHP
from http://www.php.net/downloads.php.

To debug PHP scripts in a web environment, be sure PHP is operating correctly with your web server
before configuring the debugger extension. Consult the PHP documentation for details on configuring
PHP with various web servers.

### Windows

If you are unfamiliar with the installation of PHP, we recommend using the Windows InstallShield
package. To install the PHP executable or SAPI module manually, see the PHP website. Be sure that the
PHP directory is included in your system's PATH.

### Linux

Your Linux system may already have PHP installed. Login and type 'php −v' to determine the version of
your current PHP interpreter. If it is earlier than version 4.3.1 you must upgrade. PHP must also support
loading dynamic extensions (the default for PHP under Linux). If it does not, reinstall PHP as per the
instructions on the PHP website.

- RPMs are available for Red Hat from www.redhat.com/apps/support/updates.html
- RPMs for for other distributions are available from http://rpmfind.net/.

When installing PHP to a non−default directory, you must add the following argument to the `./configure` command:

```
--with-config-file-path=/path/to/php.ini
```

...where *path/to/php.ini* is the full path to the directory where the *php.ini* file is located.

Refer to the PHP website for further information on installing PHP.


# Local PHP Debugging

In local debugging mode, Komodo executes PHP directly. While this is convenient for quickly debugging a PHP script, if your script depends on the availability of a web server, use Remote PHP Debugging even if the script is running on the same machine as Komodo. This makes it possible to test the script in its true environment.

When debugging locally, certain environment variables are not available, such as those provided by the CGI environment. However, it is possible to simulate a CGI environment by specifying CGI environment variables and CGI input in the Debugging Options dialog box. It is is not necessary to install a web server to use Komodo's local debugging features. Once you have configured PHP to use the debugger extension as described below, you can debug your scripts by opening a PHP file and using Komodo Debugger Functions.

If you receive an error message when attempting to debug a PHP script, check the PHP troubleshooting section of the Komodo FAQ.

## Configuring Local PHP Debugging

Before debugging PHP scripts in Komodo, PHP must be configured to use the Xdebug extension (php_xdebug.dll or xdebug.so).

Komodo's **PHP Configuration Wizard** simplifies the process of selecting a PHP executable to use for local debugging. It copies the *php.ini* file to a new directory, modifies it for debugging, then copies the Xdebug extension to the directory defined in the .ini file. The PHP Configuration Wizard should only be used to configure local debugging. To debug PHP remotely, see Configuring Remote PHP Debugging.

To start the wizard:

1. On the *Edit* menu, click *Preferences*.
2. Under *Languages*, select *PHP*.
3. Click the *Debugger Configuration Wizard* button to display the wizard's introductory window.
4. Click *Next* to begin configuring PHP for debugging.

The wizard will guide you through the steps necessary to configure the debugging extension.

1. ***Choose Installation***:

   The first step assumes that you have already installed PHP. If you have more than one version of PHP installed, choose the version to configure. Browse to the directory containing the PHP executable or enter the directory path in the ***Set up this installation*** field, and then click ***Next***.
2. ***Choose PHP INI Path***:

   Next, choose the *php.ini* file to be copied and its destination directory. Subsequent changes to the original file will not be available in the new copy. Modify the new file directly, or rerun the wizard to copy the changes. Click ***Next***.

   - On Windows, the *php.ini* file is generally located in `c:\windows`, or `c:\winnt`, depending on your operating system. It may also be located in the same directory as your php.exe executable.
   - On Linux, the default location is `/usr/local/lib`. It may also be located in the same directory as your PHP executable.

3. ***Choose PHP Extension Directory***:

   Many PHP installations already include a default "extensions" directory. This is where the debugger extension should be installed. If you have specified an extensions directory in the .ini file, you do not need to change the path that appears in the ***Use this extensions directory*** field. It is important that the extensions are installed in the same directory as your PHP installation. If you choose a different location, some extensions may not work. Once the desired path is set, click ***Next***.
4. ***Ready to Install***:

   The final window in the wizard displays the installation options. Confirm that the selections are correct and click ***Next***. To change any of the selections, click ***Back***.

## Starting and Stopping a PHP Local Debugging Session

To step through the script, from ***Debug*** menu, select ***Step In*** or press 'F11'.

To run the script to the first breakpoint, from the ***Debug*** menu, select ***Go/Continue*** or press 'F5'.

To stop the debugger, from the **Debug** menu, select **Stop**, or press 'Shift+F5'.

See Komodo Debugger Functions for full instructions on using Komodo's debugging functionality.

# Remote PHP Debugging

Remote PHP debugging encompases all types of PHP debugging not initiated from within Komodo, including debugging PHP scripts running under a local web server.

When a PHP script is run through a web browser, the web server uses the PHP interpreter to execute the script. If PHP is configured for remote debugging, the server contacts Komodo to start a debugging session. Komodo controls the debugging (e.g. stepping and breakpoints) once the session starts. CGI variables are available, as are all other variables that are available when running PHP under a web server.

Though remote PHP debugging allows PHP scripts to be run in their true environment, it may be slower than local PHP debugging.

## Configuring Remote PHP Debugging

Remote debugging of PHP in Komodo is set up differently depending on how many people will be debugging scripts on the same web server:

*Single User Remote PHP Debugging*: In single user remote debugging, PHP is configured to always look for a specific instance of Komodo on a specific machine. This configuration requires no changes to the PHP script. Your web server and your instance of Komodo can be on one machine or two machines

*Multi−User Remote PHP Debugging*: When multiple users need to debug PHP scripts on a single web server, use the DBGP Proxy with the remote PHP debugging instructions below. While it is possible to configure Apache with Virtual Hosting, it is easier to configure multi−user remote PHP debugging with the proxy.

Remote PHP debugging must be configured manually. The following procedure assumes that you have already installed PHP.

### Step 1 − Copy the Debugging Extension to the Web Server

Before debugging PHP scripts in Komodo, PHP must be configured to use the Xdebug extension (php_xdebug.dll or xdebug.so).

Manually copy `xdebug.so` (Unix) or `php_xdebug.dll` (Windows) into a directory on the server that the PHP interpreter and web server can access. We recommend installing Xdebug in the existing PHP extensions directory on the web server (specified by the `extension_dir` variable in the *php.ini* file). These Xdebug files can be found in the *php/debugging/<PHP version>* sub−directory of the Komodo installation or downloaded from the Komodo Remote Debugging page.

If you are installing PHP for the first time, the `extension_dir` may be set to "./". You should change this to a full, direct path, such as `C:\php\extensions` under Windows, or `/usr/local/lib/php/extensions` under Linux.

*Windows*

- *File required*: php_xdebug.dll
- *Source location*:
  `<komodo-install-directory>\php\debugging\<PHP-version>` or the Komodo
  Remote Debugging page.
- *Destination*: the extension_dir directory as defined in the *php.ini* file.

*Linux*

- *File required*: xdebug.so
- *Source location*:
  `<komodo-install-directory>/php/debugging/<PHP-version>/`
- *Destination*: the extension_dir directory as defined in the *php.ini* file.

## Step 2 – Edit the Web Server's PHP Configuration

*Windows*

Open the *php.ini* configuration file on the web server. In the "Dynamic Extension" section, add the
following lines:

```
; xdebug config for Windows
zend_extension_ts=c:\path\to\php_xdebug.dll
xdebug.remote_enable=1
xdebug.remote_handler=dbgp
xdebug.remote_mode=req
xdebug.remote_port=9000
xdebug.idekey=<idekey>
```

*Note*: The *php.ini* configuration file should be in your operating system directory (e.g. C:\WINDOWS or
C:\WINNT), or in the same directory as php.exe (e.g. C:\PHP). If you used the PHP Windows
installer, this file should be in the correct location.

*Linux*

Open the *php.ini* configuration file on the web server. In the "Dynamic Extension" section, add the
following lines:

```
; xdebug config for Linux
zend_extension=/path/to/xdebug.so
xdebug.remote_enable=1
xdebug.remote_handler=dbgp
xdebug.remote_mode=req
xdebug.remote_port=9000
xdebug.idekey=<idekey>
```

Set the "remote_port" to the same value as the debugging listener port configured in
*Edit|Preferences|Debugger*.

Once the *php.ini* file is updated, verify that Xdebug is configured by running the following command:

```
php -m
```

Lists of all PHP and Zend modules are displayed. Make sure Xdebug appears in both lists.

*Note*: Recent versions of PHP are set to buffer program output by default. While debugging, it is useful to disable output buffering so that results of `print` and `echo` statements can be seen immediately when stepping through code. To disable output buffering, comment out the `output_buffering` setting in *php.ini* with a ";" character, or set it to "`off`".

## Starting and Stopping a PHP Remote Debugging Session

Once remote PHP debugging is configured, the PHP interpreter can contact Komodo and initiate a remote debugging session when a PHP script is executed on the web server.

To debug a PHP script using a web browser:

1. Ensure you have configured PHP for your web server, have it properly working, and have configured PHP and Komodo as described in Configuring Remote PHP Debugging.
2. On the *Debug* menu, click *Listen for Remote Debugger*.
3. Open your web browser and access the script you want to debug. Append "?XDEBUG_SESSION_START=<idekey>" to the end of the URL as a GET argument. The IDE Key should match the *Proxy Key* value shown in *Debug|Listener Status*. For example:

   ```
   http://example.org/sample.php?XDEBUG_SESSION_START=jdoe
   ```

   *Note*: This is only required for the first request. After that, Xdebug tracks the debugging session with a cookie. For more information on how this works, see www.xdebug.org/docs−debugger.php#browser_session
4. A PHP debugging session starts in Komodo. Use '*F11*' to *Step In*, or '*F5*' (*Go*) to run to the first breakpoint.

To debug a PHP script remotely from the command line:

1. From the *Debug* menu, click *Listen for Remote Debugger*.
2. Set the XDEBUG_CONFIG environment variable. Use the port specified in *Edit|Preferences|Debugger* or listed in *Debug|Listener Status*.

   On Windows:

   ```
   set XDEBUG_CONFIG=remote_port=9000 remote_enable=1
   ```

   On Linux (using the bash shell):

   ```
   export XDEBUG_CONFIG="remote_port=9000 remote_enable=1"
   ```

3. Run the script using the PHP interpreter:

```
    php -f sample.php
```
4. A PHP debugging session will start in Komodo. Click *Step In* ('F11') to start stepping through the script or *Go* ('F5') to run to the first breakpoint.

To debug a PHP script remotely using the [DBGP Proxy](#):

1. From the *Debug* menu, select *Listen for Remote Debugger*.
2. Set the XDEBUG_CONFIG environment variable as above. Use the port specified in *Edit/Preferences/Debugger* or listed in *Debug/Listener Status*. Add an IDE Key value to the XDEBUG_CONFIG environment variable that matches the *Proxy Key* value shown in *Debug/Listener Status*.

   On Windows:

   ```
   set XDEBUG_CONFIG=remote_port=9000 remote_enable=1 idekey=<USERNAME>
   ```

   On Linux (using the bash shell):

   ```
   export XDEBUG_CONFIG="remote_port=9000 remote_enable=1 idekey=<USER>"
   ```
3. Run the script using the PHP interpreter:

   ```
   php -f sample.php
   ```
4. A PHP debugging session will start in Komodo. Click *Step In* ('F11') to start stepping through the script or *Go* ('F5') to run to the first breakpoint.

Output from the debug session appears in the Bottom Pane of the Komodo Workspace. Komodo does not support a console for remote debugging. The browser will not show the script output until debugging is complete.

To stop the debugger:

- From the **Debug** menu, select **Stop**
  or
- Press 'Shift+F5'

See [Komodo Debugger Functions](#) for full instructions on using Komodo's debugging functionality.

If you receive an error message while debugging a PHP script that is not caused by the errors in the script itself, check the [PHP troubleshooting](#) section of the Komodo FAQ.


## Using xdebug_break()

The `xdebug_break()` function is used to hard-code a break in a PHP program. It can be used instead of a Komodo breakpoint. For example:

```php
<?php
echo "<p>Breaks after this line.</p>";
xdebug_break();
echo "<p>Breaks before this line.<p>";
?>
```

This function breaks the code during a debugging session but will not initiate a new session. Use xdebug_break() in conjunction with the methods described above for starting debugging sessions.

# Debugging Tcl

Komodo can be used to debug Tcl programs locally or remotely. The following instructions describe how to configure Tcl debugging. For general information about using the Komodo debugger, see Komodo Debugger Functions.

*Tutorial*

- Tcl Tutorial

## Configuring Local Tcl Debugging

Specify the Tcl interpreter Komodo uses to debug and run Tcl programs:

1. On the *Edit* menu, select *Preferences*.
2. In the Preferences dialog box under *Languages*, click *Tcl*. Komodo searches for Tcl interpreters in the system PATH and lists all `tclsh` and `wish` interpreters available in separate drop–down lists. If no Tcl interpreters are displayed in the list, check that the location of the interpreters is specified in your PATH environment variable.
3. If the preferred interpreters are in these lists, click to select them. If they are not, click *Browse* to locate them.
4. Click *OK*.

To start a local Tcl debugging session, click *Go/Continue* (F5) or *Step In* (F11) in the Debugger menu or toolbar. See Komodo Debugger Functions for full instructions on using Komodo's debugging functionality.

# Remote Tcl Debugging

When debugging a Tcl program remotely, the program is executed on the remote machine and the debug output is sent to Komodo. Komodo controls the debugging session (e.g. stepping, breakpoints, and spawnpoints) once the session has been started on the remote machine.

### Installing the Tcl Debugger Application on a Remote Machine

To debug a Tcl program remotely, the Tcl debugger application, *dbgp_tcldebug.exe* (Windows) or *dbgp_tcldebug* (Linux), must be installed on the remote machine. This file is installed with Komodo in the *tcl* sub–directory of the Komodo installation directory and is also available for download from the Komodo Remote Debugging page.

To install the Tcl debugger application on the remote machine:

- If necessary, install a Komodo license.
- Copy the *dbgp_tcldebug* executable to any convenient directory.

# Invoking the Tcl Debugger Application

To debug a Tcl script on a remote machine:

1. In Komodo, select *Listen for Remote Debugger...* from the *Debug* menu.
2. Log in to the remote machine.
3. On the remote machine, run the *dbgp_tcldebug* executable from the command line.

   ```
   dbgp_tcldebug -dbgp <komodo_host:port>
           -app-file <tcl_program>
           -app-shell </path/to/tclsh_or_wish>
   ```

   The following options are available:

   - ♦ *-dbgp*: Sets the hostname (or IP address) and port where Komodo or the DBGP Proxy is running. In Komodo, select *Debug/Listener Status* to check the current port setting.
   - ♦ *-app-file*: Specifies the Tcl program to debug. Program arguments should follow a "--" delimiter after the Tcl program name (e.g. `... -app-file test.tcl -- arg_0 arg_1`).
   - ♦ *-app-shell*: Sets the path to the Tcl interpreter (`tclsh` or `wish`).
   - ♦ *-help*: Displays a complete list of options.
4. A Tcl *Debug* tab opens in Komodo. Use *F11* to *Step In*, or *F5* (*Go/Continue*) to run to the first breakpoint (see Komodo Debugger Functions for full instructions).

*Example*

Remote Machine (Windows):

- The file *dbgp_tcldebug.exe* has been copied into the `C:\remote_debug` directory.
- The Tcl file to be debugged is called `test.tcl` and is located in the current working directory.
- The Tcl interpreter is `C:\Tcl\bin\wish.exe`.

Local Machine:

- The hostname is "mybox".
- The Komodo remote debugging listener port is set to 9000.

In this scenario, the following command is entered on the remote machine:

```
C:\remote_debug\dbgp_tcldebug.exe -dbgp mybox:9000
        -app-file test.tcl -app-shell C:\Tcl\bin\wish.exe
```

# Debugging XSLT

Komodo does not need to be manually configured for local XSLT debugging. It uses the *libxslt* and *libxml* libraries directly to transform XML documents into HTML, text, or other XML document types. See www.libxml.org for more information on this XML C parser and toolkit.

*Tutorial*

- XSLT Tutorial

For general information about debugging with Komodo, see General Debugger Functions.

## Using the XSLT Debugger

To debug an XSLT file:

1. Open the XSLT file and set breakpoints.
2. Start the debugger by clicking *Go/Continue* (F5) or *Step In* (F11) in the *Debug Toolbar*.
3. In the *Debugging Options* dialog, *Select the input XML file*
4. Click *OK* to start the debugger.

The XSLT program, the input XML file, and the results of the transformation appear simultaneously. By default, Komodo splits the Editor pane horizontally.

- The XSLT program continues to appear in the top tab group.
- The XML input file appears in a new tab below the XSLT program.
- The results of the transformation are displayed in the Output tab.

A yellow arrow on the breakpoint margin shows the current line of execution in both the XSLT and XML file. Breakpoints can be set in the both files before starting the debugging session, or while stepping through the code.

### Using a Remote XML Input File

To debug using an XML file on a remote server, enter the full URL to the file in the *Select the input XML file* field (for example, http://www.example.org/input_file.xml).

### XSLT Stepping Behavior

Stepping behavior in the XSLT file is similar to the standard stepping behavior described in Debugger Stepping Behavior, but the terminology for describing XSLT is slightly different than that used for scripting languages.

- *Step In*: Executes the current XSL element or template line and pauses at the following line.
- *Step Over*: Not applicable. Behaves the same as *Step In*.

- *Step Out*: When the debugger is within an XSL element, *Step Out* will execute the entire block without stepping through the code line by line. The debugger will stop on the line following the closing tag of the element.

Though the current line is highlighted in both the XSLT and XML files, the stepping behavior is only applicable to the XSLT file.

# Interactive Shell

Komodo's interactive shell implements individual language shells within Komodo. Shells are used to directly communicate with the specified language interpreter. Statements, expressions, and code fragments can be entered independent of program files. The shell can be used as a stand–alone interactive tool or as a shell that interacts from within a debugging session.

*Tutorial*

- Python Tutorial

*Feature Showcase*

- interactive shell

## Stand–Alone Interactive Shell

When the interactive shell is started as a stand–alone tool, use the shell to help test modules and experiment with new languages or programs. Other uses for a stand–alone interactive shell include:

- prototyping code
- identifying bugs
- experimenting with a library
- programming interactively
- learning new syntax

The interactive shell supports history recall, AutoComplete and CallTips (Tcl only), and custom colors and fonts.

## Debugging with an Interactive Shell

When the interactive shell is started from within a debug session, use the shell to access all functions and code being debugged. When the shell is closed from within a debug session, continue the debug process where you left off. Depending on the language used, changes made in the shell remain in effect for the duration of the debug session. Other uses for an interactive shell within a debug session include:

- exploring and debugging a program
- adding new code to the program being debugged (language–dependent)
- modifying existing variables using complex expressions
- adding new variables with code

The interactive shell supports history recall, AutoComplete and CallTips (Tcl only), and custom colors and fonts.

# Using the Interactive Shell

Each Komodo interactive shell is associated with a corresponding interpreter and is thus language–specific. Each time a command or multi–line string is entered into the Shell tab, that code is sent to the corresponding interpreter for evaluation. The interpreter evaluates the command, and then returns output and error text.

## Setting Shell Preferences

Use the Preferences dialog box to specify the default language to use within an interactive shell. Other shells can still be accessed via *Tools*/*Interactive Shell*.

To set the default shell preference:

1. On the *Tools* menu, select *Interactive Shell*/*Configure*
2. On the *Preferred Interactive Shell* drop–down list, select the desired language (Python, Perl, Tcl).
3. Click *OK*.

## Starting the Interactive Shell

The interactive shell can be opened as a stand–alone tool or as a shell inside of a debugging session.

To start the shell as a stand–alone tool:

- Select *Tools*/*Interactive Shell*, and then select the desired language (Python Shell, Tcl Shell, Perl Shell). Alternatively, click the *Shell* button on the Workspace toolbar.

The interactive shell opens in a Shell tab in the Bottom Pane beside the Command Output and Breakpoint tabs.

To start the shell from within a debug session:

- On the active *Debug* tab, click the >> button on the Debug toolbar, or select *Debug*/*Interact*. The *Debug* tab toggles to a Shell tab. Enter code as desired. To toggle back to the *Debug* tab, click the >> button on the Debug Toolbar.

View debugging and code inspection functions by clicking the "Collapse/Expand Pane" button at the left side of the Bottom Pane. This splits the shell into a left and right pane. The left pane performs debugging functions while the right pane contains the interactive shell.

## Using Multiple Shells

Open multiple interactive shells to interact with various code snippets from a single language or use many shells to simultaneously explore a different language in each shell.

## Using AutoComplete and CallTips

The Tcl interactive shell displays AutoComplete and CallTips when recognized code and commands are entered into the shell. Use AutoComplete and CallTips to limit the amount of typing in each session. To select a suggested item, press *Enter*. Use the up and down arrow keys to scroll through the various options on the screen. To cancel or ignore the suggested AutoComplete or CallTip, press *Esc*.

Komodo can also detect when further data is required at the command prompt. When insufficient programming data is entered at the prompt, Komodo displays a language–dependent "more" prompt. This prompt indicates that the language interpreter requires more information before the code can run. Once enough data is entered, Komodo executes the code and the standard language–dependent input prompt returns.

## Customizing Colors and Fonts

The Shell tab displays commands, variables, error messages, and all language syntax in the same scheme as specified in *Edit|Preferences|Fonts and Colors*. See Customizing Fonts and Colors for more information.

## Viewing Shell History

The code history consists of the ordered, numbered sets of commands entered in the lifetime of the shell, including interleaved output and error messages. Use the up and down arrow keys to cycle through the history of all entered commands. When viewing a multi–line command or function, use the 'Enter' key to select the desired function and then use the arrow keys to cycle through the multiple lines within that function.

## Stopping a Shell Session

To stop an interactive shell session and close the Shell tab, click the *X* button located in the upper–right corner of the Shell tab. To stop the interactive shell and keep the Shell tab open, click the square button, or use the associated key binding.

## Clearing the Shell Buffer

To clear the shell buffer, click the *Clear Buffer* button. There is no limit to buffer size; unless it is manually cleared, the buffer will continue to increment until the interactive shell session is closed. Manually clearing the buffer only removes the command history and command results, and has no effect on the buffer state (such as changes to the working directory, etc).

# Using the Python Interactive Shell

The Python shell prompt is a group of three angle brackets >>>. A **...** prompt is displayed if Komodo determines that more information is required before the code can execute. A ⚡ prompt is displayed when input from `stdin` is required (for example, in a Python shell, enter `help()`). No prompt is displayed when program output is sent to the screen. Code errors are displayed in italics. When a Python interactive shell session begins, a welcome message is printed stating a version number and copyright notice. The first prompt is printed as follows:

```
Python 2.2.2 (#37, Nov 25 2002, 13:15:27) [MSC 32 bit (Intel)] on win32
Type "copyright", "credits" or "license" for more information.
>>>
```

The following example shows a series of Python statements with resulting output:

```
>>> # Comment: my hello world test
>>> print "Hello World"
Hello World
>>> x=12**2
>>> x/2
72
>>>
```

## Debugging with the Python Shell

To start a Python shell from within a debug session, click the » Interact button, located in the upper-right corner of the *Debug* tab. Starting a shell within a debug session enables Interact Mode. In Interact Mode, view debugging and code inspection functions by clicking the "Collapse/Expand Pane" button at the left side of the Bottom Pane. This splits the shell into a left and right pane. The left pane performs debugging functions while the right pane contains the interactive shell. In Interact Mode, debugging functionality (for example, Run, Step In, Step Out) is not available. To return to the debugger, click the Interact button again to exit Interact Mode.

# Using the Tcl Interactive Shell

The Tcl interactive shell supports the tclsh interpreter. The Tcl shell prompt is a percent character *%*. A >
prompt is displayed if Komodo determines that more information is required before the code executes. A
 prompt is displayed when input from `stdin` is required. No prompt is displayed when program
output is sent to the screen. Code errors are displayed in italics. The following examples show how input,
output, and errors are displayed in the Tcl shell:

```
%puts "Hello World"
Hello World
%

%put "hello world"
invalid command name "put"
%puts "hello world"
hello world
%
```

## Debugging with the Tcl Shell

To start a Tcl shell from within a debug session, click the  Interact button, located in the upper−right

corner of the *Debug* tab. Starting a shell within a debug session enables Interact Mode. In Interact Mode,
view debugging and code inspection functions by clicking the "Collapse/Expand Pane" button at the left
side of the Bottom Pane. This splits the shell into a left and right pane. The left pane performs debugging
functions while the right pane contains the interactive shell. In Interact Mode, debugging functionality
(for example, Run, Step In, Step Out) is not available. To return to the debugger, click the Interact button
again to exit Interact Mode.

# Using the Perl Interactive Shell

The Perl interactive shell prompt is a percent character *%*. A > prompt is displayed if Komodo
determines that more information is required before the code executes. No prompt is displayed when
program output is sent to the screen. Code errors are displayed in italics. The following examples show
how input, output, and errors are displayed in the Perl shell:

```
%print "Hello World! \n";
Hello World!
%
```

```
%prin "Hello World! \n";
syntax error
%print "Hello World!!! \n";
Hello World!!!
%
```

*Using Strings, Function Definitions, and Multiple Line Input*

Use the Perl shell to enter function definitions, long strings, and specify `if` and `while` blocks interactively. The Perl shell also handles multiple line input delimited by braces, curly braces, single quotes, and double quotes. The following examples demonstrate this usage.

*Example*: Using single quotes "''" to enter multiple line input.

```
% $b = 'abc
> def
> ghi
> jkl'
abc
def
ghi
jkl
%
```

*Example*: Using curly braces "{}" to define a function and enter multiple line input.

```
% sub foo {
> my $arg = shift;
> my $arg2 = shift;
> return $arg + $arg2;
> }
% foo(10, 12)
22
%
```

*Example*: Using braces to enter a multiple line string.

```
% $name = 'Bob'
Bob
% print qq(<html><head><title>
> Browser Window Caption Text
> </title></head><body bg="white">
> <p>Welcome to my fine web site, $name
> </body>
> </html>)
<html><head><title>
Browser Window Caption Text
```

```
</title></head><body bg="white">
<p>Welcome to my fine web site, Bob
</body>
</html>
%
```

*Example*: Using a backslash to continue a statement.

```
% print 'abc ', 'def ', \
> 'ghi'
abc def ghi
%
```

*Example*: Using a backslash to continue a statement.

```
% $first_long_variable_name = 3
3
% $second_long_variable_name = 4
4
% $third_long_variable_name_to_store_result =
$first_long_variable_name + \
> $second_long_variable_name
7
```

*Example*: Using a braced construct

```
% foreach $var (sort keys %ENV) {
> print "$var = $ENV{$var}\n";
> }
ALLUSERSPROFILE = C:\Documents and Settings\All Users
COMMONPROGRAMFILES = C:\Program Files\Common Files
COMSPEC = C:\winnt\system32\cmd.exe
KOMODO_VERSION = 3.0.0-beta2
LESS = --quit-at-eof --quit-if-one-screen --ignore-case
--status-column --hilite-unread --no-init
MSDEVDIR = C:\PROGRA~1\MICROS~3\Common\msdev98
MSVCDIR = C:\PROGRA~1\MICROS~3\VC98
NETSAMPLEPATH = C:\PROGRA~1\MICROS~1.NET\FRAMEW~1\Samples
OS = Windows_NT
OS2LIBPATH = C:\winnt\system32\os2\dll;
PATH = C:\PROGRA~1\MICROS~3\Common\msdev98\BIN;
PROCESSOR_ARCHITECTURE = x86
PROCESSOR_IDENTIFIER = x86 Family 6 Model 7 Stepping 3, GenuineIntel
PROCESSOR_LEVEL = 6
PROCESSOR_REVISION = 0703
PROGRAMFILES = C:\Program Files
PROMPT = $P$G
```

Interactive Shell

```
SYSTEMDRIVE = C:
SYSTEMROOT = C:\winnt
TEMP = C:\DOCUME~1\toto\LOCALS~1\Temp
TMP = C:\DOCUME~1\toto\LOCALS~1\Temp
USERNAME = toto
USERPROFILE = C:\Documents and Settings\toto
WINDIR = C:\winnt
```

## Debugging with the Perl Shell

To start a Perl shell from within a debug session, click the ⟫ Interact button, located in the upper–right corner of the *Debug* tab. Starting a shell within a debug session enables Interact Mode. In Interact Mode, view debugging and code inspection functions by clicking the "Collapse/Expand Pane" button at the left side of the Bottom Pane. This splits the shell into a left and right pane. The left pane performs debugging functions while the right pane contains the interactive shell. In Interact Mode, debugging functionality (for example, Run, Step In, Step Out) is not available. To return to the debugger, click the Interact button again to exit Interact Mode.

# Code Intelligence

Komodo's Code Intelligence system is a set of tools that make browsing, searching, and programming complex code easier and more accessible. Use Code Intelligence functionality to view the hierarchical code structure within a program file or project, search for language–specific code constructs (for example, variables, methods, imports), and easily preview and open associated source files in the editor. Code Intelligence supports programming in Python, Perl, Tcl, PHP, and JavaScript. Code Intelligence is comprised of the following tools:

- *Code Browser*: A tab that displays a hierarchical view of all code constructs (for example, variables, methods, imports) in all open files. In the Code Browser, symbols can be sorted and filtered; the current scope of a symbol can be located. The lower part of the Code Browser provides additional documentation (when available) on various program components. To access the Code Browser, click **View|Tab|Code Browser**. The Code Browser is displayed on the **Code** tab beside the **Projects** tab.

- *Object Browser*: A graphical browser that searches the Code Intelligence database for specified code symbols and modules. Use the Preview Pane to view code snippets containing the search criteria. To open the Object Browser, select **Tools|Object Browser**.

- *Python AutoComplete and CallTips*: The Code Intelligence system provides AutoComplete and CallTips when programming in Python. Python AutoComplete and CallTips require a Code Intelligence database. See Building the Code Intelligence Database for more information. Note that other languages do not require the Code Intelligence database for AutoComplete and CallTips.

All Code Intelligence tools require a database to operate fully. See Building the Code Intelligence Database for more information. The Python Tutorial demonstrates the Code Browser and other Code Intelligence tools. See the Python Tutorial to explore a Python program in Komodo.

## Building the Code Intelligence Database

Before using Komodo Code Intelligence tools, you must populate the Code Intelligence database. The Code Intelligence database contains the information regarding code constructs in source files and language installations. Information in the database is used by the Code Browser, Object Browser, and Python AutoComplete and CallTips. Komodo automatically updates the database as files are opened in Komodo. However, in order to use the Object Browser to view constructs in files that have not been

*Feature Showcases*

- view help using the code browser
- find components using the object browser
- view scope

opened, use the following wizards: *Scan language installations* and *Scan custom directories*. To build the database to include all desired language installations, run the Scan language installations wizard (located under *Edit/Preferences/Code Intelligence*).
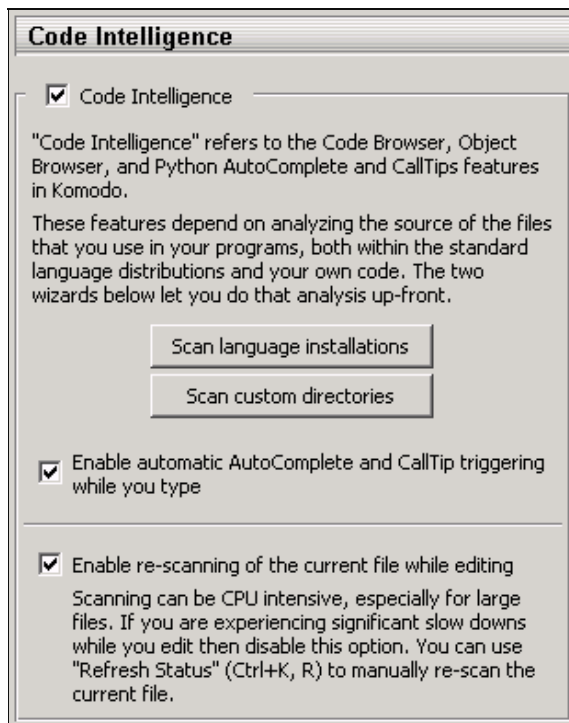


If you use custom directories to store code components (functions, modules, etc.) outside of a language installation, run the Scan custom directories wizard (located under *Edit/Preferences/Code Intelligence*) to configure the Code Intelligence database.

See Code Intelligence Preferences for more information on building a database.

# Code Browser

Use the *Code Browser* to view the general program structure of all source files open in the editor. For each source file, the Code Browser displays a tree of symbol nodes, including: modules, classes, functions, interfaces, namespaces, imports and variables. In Python, instance attributes are also displayed. Each node in the tree hierarchy can be expanded to display further detail, acting as an index to your source code. Symbols can be sorted, filtered, and the current scope of a symbol can be located. The lower part of the Code Browser displays code descriptions (when available) on various program components. The Code Browser supports the following languages: Python, Perl, PHP, Tcl and JavaScript.
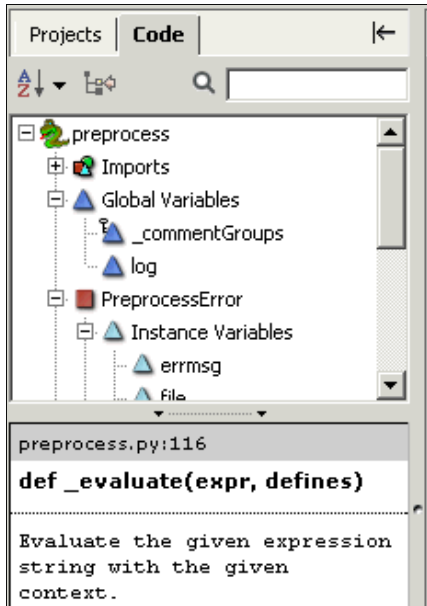
Use the Code Browser to:

- View program structure.
- Browse from a listed namespace, command, or variable definition and jump to the actual source code where it is declared.
- Locate all variables used within a file.
- View a symbol definition signature.
- Find all defined symbols matching a pattern.

## Context Menu

Right−click in the *Code Browser* Pane to access code searching options. The following options are available:
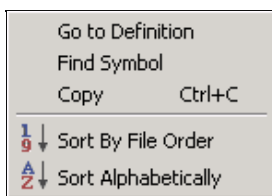


- *Go to Definition*: Jumps to the definition of the associated symbol in the editor. Alternatively, double−click the symbol name in the Code Browser tree.
- *Find Symbol*: Right−click on a symbol name to search for matching symbols in the Object Browser.
- *Copy*: Copies the symbol name to the clipboard.
- *Sort By File Order*: Sorts all symbols in the tree by file order.
- *Sort Alphabetically*: Sorts all symbols in the tree alphabetically.

## Sorting

Use the *Sort By* ![Sort By icon] button to organize all Code Browser symbols by file order or alphabetically. To sort all symbols by file order, click *Sort By*, and then select *Sort By File Order* from the drop−down list. To sort all symbols alphabetically, click *Sort By*, and then select *Sort Alphabetically* from the drop−down list. Alternatively, use the context menu (right−click in the Code Brower) to access sorting options.

## Locating Current Scope

Use the *Locate Current Scope* ![Locate Current Scope icon] button to find the scope of a symbol (for example, namespace, command, or variable definition). To view the scope of a symbol, place the cursor on the desired symbol in the source code and then click the *Locate Current Scope* button. The Code Browser tree opens to the associated scope. Alternatively, click *Code/Locate Current Scope in Code Browser* to open the Code Browser tree to the associated scope.

## Using the Scope Indicator

The Komodo status bar displays a Scope Indicator when a file written in a supported language is open in the Editor Pane. Place the cursor over the Scope Indicator to display the current scope name and type (class, function, etc). Double−click the Scope Indicator to open the Code Browser and locate the current scope within the tree hierarchy.
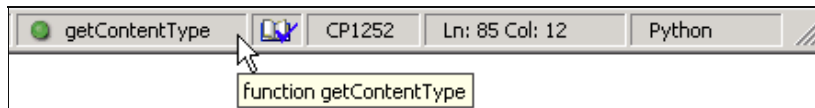


## Filtering Symbols

The *Filter Symbols* ![Filter Symbols text box] text box limits the Code Browser display to matching symbols. To filter symbols in the Code Browser, enter the desired symbol name, or partial name, in the text box. Press 'Tab' to switch focus between the Filter text box and the Code Browser tree. Press 'Esc' to clear the current filter pattern.

## Viewing Code Descriptions

Use the *Code Description* pane to view additional information on a method, class, variable, etc. Code documentation is only displayed when the documentation itself is included within the source file. To open the Code Description pane, click the *Show/Hide Description* ![Show/Hide Description button] button, located at the bottom of the Code Browser. To view code descriptions, select the desired symbol in the Code Browser tree hierarchy.
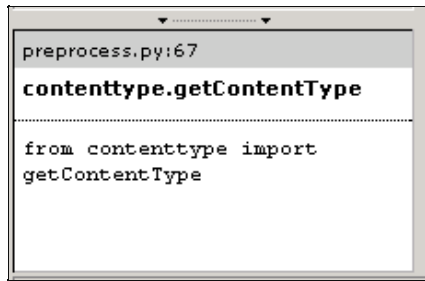
The Code Description pane displays the following information, when available:

- Filename where the symbol exists.
- Line number where the symbol is declared.
- Name of the symbol.
- Internal documentation, if available (for example, Python docstrings).
- Symbol declaration signature.

Code Description limitations:

There are limitations in how the Code Intelligence system matches internal documentation to declared symbols. These limitations only affect whether associated documentation is shown in the Code Browser's Description pane. Specifically, nearby comments are typically not associated with declarations (function, variable, class, etc). Further, POD documentation in Perl files is not mapped to associated modules and subs. However, the following properly display in the Code Description pane: PHP block comments immediately preceding classes and functions, and Python modules, classes, and def docstrings.

# Object Browser

The *Object Browser* is a graphical browsing tool that searches the Code Intelligence database for specified code symbols and modules. To open the Object Browser, select *Tools|Object Browser*. Alternatively, invoke the Object Browser from the Code Browser context menu.
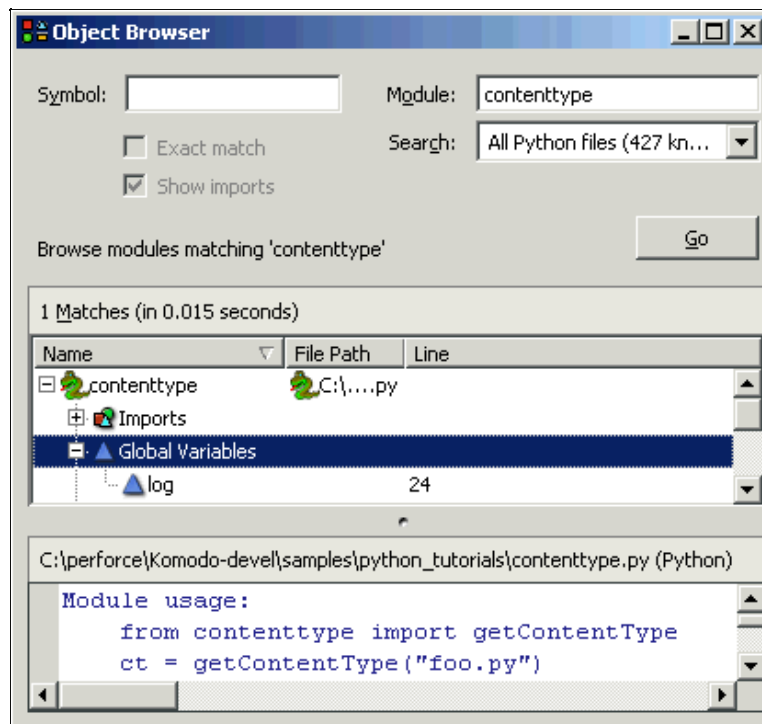
When searching source files by symbol, module, or a combination of both, the **Matches** pane displays a tree of symbol nodes that outline the general program structure of found search criteria. Each node in the tree hierarchy can be expanded to display further detail. Select a node to view the symbol code in the **Preview** pane. Double−clicking a symbol opens the file in the editor at the position where that symbol is declared.

## Searching

Search by symbol, module or a combination of both to locate all instances where a component is used.
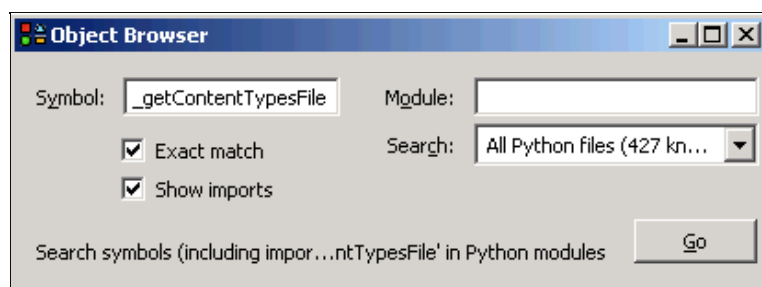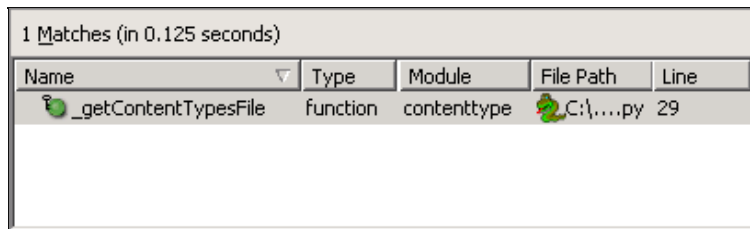


To search:

1. In the **Symbol** text box, enter the desired name to search. Leave blank to just search for modules.
2. In the **Module** text box, enter the desired name to search. Leave blank to search all modules.
3. Select **Exact match** to find the literal symbol and/or module name in the Code Intelligence database.

4. Select **Show imports** to include `import`, `use`, and `include` statements in the results.
5. In the **Search** drop–down list, select the desired language type. Note that only those language installations scanned into the Code Intelligence database can be selected. To scan a language installation in the Code Intelligence database, see Building the Code Intelligence Database for more information.
6. Click **Go**.

Alternatively, select the desired symbol or module in the Editor Pane, and then click **Code/Find Symbol in Object Browser**. The Object Browser is invoked (if not already launched) and displays the search results for the selected symbol or module.



Use the **Matches** pane to sort the symbols that match the search criteria. Sorting options are:

- Name
- Type
- Module
- File Path
- Line

# Source Code Control (Komodo Pro)

"Source code control" refers to the practice of storing files containing program source code (and other project artifacts) in a common repository. Using source code control (SCC), multiple developers can work on the same project (including the same project file) at the same time. The SCC repository can be queried for a detailed listing of the changes that occurred each time a file was edited. Files under source code control that are open for editing can also be reverted to their previous state.

Komodo's SCC integration works in conjunction with the CVS and Perforce source code control systems. From within Komodo, you can perform the following SCC actions:

- check files out of the repository
- add files to the repository
- remove files from the repository
- compare files in the editor against their repository versions (a "diff")
- submit files back to the repository
- revert files to their previous state

Depots, repositories, branches, and projects cannot be created or configured from within Komodo. This must be done using CVS or Perforce directly.

Source code control functions are accessible from the following locations:

- ***Toolbox Context Menu***: Right−click a file in the Toolbox to access the ***Source Control*** menu options. Source code control functions can also be performed on Folders in the Toolbox; right−click a folder and select ***Source Control on Contents*** to access the menu.
- ***Project Manager context menu***: Right−click a file in the Project Manager to access the ***Source Control*** menu options. SCC functions can also be performed on Folders in the Toolbox; right−click a folder and select ***Source Control on Contents*** to access the menu. Two source code control options are available from the Project Manager and Toolbox context menus:
    - ♦ ***Source Control*** applies source code functions to the selected file only.
    - ♦ ***Source Control on Contents*** applies source code functions to the contents of the selected project or folder.
- ***Editor Pane***: Right−click a file in the Editor Pane to access the ***Source Control*** menu options.
- ***Filename Tab***: Right−click the filename tab above the Editor Pane to access the ***Source Control*** menu options.
- ***File Menu***: Select the ***Source Control*** option from the ***File*** menu.

The ***Source Control*** submenu options are the same regardless of which method is used to access the menu.

## Configuring Source Code Control Integration

## Configuring CVS

### Installing the CVS Executable

To view the location of the CVS executable files found on your system and to determine which executable is used, select ***Edit***/***Preferences***/***Source Code Control***/***CVS***. If CVS is not properly configured, or if the CVS executable file you have is incompatible with Komodo, the CVS Source Code Control page in Komodo Preferences displays a message advising that CVS integration is disabled. To begin configuring CVS, click ***Download CVS*** and follow the instructions on the ASPN web site.

On Windows, determine the version of *cvs.exe* installed on your system by entering `cvs -v` at a command prompt. If this returns `Concurrent Versions System (CVSNT)` rather than `Concurrent Versions System (CVS)`, your CVS version is not compatible with Komodo. Linux uses the standard version of CVS.

The CVS executable must be located in a directory specified in your system's PATH environment variable.

On Windows 98 and Me, you must configure a `HOME` environment variable, for example `"HOME=c:\"`. Reboot before proceeding.

## CVS Over SSH

Some CVS repositories (e.g. SourceForge) will only support CVS access over SSH (secure shell). When accessing these repositories, an SSH client is required.

### Installing and Configuring Putty on Windows

Putty is a free SSH, Telnet and Rlogin client for Windows.

*1. Install Putty*

Download Putty (version 0.52 or greater) and associated programs from:

http://www.chiark.greenend.org.uk/~sgtatham/putty/

To connect to a CVS server via SSH, the following programs are required:

- *putty.exe*
- *puttygen.exe*
- *pageant.exe*
- *pscp.exe*
- *plink.exe*

Ensure that the directory where Putty is installed is specified in your system's `PATH` environment variable.

### 2. Generate the Putty Key

Run the `puttygen` utility. Configure as follows:

1. **Set Parameters**: Select either "SSH2 RSA" or "SSH2 DSA".
2. **Generate Key Pair**: Click the **Generate** button to generate the key pair. While the key is being generated, move the mouse pointer around the blank space to provide key randomness.
3. **Enter Key Passphrase**: Enter and confirm a passphrase for the key. Remember the passphrase – it is required later.
4. **Save Public Key**: Click the "Save public key" button and store the key in a file called *public1.key*.
5. **Save Private Key**: Click the **Save private key** button and store the key in a file called *private1.key*, in the same directory as the public key.
6. **Copy Key Contents**: Copy the contents of the public key field (at the top of the dialog box) to a file named *public1−openssh.key*. This key is required later.
7. **Close puttygen**

### 3. Load and Configure the Putty Authentication Agent

Run the `pageant` program. This loads the Putty Authentication Agent into the Windows System Tray.

Right−click the Pageant icon in the Windows System Tray. Select **Add Key**. Navigate to the directory where you saved the public and private keys in the previous step, and select the file *private1.key*.

### 4. Configure Putty To Use Pageant

Run the `putty` program. Configure as follows:

1. **Specify CVS Server**: On the **Session** page of the Configuration form, enter the host name or IP address of the CVS server.
2. **Specify Protocol**: On the **Session** page, in the **Protocol** field, select the "SSH" protocol.
3. **Create Saved Session**: In the **Saved Sessions** field, enter the host name again. Click the **Save** button.
4. **Configure Connection**: on the **Connection** page of the Configuration form, enter your username for the CVS server in the **Auto−login username** field.
5. **Configure SSH Protocol**: On the **SSH** page of the Configuration form, specify "2" for the **Preferred SSH protocol version**.
6. **Enable Agent Forwarding**: On the **Auth** page of the Configuration form, check **Allow agent forwarding**. In the **Private key file for authentication** field, specify the path and filename of the private key created above (*private1.key*).
7. **Save Session Information**: On the **Session** page of the Configuration form, click the **Save** button.

### 5. Store Public Key on CVS Server

You must store the public key file generated in step 2 (*public1−openssh.key*) on the CVS server.

1. ***Open Command Prompt Window***: Type `cmd` in the Windows Run dialog box.
2. ***Copy Public Key to Server***: At the command prompt, enter:

   ```
   pscp c:\path\to\public1-openssh.key username@cvs.server.com:public1-openssh.key
   ```

   ...where *c:\path\to\public1−openssh.key* specifies the location of the key file created in step two, and *username@cvs.server.com* specifies your user name on the CVS server and the URL of the CVS server. You are prompted to confirm the legitimacy of the host, and may be prompted to enter your password for the CVS server.
3. ***Connect Using Putty***: If necessary, run the `putty` program. In the ***Saved Sessions*** field, double−click the configuration created in Step 4. This establishes a connection to the CVS server.
4. ***Configure the Key on the Server***: After logging on to the CVS server, enter the following commands to configure the SSH key:

   ```
   mkdir ~/.ssh
   chmod 700 .ssh
   cat ~/public1-openssh.key >> ~/.ssh/authorized_keys
   rm ~/public1-openssh.key
   chmod 600 ~/.ssh/*
   ```

5. ***Log Off and Exit Putty***: Enter `exit` to close the session of the server.

## 6. Test the Configuration

Restart Putty. In the ***Saved Sessions*** field, double−click the configuration created in Step 4. You should not be prompted to log in. If you are, the configuration failed. Review the steps above and ensure that they were completed correctly.

## 7. Check Out a CVS Module

1. ***Create Local CVS Directory***: Create a directory to store a copy of the CVS repository.
2. ***Copy Files to Local Directory***: At a command prompt, enter:

   ```
   set CVS_RSH=plink
   set PLINK_PROTOCOL=ssh
   cvs -d :ext:username@cvs.server.com:/repository_name co cvs_module
   ```

   ...where *username@cvs.server.com* specifies your username on the CVS server and the URL of the CVS server, *repository_name* specifies the name of the repository on the server, and *cvs_module* specifies the name of the module in the chosen working repository.

   Login is handled by SSH. The files are copied to the local system. These environment variables do not interfere with non−SSH repositories.

Ensure that these variables are permanently configured in your system environment (for example, by adding them to the *autoexec.bat* file or configuring them in the system properties). Reboot Windows Me and 9x systems after adding environment variables.

*8. Using Komodo and CVS*

Before starting Komodo, perform the following steps:

- *Set PLINK_PROTOCOL=ssh*: In the user environment, set the environment variable `PLINK_PROTOCOL` to "ssh".
- *Set CVS_RSH=plink*: In the user environment, set the environment variable `CVS_RSH` to "plink".
- (Windows 9x and Me only)*Set HOME=C:\*: On Windows 98 and Windows Me, configure the HOME environment variable in the *autoexec.bat* file.
- *Ensure Pageant Is Running*: Run the `pageant` program to enable the authentication agent. Ensure that the *private1.key* is loaded.

   You can also execute Pageant and load the key via a batch file. For example:

   ```
   C:\PuTTY\pageant.exe c:\path\to\private.key c:\path\to\private2.key
   ```

If you are running Windows 9x or Windows Me, permanently configure these variables in the environment (for example, by adding them to the *autoexec.bat* file) and reboot before proceeding.

## Configuring Windows/Cygwin−SSH or Linux/SSH

To configure CVS to use SSH, refer to [http://xml.apache.org/forrest/community/howto/cvs−ssh/howto−cvs−ssh.html](http://xml.apache.org/forrest/community/howto/cvs−ssh/howto−cvs−ssh.html).

On all platforms, create an environment variable as follows:

   ```
   CVS_RSH=ssh
   ```

CVS determines when to use SSH, depending on how you check out the modules. If you use the "cvs login" method with the "pserver" protocol, CVS does not use SSH, even if CVS_RSH=ssh is set in the environment.

On Windows, also configure the cygwin SSH Agent as follows:

1. Open a cygwin shell.
2. Enter `exec ssh-agent bash`.
3. Enter `ssh-add`.
4. To check out a CVS module, enter:

   ```
   cvs -d :ext:username@cvs.server.com:/repository_name co cvs_module
   ```

...where *username@cvs.server.com* specifies your username on the CVS server and the URL of the CVS server, *repository_name* specifies the name of the repository on the server, and *cvs_module* specifies the name of the module in the chosen working repository.

5. Start Komodo within the cygwin shell as follows:

```
/path/to/komodo/komodo.exe
```

After completing the configuration steps above, follow these steps to open Komodo with CVS−SSH enabled:

1. Open a cygwin shell.
2. Enter `exec ssh-agent bash`.
3. Start Komodo within the cygwin shell as follows:

```
/path/to/komodo/komodo.exe
```

## Configuring Perforce

Ensure that the command−line version of Perforce ("P4") is correctly installed and functional before working with a Perforce repository within Komodo. The `p4.exe` file must be located in a directory that is specified in your PATH environment variable.

Users of Perforce's P4CONFIG feature may find that Komodo's source code control doesn't work unless Komodo is started from within the client view of the Perforce repository.

## Configuring Preferences

Use the *Source Code Control* page in Komodo's Preferences to configure Source Code Control integration. To open the Komodo Preferences dialog box, select *Edit*/*Preferences*.

# Using Source Code Control

## SCC Toolbar, Menus and Output Tab

**Source Code Control Toolbar**

Access common source code control commands from the SCC Toolbar at the top of the Komodo workspace. Refer to the command descriptions below for more information. The toolbar commands only apply to the file currently active in the Editor Pane.

**Source Code Control Menus**

To access source code control functions, select *File/Source Control*.

Invoke source code control context menus by right–clicking files or folders in the following areas:

- Project Manager
- Toolbox
- Editor Pane

**Source Code Control Output Tab and Status Messages**

The *SCC Output* tab is located in the Bottom Pane of the Komodo workspace. As you execute source code control commands, such as editing or checking in files, details of the commands are displayed on the *SCC Output* tab.

Error messages and warnings are also displayed on the *SCC Output* tab. Additionally, error messages and warnings are displayed on the status bar in the bottom left corner of the Komodo workspace.

# Source Code Control Commands

As described above, source code control commands are invoked from the toolbar, the *File* menu and the Source Control context menu. The following commands are available, depending on the context:

- *Add*: Add a file from a designated source code directory on your local drive to the source code repository.
- *Edit*: Check out a file from the current source code repository.
- *Revert Changes*: Check the file back into the repository, abandoning any changes made since it was checked out.
- *Remove*: Delete the file from both the source code repository and the corresponding local directory.
- *Update*: When the local version no longer matches the repository version of a file, select this command to update the local version of the file.
- *Diff (Compare Files)*: Compare the version of a file open in the Editor Pane with the version in

the source code repository. Depending on the setting in Preferences, the diff display is shown on another Komodo editor tab or in a separate window. If the display style for diffs (*Edit|Preferences|Source Code Control*) is set to *Create new window*, press 'F9' or select *Jump to Corresponding Line* to open and/or shift focus to the original file in the Editor Pane. If viewing a diff in an editor tab, right−click and select *Jump to Corresponding Line* to shift focus to the editor tab containing the source code. Selecting this option opens the source code tab in the Editor Pane if it is not already open.

- *Commit Changes*: Submit the file back to the source code repository.

Under Perforce, to open files for edit from within a Komodo project, you must first add files to the project using options on the *Project* menu. To accomplish the same under CVS, use the following procedure:

1. At the command prompt, enter `cvs co <projname>` to open working copies of the files.
2. On the *Projects* tab, right−click the project and select *Import from File System*.
3. In the *Import from File System* dialog box, specify the location of the files to be imported and click *Next*.
4. Click *OK* to confirm changes to the project.

## File Status Icons

If Perforce or CVS is enabled in Komodo's Preferences, the status of files in the source code repository is indicated by icons that appear on file tabs in the Editor Pane and next to files and projects on the Projects tab.

The icons can appear in a variety of combinations, depending on the status of the file and where they are displayed in the Komodo workspace. For example, a green circle next to a padlock on a tab in the Editor Pane indicates that the file is open for edit and that the version of the file in your local directory is in sync with the version in the source code repository.

✚ The file is being added to the source code repository.

✘ The file is being deleted from the source code repository.

✔ The file is open for edit.

◎ The version of the file in your local directory is in sync with the version in the source code repository.

🔒 The file is read−only.

◎ The version of the file in your local directory is out of sync with the version in the source code repository.

⬣ There is a conflict between the version of the file in your local directory and the source file that cannot be resolved by simply syncing your directory with the source code repository. The discrepancy must be manually resolved.

# GUI Builder (Komodo Pro)

Komodo's GUI Builder is an application used to create graphical user interfaces, such as dialogs containing radiobuttons and list boxes. The GUI Builder supports building applications using TK with an extended widget set that includes BWidgets and IWidgets. The GUI Builder is integrated with Komodo via dialog projects, which consist of a top–level container (with the extension ".ui") and program files that contain the GUI code and the application code.

## Creating Dialog Projects

GUI Builder projects are referenced as "dialog" components within Komodo projects or the Toolbox. To create a new dialog project, right–click a project name or a folder within a project and select *Add/New Dialog*. Alternatively, select *Project/Add/New Dialog* or *Toolbox/Add/New Dialog* from the drop–down menu. Provide the following information about the dialog project:

1. *Specify Dialog Project File*: The files related to the dialog are stored in a project file with a ".ui" extension.
2. *Specify Target Language*: The GUI Builder can generate code in various languages and language versions:

    - *Perl/Tk (8.0)*
    - *Perl/Tk (8.4)*
    - *Tcl/Tk (8.3)*
    - *Tcl/Tk (8.4)*
    - *Python/Tkinter (8.3)*
    - *Python/Tkinter (8.4)*

3. Click *OK*. The GUI Builder application loads.

Use the GUI Builder to configure the dialog, as described in the sections below.

When you save a new dialog in the GUI Builder, the following items are created in either the Toolbox or the Project Manager (depending on where the *New Dialog* command was invoked):

- *Dialog.ui Project File*: The name specified for the dialog is suffixed with ".ui". This dialog project file contains two files:
    - *Dialog.<language>*: Edit this file to add functionality to widgets.
    - *Dialog_ui.<language>*: This file is automatically generated by the GUI Builder and should not be modified.

## Modifying an Existing Dialog

To modify the graphical properties of the dialog, right–click the dialog project (with the ".ui" extension) and select **Edit Dialog**. This launches the GUI Builder and loads the project. Code within the sections described in Adding Code to a Dialog are not affected by modifying the graphical properties of a dialog.

## Adding Code to a Dialog

To modify the functions within the dialog, open the *Dialog.<language>* file, (where "Dialog" is the name of the project, and "<language>" is the standard language suffix) located beneath the dialog project file. Do not edit the *Dialog_ui.<language>* file because this file is automatically generated by the GUI Builder. Therefore, changes could be lost. Note, however, that code entered within the following sections is preserved, even if the dialog is subsequently modified:

- *Tcl*: `proc` sections
- *Perl*: `sub` sections
- *Python*: `def` sections

Open the callback file by selecting **Open File** on the **Projects** context menu, or by double–clicking the filename on the **Projects** tab. Comments included in the callback file indicate where to insert callback code and user code.

## Testing the GUI

- To run the application, click the green arrow button on the toolbar, or select **Commands/Start Test**.
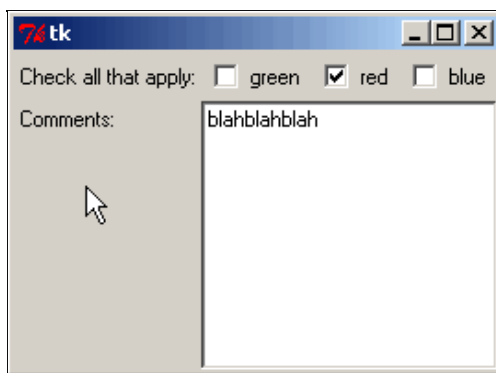


- To stop a running application, click the red stop button on the toolbar, or select **Commands/Stop Test**.

# Viewing Code in the Komodo Editor

When you save the GUI in the GUI Builder, corresponding dialog project and language files are created in Komodo, either in the Project Manager or the Toolbox, depending on where the *New Dialog* command was invoked.

To display the code in the Komodo editor, select *Commands/View Code*. Alternatively, in Komodo, open the *Dialog.<language>* file beneath the dialog project.

# Dialog Project Options

Dialog projects consist of a "container" file (with the extension ".ui") and the source files containing the code for the GUI and the code for the application. Options for dialog projects are described below; options for files contained in dialog projects are the same as options for regular files.

To access options for the selected dialog, do one of the following:

- *Toolbox/dialog_name/option* or *Project/dialog_name/option*: When a dialog is selected in the Project Manager or Toolbox, use the *Project* or *Toolbox* drop−down menus to access the list of options. The name of the dialog currently selected in the Project Manager or the Toolbox is displayed in the drop−down menu.
- *Context Menu*: Right−click a dialog in a project or the Toolbox and select the desired option.

The following options are available:

- *Edit Dialog*: Use this option to launch the GUI Builder application.
- *Test Dialog*: This option runs the application.
- *Refresh Status*: This option checks the read/write disk status of the dialog project file. If the dialog project file is stored in a source code control system, *Refresh Status* also checks the repository status of the file.
- *Cut/Copy/Paste*: These options are used to remove the dialog from a project or the Toolbox, or to move dialog projects between the Project Manager and the Toolbox (and vice versa).
- *Export as Project File*: When this option is invoked, a new project file is created that contains the dialog from which the option is invoked. You are prompted to provide the name of the new project file and the directory where it is stored. To open the new project file, select *File/Open/Project*.
- *Export Package*: Dialogs can be archived and distributed among multiple Komodo users via "packages". Packages are compressed archive files that contain the dialog from which the *Export Package* option was invoked. Packages are stored in files with a ".kpz" extension, and can be opened by any archiving utility that supports `libz` (for example WinZip). The *Export Package* option differs from the *Export as Project File* option in that copies of filesystem−based components (such as files and dialog projects) are included in the archive. Conversely, *Export as*

*Project File* creates a project with a reference to the component's original location and does not create copies of the components. When *Export Package* is invoked, you are prompted for a name and file location for the package. Exported packages can only be imported into "container" objects in Komodo, such as projects, the Toolbox, and folders within projects and the Toolbox. See Toolbox – Exporting and Importing Toolbox Contents, Projects – Importing and Exporting Projects via Packages, or Folders – Import Contents from Package for more information.

- *Source Control*: If Komodo is configured to work in conjunction with a Source Code Control system, this option is used to access SCC commands for the dialog project file.
- *Source Control on Contents*: If Komodo is configured to work in conjunction with a Source Code Control system, this option is used to invoke SCC commands that operate on the contents of the dialog project.
- *Delete*: To remove a dialog from a project or the Toolbox, select this option. The dialog project (and the files it contains) is not deleted from disk.

# GUI Builder Overview
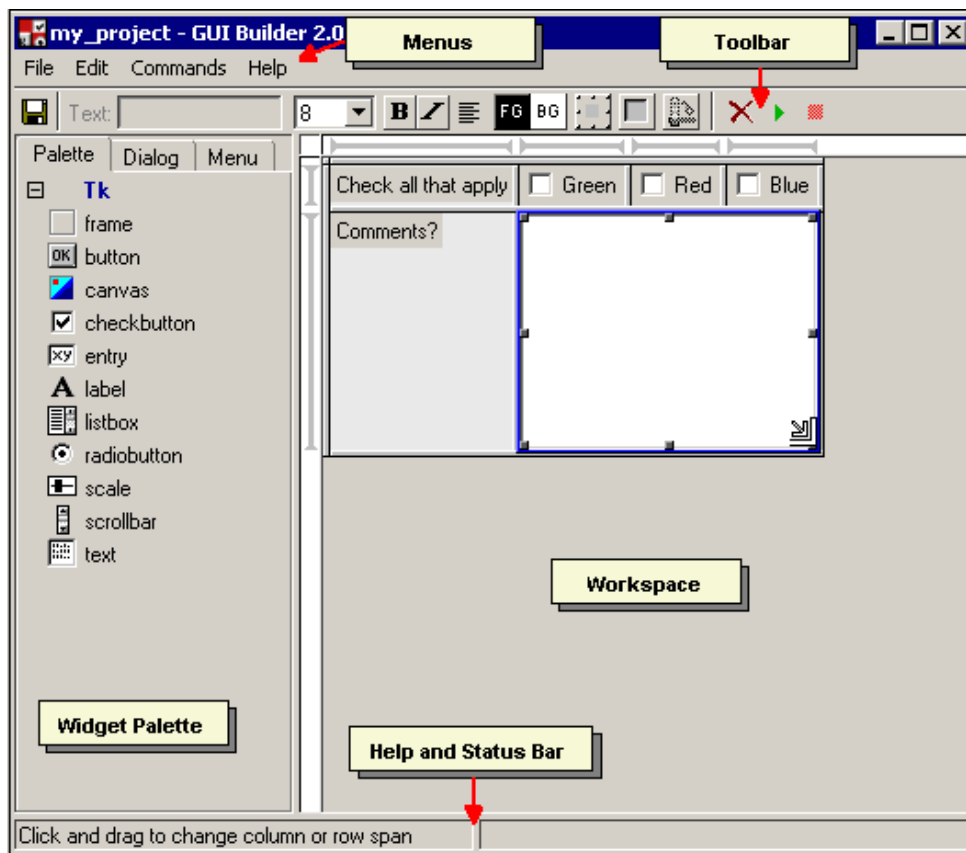


*Quick Overview*

- Add rows and columns by double–clicking grid lines.

- Resize cells by dragging the grid lines.
- Drag and drop widgets from the *Palette* tab to a cell in the workspace.
- Double−click widgets in the workspace to configure their properties.
- Use the *Menu* tab to create menus and add menu item widgets.
- Select *Edit/View Menus* or press 'Ctrl'+'M' to show or hide menus in the workspace.
- Use the *Dialog* tab to manage the structure of GUI Builder projects.
- Use the *Menu* tab to add cascading menus and menu items.
- Click the green arrow button on the toolbar or select *Commands/Start Test* to test the interface.
- Save the GUI to create the files in Komodo.

*Limitations*

- There are numerous "container" widgets available for building UIs in the GUI Builder that vary depending on language. These include frame, notebook and panedwindow, among others. The only "container" widgets that GUI Builder supports in full are the core frame and labelframe widgets. Placement widgets inside the other "container" widgets must be done by hand currently.
- Specification of the dialog title must be done in the user code.
- The GUI Builder does not check whether you have the requisite Tk installation and associated widget modules (if necessary). This means you may be able to create UIs that you cannot actually test.
- If you create widgets, generate code, and later rename or remove those widgets, any generated command code for those widgets is not removed. This is because the round−trip code generation doesn't know about what has been deleted, just what currently exists, and pre−existing code is just parsed and reinserted.

# Workspace

The workspace is used to design the GUI. See Adding and Resizing Rows and Columns and Adding Widgets to a Dialog for more about using the GUI Builder workspace.
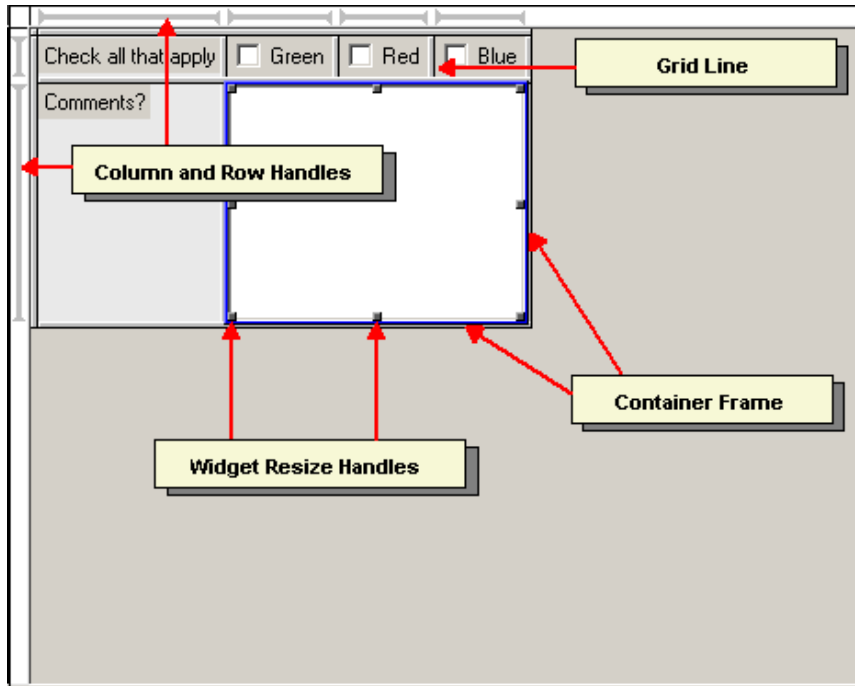
The column and row handles at the top and the left of the workspace indicate the selected cell. When a cell is selected, the column and row handles are displayed in blue. When a single column or row handle is selected, it is displayed in red. Use the Appearance Preferences to set color and size preferences for the workspace.

## Toolbar

The toolbar contains buttons for the most common widget configuration options. You must select the widget in the workspace or on the Dialog tab before changing widget properties. For a description of how to use the toolbar buttons to configure basic widget properties, see Configuring Widget Properties.
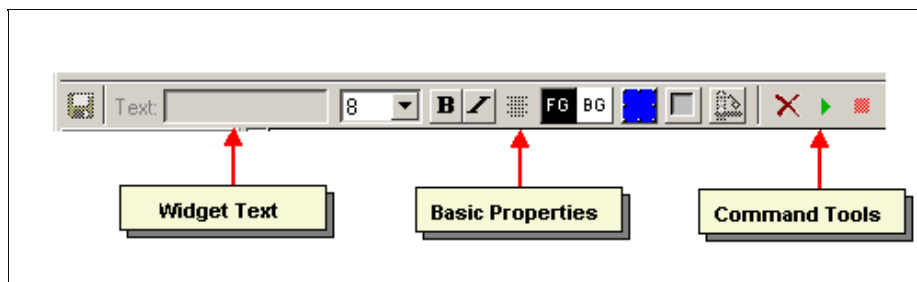


From left to right, the tools are:

          Save file

| | |
|---|---|
| Text: red | Widget text |
| 8 ▼ | Widget text size |
| **B** *I* | Widget text bold and italic |
| ≡ | Widget text alignment |
| FG BG | Widget foreground and background color |
| ▪ | Widget position in cell |
| ▫ | Widget border and relief |
| ▣ | Widget orientation (for the scrollbar widget) |
| ✕ | Widget delete |
| ▶ ▦ | Run or stop running interface |

## Widget Palette Tab

The *Palette* tab is the tab displayed by default in the GUI Builder. Its contents vary depending on the language that was selected when a dialog was added to the Toolbox or Project Manager. For more about language selection, see Creating Dialog Projects.

Use the *Next Widget* and *Previous Widget* commands on the *Navigate* submenu to move from one widget to another in the workspace.

### Widget Properties

Properties vary according to the selected target language. Select *File/Project Settings* to view the selected target language.

- Tk
  - frame
  - [OK] button
  - canvas
  - ☑ checkbutton
  - [xy] entry
  - **A** label
  - listbox
  - ⊙ radiobutton
  - scale
  - scrollbar
  - text

*Perl/Tk 8.0 Palette tab*

- To change the properties of an individual widget, double−click the widget in the workspace.
- To change the default properties for a widget, double−click the widget on the *Palette* tab.

The settings available in the widget properties dialog box vary depending on the type of widget that is selected.

## Dialog Tab

The *Dialog* tab provides a hierarchical overview of a GUI Builder project. As you add controls and menus to the workspace, a structured outline of the chosen widgets is displayed on the *Dialog* tab. The contents of each project are displayed beneath a collapsible "Container" node that appears at the top of the tab. Double−click widgets on the *Dialog* tab to view or edit their properties.

Use commands on the *Navigate* submenu to determine the relationships between widgets. Select *Navigate/Select Parent* to highlight the parent element of a selected widget. Select *Navigate/Select 1st Child* to highlight the widget that is one level below the selected element in the project hierarchy. Note that these commands are only available if you select a widget that has a "parent/child" relationship with one or more other widgets.
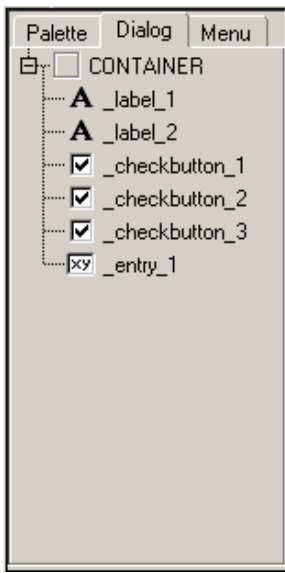
## Menu Tab
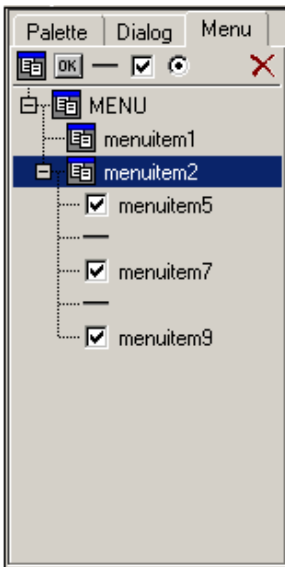
Use the widgets on the *Menu* tab to add cascading menus and menu items to a GUI. Just as the Dialog tab provides a hierarchical overview of the controls and buttons in a dialog, the *Menu* tab displays the menus and menu options that make up the menu bar in a GUI Builder project. Select *Edit/View Menus* to display the menus at the top of the workspace. The available menu widgets are (from left to right) "cascade", "command", "separator", "checkbutton", and "radiobutton". For directions on adding menu widgets to the workspace, see Adding Widgets to a Dialog.

Add drop−down menus and menu items to your GUI using the widgets on the *Menu* tab. Select *Edit/View Menus* to display menus in the workspace.

## Status Bar

The status bar displays warnings, progress messages and diagnostics. Help messages are displayed only if **Show Statusbar Help** is selected in GUI Builder Preferences. Help messages are displayed for the object under the current cursor position. In addition, when you drag a widget or resize handle, the status bar displays the grid position or size.
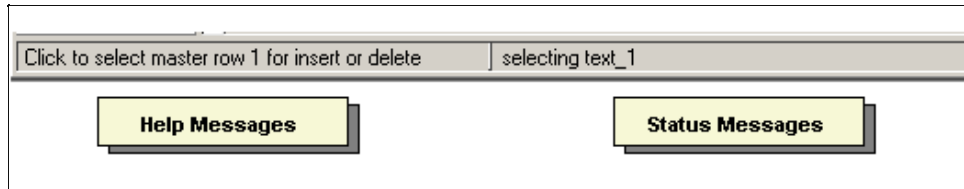


# Building GUI Applications

## Adding and Resizing Rows and Columns

When you launch the GUI Builder, the workspace contains a default grid with two rows and two columns. You can create a new dialog with this default grid, or modify the default grid using the options described below.

To add rows and columns:

- *Add a Row or Column*: Click on a grid line, then press the 'Insert' key. Alternatively, double–click the grid line.
- *Add a Row and a Column*: Click in a grid cell, then press the 'Insert' key.

To delete rows and columns:

- *Delete a Row or Column*: Click on a grid line, then press the 'Delete' key. Rows and columns cannot be deleted if they contain widgets.
- *Delete a Row and a Column*: Click in a grid cell, then press the 'Delete' key. Rows and columns cannot be deleted if they contain widgets.

To resize rows and columns:

- *Configure Row and Column Size, Weight and Widget Padding*: Click in a grid cell, then select *Edit/Row & Column Properties*. Use this dialog to change size, weight and the padding around widgets.
- *Resize Row or Column*: Click and drag a grid line.

To straddle rows and columns:

Grid cells can be resized to span multiple columns or rows. Note that straddling rows and columns is only possible when the grid cell that you want to expand contains a widget, and the adjacent cells do not contain widgets. To straddle rows or columns:

1. Click to select the cell containing the widget.
2. Position the cursor over one of the resize handles. The cursor changes to an arrow pointing towards a line.
3. Click and drag across the adjacent cells.

## Adding Widgets

To add widgets from the Palette tab:

- On the *Palette* tab, click the desired widget, drag it into a cell in the workspace.

To add widgets from the Menu tab:

1. Click the *MENU* container on the *Menu* tab or, if *Edit*/*View Menus* has been selected, click *New Cascade* at the top of the workspace, to add a new menu to the GUI.
2. Select a top−level "menuitem" on the *Menu* tab to make the Menu widgets available. Use the "cascade", "command", "separator", "checkbutton", and "radiobutton" menu item widgets to create menus.

## Deleting Widgets

To remove a widget or menu item from the workspace, select it and click the "delete" button on the toolbar for widgets, or click the "delete" button on the Menu tab for menu items.

## Configuring Widget Properties

### Basic Widget Properties

1. Select the widget in the GUI Builder workspace. The associated options become available on the toolbar.
2. Use the controls on the toolbar to edit widget properties.

### Advanced Widget Properties

- Select the widget in the GUI Builder workspace, then double−click it or select *Edit*/*Widget Properties* to display a widget properties dialog box.

## Resizing a Widget

- To make a widget resizable, click the column or row handle once to select, then again to toggle the resizable setting. When the arrows on the handle point outwards, the widget becomes resizable. Note that this characteristic only has an effect if the widget is configured to adhere to two (or four) sides of the cell. (The scrollbar widget is resizable by default.)
- To make a widget span rows or columns, point the mouse at the edge of the widget and drag the widget resize handle when the mouse pointer changes (grid lines spanned by the widget disappear).

## Attaching Scrollbars to a Widget

- To attach scrollbars to widgets, first insert the desired widget and scrollbar in the workspace. Then select the widget, and select *Commands/Attach Scrollbars*. You are prompted to specify which scrollbars you wish to attach. Scrollbars can only be attached to scrollable widgets.

## Loading a GUI Builder Project into a Frame Widget

- *Load Project Into Frame*: This option becomes available on the *Commands* menu when a "frame" or "labelframe" widget is added to the workspace. Select *Commands/Load Project Into Frame* to navigate to a GUI component contained in another GUI Builder project (.ui) file, and then load it into the selected frame. This feature is particularly useful if you have an interface component that you want to insert repeatedly in your GUI projects.

# GUI Builder Preferences

## General Preferences

- *Insert if widget dropped on gridline*: If this option is enabled, a widget is automatically inserted in a new row or column when you drop the widget on a grid line. If this option is not enabled, the GUI Builder ignores widgets dropped on grid lines.
- *Confirm saves before test*: If this option is enabled, you are prompted to save changed files before testing the application.
- *Confirm widget delete*: If this option is enabled, you are prompted to confirm that you want to delete a widget in the workspace.
- *Autosave on quit*: If this option is enabled, changed files are automatically saved. Otherwise, you are prompted to save changed files.

- *Mouse Gravity*: This setting determines how far the mouse pointer must move to invoke the action. Set the value lower to make the mouse more responsive and higher to give the mouse more tolerance.
- *Show Tooltips*: If this option is enabled, yellow pop–up hints are displayed whenever you hover the mouse pointer over an object.
- *Show Statusbar Help*: If this option is enabled, usage tips are displayed in the bottom right corner of the GUI Builder window.

## Appearance Preferences

- *Selection Color*: When a widget or grid line is selected (by clicking on it), it is outlined in the color specified here.
- *Grid Background*: This setting specifies the color of the grid background.
- *Active Over Color*: When you hover the mouse pointer over a grid line, the line changes to the color specified here.
- *Frame Background*: Set the color for the background of the frame widget.
- *Show Grid Lines*: If this option is enabled, grid lines are displayed. Alternatively, click the button at the intersection of the row and column handles (top left corner of the workspace) to toggle grid lines on and off.
- *Grid Line Thickness*: Set the thickness of the grid line.
- *Default Grid Spacing*: When you create a new GUI, cells are created by default with the pixel size specified here.
- *Control Points Size*: Set the size of the Widget Resize Handles.

# Tk and Widget Reference

Reference pages for individual widgets are included in the ActiveTcl documentation. Select *Help/Help for Languages* to access either a local version (if installed) or web version of the ActiveTcl documentation.

"Tk" is a toolkit for building graphical user interfaces. It was originally developed for Tcl, but has since been adapted for Perl and Python. The ActivePerl, ActivePython and ActiveTcl language distributions include documentation regarding each language's Tk support. (Note that the Tcl/Tk reference is divided into three categories: the core (Tk) manual and two groups of extensions (BWidgets and IWidgets).

Select *Help/Help for Languages* to launch the local version of the language documentation (if installed) or to access the web version. Links to the language–specific Tk reference pages on ActiveState's ASPN site are provided below:

- Perl/Tk
- Tcl/Tk:
    - Tk Manual

- ◆ BWidgets
- ◆ IWidgets
- Python/Tkinter

---

# Using the Rx Toolkit

Komodo's Rx Toolkit is a tool for building, editing and debugging regular expressions. Build a regular expression in the *Regular Expression* pane, and enter the sample search string in the *Search Text* pane. Adjust the regular expression as necessary to produce the desired matches in the *Match Results* pane.

*Note*: Although the Rx Toolkit has a Python back end, most Perl, PHP and Tcl regular expression syntax is also supported.

If you are new to Regular Expressions, see the Regular Expressions Primer. In addition, online references for Python, Perl, PHP and Tcl regular expressions are available via the Rx Toolkit's *Help* menu.

To open the Rx Toolkit, do one of the following:

- On the Standard Toolbar, click ⬚.

  or

- On the *Tools* menu, select *Rx Toolkit*.

To close the Rx Toolkit:

- Click the "X" button in the top right corner of the Rx Toolkit window.

*Rx Toolkit Quick Reference*

- Create and edit regular expressions in the *Regular Expression* pane.
- Apply one or more metacharacters to a regular expression by selecting them from the *Shortcuts* menu or entering them in the *Regular Expression* pane.
- Select a match type using the buttons at the top of the Rx Toolkit window.
- Apply modifiers to a regular expression by selecting one or more *Modifiers* check boxes.
- Test a regular expression against a string by entering text in the *Search Text* pane, or by clicking the *Search Text* pane's *Open* button and navigating to a file.
- Right−click a pane in the Rx Toolkit to access a context menu used to *Show/Hide Indentation Guides*, *Show/Hide Line Numbers*, *Show/Hide EOL Markers*, *Show/Hide Whitespace* and turn *Word*

*Tutorial*

- Regular Expression Primer

*Feature Showcase*

- test a regular expression

*Wrap* on and off. This context menu is *not* available in the *Match Results* pane.
- Select *Help/Load Sample Regex and Search Text* to reload the default regular expression and search text if it has been replaced or deleted.
- Double–click a result in the *Match Results* pane to highlight the corresponding text in the *Search Text* pane.

# Creating Regular Expressions

Use the Rx Toolkit to create and edit regular expressions. Create regular expressions by typing them in the *Regular Expression* pane. A regular expression can include metacharacters, anchors, quantifiers, digits, and alphanumeric characters.

Many of the display characteristics available in the Editor Pane can be enabled in the *Regular Expression* field. However, these characteristics must be manually enabled via key bindings. For example, to display line numbers in the *Regular Expression* field, press 'Ctrl'+'Shift'+'6' (if the default key binding scheme is in effect).

*Note: Do not enclose regular expressions in forward slashes ("/"). The Rx Toolkit does not recognize enclosing slashes.*
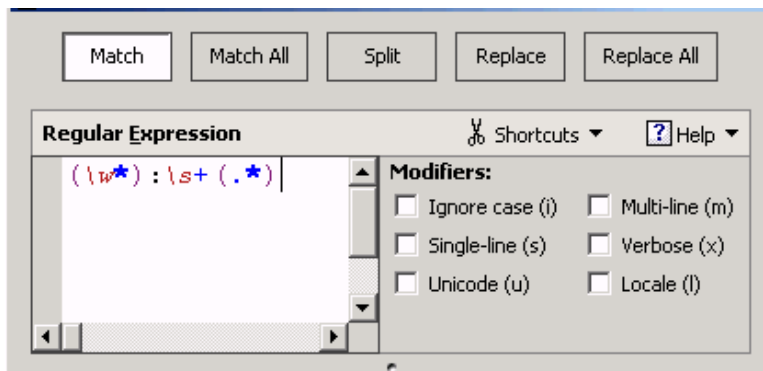


## Adding Metacharacters to a Regular Expression

The *Shortcuts* menu provides a list of all of the metacharacters that are valid in the Rx Toolkit.

To add a metacharacter to a regular expression:

1. Click *Shortcuts* to the right of the *Regular Expression* pane.
2. Select a metacharacter from the list. The metacharacter is added to the *Regular Expression* pane.

or

- In the *Regular Expression* pane, type a metacharacter.

## Setting the Match Type

The buttons at the top of the Rx Toolkit Window determine which function is used to match the regular expression to the search text. The options are based on module–level functions of Python's re module. Choose from the following options:

- *Match*: Scan the search text for the first instance of a match for the regular expression.
- *Match All*: Find all matches for the regular expression in the search text and display them as a series of matches in the *Match Results* pane.
- *Split*: Scan the search text for regular expression matches and split the string apart wherever there is match. Each match is displayed on a separate line in the *Match Results* pane.
- *Replace*: Scan the text for the first occurrence of the regular expression and replace it with text specified in the *Replacement* pane. The *Replacement* pane is only displayed when either the *Replace* or *Replace All* is selected.
- *Replace All*: Scan the text for all occurrences of the regular expression and replace them with the text specified in the *Replacement* pane. The *Replacement* pane is only displayed when either the *Replace* or *Replace All* is selected.

## Adding Modifiers to a Regular Expression

Add modifiers to regular expression by selecting one or more of the check boxes to the right of the *Regular Expression* pane:

*Note*: You must use the *Modifiers* check boxes to add modifiers to a regular expression. The Rx Toolkit does not recognize modifiers entered in the *Regular Expression* pane.

- *Ignore Case*: Ignore alphabetic case distinctions while matching. Use this to avoid specifying the case in the pattern you are trying to match.
- *Multi–line Mode*: Let caret "^" and dollar "$" match next to newline characters. Use this when a pattern is more than one line long and has at least one newline character.
- *Single–line Mode*: Let dot "." match newline characters. Use this when a pattern is more than one line long and has at least one newline character.
- *Verbose*: Permit the use of whitespace and comments in regular expressions. Use this to pretty print and/or add comments to regular expressions.
- *Unicode (Python Only)*: Make the special characters "\w", "\W", "\b" and "\B" dependent on Unicode character properties.
- *Locale (Python Only)*: Make the special characters "\w", "\W", "\b" and "\B" dependent on the current locale.

# Evaluating Regular Expressions
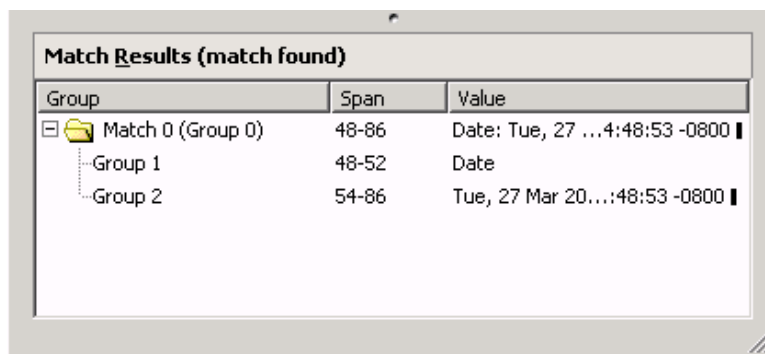
A debugged regular expression correctly matches the intended patterns and provides information about which variable contains which pattern.

*If there is a match...*

- Matches are displayed in the *Match Results* pane.
- Komodo highlights the search text string in yellow.

*If there is no match...*

- If a regular expression does not match the test string, an error message is displayed in the *Match Results* pane.
- If a regular expression is invalid, the title bar of the *Match Results* pane becomes red and details of the error are displayed in that pane.



## Match Results

If a regular expression collects multiple words, phrases or numbers and stores them in groups, the *Match Results* pane displays details of the contents of each group.

The *Match Results* pane is displayed with as many as four columns, depending on which match type is selected. The *Group* column displays a folder for each match; the folders contain numbered group variables. The *Span* column displays the length in characters of each match. The *Value* column lists the values of each variable.

# Modifier Examples

This section shows some sample regular expressions with various modifiers applied. In all examples, the default match type (*Match All*) is assumed:

- [Ignore Case](#)
- [Multi–line](#)
- [Single–line](#)
- [Multi–line and Single–line](#)
- [Verbose](#)

## Using Ignore Case

The *Ignore case* modifier ignores alphabetic case distinctions while matching. Use this when you do not want to specify the case in the pattern you are trying to match.

To match the following test string...

```
Testing123
```

...you could use the following regular expression with *Ignore case* selected:

```
^([a-z]+)(\d+)
```

The following results are displayed in the *Match Results* pane:

| Match Results (1 match found) | | |
|---|---|---|
| Group | Span | Value |
| ⊟ 📁 Match 0 (Group 0) | 0-10 | Testing123 |
| Group 1 | 0-7 | Testing |
| Group 2 | 7-10 | 123 |

*Discussion*

This regular expression matches the entire test string.

The `^` matches the beginning of a string. The `[a-z]` matches any lowercase letter from "a" to "z". The `+` matches any lowercase letter from "a" to "z" one or more times. The *Ignore case* modifier lets the regular expression match any uppercase or lowercase letters. Therefore `^([a-z]+)` matches "Testing". The `(\d+)` matches any digit one or more times, so it matches "123".

## Using Multi–Line Mode

The *Multi–line* modifier allows ^ and $ to match next to newline characters. Use this when a pattern is more than one line long and has at least one newline character.

To match the subject part of the following test string...

```
"okay?"
```

...you could use the following regular expression with *Multi–line* selected:

```
^(\"okay\?\")
```

The following results are displayed in the *Match Results* pane:

| Match Results (1 match found) | | |
|---|---|---|
| Group | Span | Value |
| ⊟ 📁 Match 0 (Group 0) | 0-7 | "okay?" |
| Group 1 | 0-7 | "okay?" |

*Discussion*

This regular expression matches the entire test string.

The ^ matches the beginning of any line. The \" matches the double quotes in the test string. The string matches the literal word "okay". The \? matches the question mark "?". The \" matches the terminal double quotes. There is only one variable group in this regular expression, and it contains the entire test string.

## Using Single–Line Mode

The *Single–line* modifier mode allows "." to match newline characters. Use this when a pattern is more than one line long, has at least one newline character, and you want to match newline characters.

To match the following test string...

```
Subject: Why did this
work?
```

...you could use the following regular expression with *Single–line* selected:

```
(:[\t ]+)(.*)work\?
```

The following results are displayed in the *Match Results* pane:

| Match Results (match found) | | |
|---|---|---|
| Group | Span | Value |
| ⊟ 📁 Match 0 (Group 0) | 7-28 | : Why did this ▌ ▌work? |
| Group 1 | 7-9 | : |
| Group 2 | 9-23 | Why did this ▌ ▌ |

*Discussion*

This regular expression matches everything in the test string following the word "Subject", including the colon and the question mark.

The (\s+) matches any space one or more times, so it matches the space after the colon. The (.*) matches any character zero or more times, and the *Single−line* modifier allows the period to match the newline character. Therefore (.*) matches "Why did this <newline> match". The \? matches the terminal question mark "?".

## Using Multi−line Mode and Single−line Mode

To match more of the following test string...

```
Subject: Why did this
work?
```

...you would need both the *Multi−line* and *Single−line* modifiers selected for this regular expression:

```
([\t ]+)(.*)^work\?
```

The following results are displayed in the *Match Results* pane:

| Match Results (1 match found) | | |
|---|---|---|
| Group | Span | Value |
| ⊟ 📁 Match 0 (Group 0) | 8-28 | Why did this ▌ ▌work? |
| Group 1 | 8-9 | |
| Group 2 | 9-23 | Why did this ▌ ▌ |

*Discussion*

This regular expression matches everything in the test string following the word "Subject", including the colon and the question mark.

The ([\t ]+) matches a Tab character or a space one or more times, which matches the space after the colon. The (.*) matches any character zero or more times, which matches "Why did this <newline>". The ^work matches the literal "work" on the second line. The \? matches the terminal question mark

"?".

If you used only the *Single–line* modifier, this match would fail because the caret "^" would only match the beginning of a string.

If you used only the *Multi–line* modifier, this match would fail because the period "." would not match the newline character.

## Using Verbose

The *Verbose* modifier ignores whitespace and comments in the regular expression. Use this when you want to pretty print and/or add comments to a regular expression.

To match the following test string...

```
testing123
```

...you could use the following regular expression with the *Verbose* modifier selected:

```
(.*?)  (\d+)  # this matches testing123
```

The following results are displayed in the *Match Results* pane:

| Match Results (match found) | | |
|---|---|---|
| Group | Span | Value |
| ⊟ 🗀 Match 0 (Group 0) | 0-10 | testing123 |
| Group 1 | 0-7 | testing |
| Group 2 | 7-10 | 123 |

*Discussion*

This regular expression matches the entire test string.

The .* matches any character zero or more times, the ? makes the * not greedy, and the *Verbose* modifier ignores the spaces after the (.*?). Therefore, (.*?) matches "testing" and populates the "Group 1" variable. The (\d+) matches any digit one or more times, so this matches "123" and populates the "Group 2" variable. The *Verbose* modifier ignores the spaces after (\d+) and ignores the comments at the end of the regular expression.

# Using Regular Expressions

Once a regular expression has been built and debugged, you can add it to your code by copying and pasting the regular expression into the Komodo Editor Pane. Each language is a little different in the way it incorporates regular expressions. The following are examples of regular expressions used in Perl, Python, PHP and Tcl.

## Perl

This Perl code uses a regular expression to match two different spellings of the same word. In this case the program prints all instances of "color" and "colour".

```
while($word = <STDIN>){
    print "$word" if ($word =~ /colou?r/i );
}
```

The metacharacter "?" specifies that the preceding character, "u", occurs zero or one times. The modifier "i" (ignore case) that follows /colou?r/ means that the regular expression will match $word, regardless of whether the specified characters are uppercase or lowercase (for example, Color, COLOR and CoLour will all match).

## Python

This Python code uses a regular expression to match a pattern in a string. In Python, regular expressions are available via the re module.

```
import re
m = re.search("Java[Ss]cript", "in the JavaScript tutorial")
if m:
    print "matches:", m.group()
else:
    print "Doesn't match."
```

The re.search() function returns a match object if the regular expression matches; otherwise, it returns none. The character class "[Ss]" is used to find the word "JavaScript", regardless of whether the "s" is capitalized. If there is a match, the script uses the group() method to return the matching strings. Otherwise the program prints "Doesn't Match".

## Tcl

This Tcl code uses a regular expression to match all lines in a document that contain a URL.

```tcl
while {[gets $doc line]!=-1} {
   if {regexp -nocase {www\..*\.com} $line} {
       puts $line
```

This `while` loop searches every line in a file for any instance of a URL and displays the results. Tcl implements regular expressions using the `regexp` and `regsub` commands. In the example shown above, the `regexp` is followed by the `-nocase` option, which specifies that the following regular expression should match, regardless of case. The regular expression attempts to match all web addresses. Notice the use of backslashes to include the literal dots (.) that follow "www" and precede "com".

## PHP

This PHP code uses a Perl Compatible Regular Expressions(PCRE) to search for valid phone numbers in the United States and Canada; that is, numbers with a three−digit area code, followed by an additional seven digits.

```php
$numbers = array("777-555-4444",
                 "800-123-4567",
                 "(999)555-1111",
                 "604.555.1212",
                 "555-1212",
                 "This is not a number",
                 "1234-123-12345",
                 "123-123-1234a",
                 "abc-123-1234");

function isValidPhoneNumber($number) {
    return preg_match("/\(?\d{3}\)?[-\s.]?\d{3}[-\s.]\d{4}$/x", $number);
}

foreach ($numbers as $number) {
    if (isValidPhoneNumber($number)) {
        echo "The number '$number' is valid\n";
    } else {
        echo "The number '$number' is not valid\n";
    }
}
```

This PHP example uses the `preg_match` function for matching regular expressions. Other Perl compatible regular expression functions are also available. If the function `isValidPhone` returns true, the program outputs a statement that includes the valid phone number. Otherwise, it outputs a statement advising that the number is not valid.

# Regular Expressions Primer

The Regular Expressions Primer is a tutorial for those completely new to regular expressions. To familiarize you with regular expressions, this primer starts with the simple building blocks of the syntax and through examples, builds to construct expressions useful for solving real every–day problems. The primer later discusses how to search for and replace text with regular expression syntax.

While the examples presented in the primer are generic in structure and syntax, there are minor usage differences amongst the Komodo supported languages (Python, Perl, Tcl, etc). These differences are relevant when "Python Regular Expressions" are specified for use in Komodo's Find dialog box and Open/Find Toolbar. See More Regex Resources for information on Python regular expressions.

Regular expressions are embedded in programs to parse text. For example, a Python program might contain a regular expression as follows:

```
import re
n = re.compile(r'\bw[a-z]*', re.IGNORECASE)
print n.findall('will match all words beginning with the letter w.')
```

An advanced Python regular expression embedded in a program:

```
# Generate statement parsing regexes.
stmts = ['#\s*(?P<op>if|elif|ifdef|ifndef)\s+(?P<expr>.*?)',
         '#\s*(?P<op>else|endif)',
         '#\s*(?P<op>error)\s+(?P<error>.*?)',
         '#\s*(?P<op>define)\s+(?P<var>[^\s]*?)(\s+(?P<val>.+?))?',
         '#\s*(?P<op>undef)\s+(?P<var>[^\s]*?)']
patterns = ['^\s*%s\s*%s\s*%s\s*$'
              % (re.escape(cg[0]), stmt, re.escape(cg[1]))
              for cg in cgs for stmt in stmts]
stmtRes = [re.compile(p) for p in patterns]
```

In this example, regular expressions are used within various statements. See the Python Tutorial for the full program where this regular expression is used.

Komodo's Rx Toolkit is used to build and evaluate regular expressions. See Using Rx Toolkit for more information.

## About Regular Expressions

Regular expressions are used to describe patterns of characters that match against text strings. They can be used as a tool to search for and replace text, manipulate data, or test for a certain condition in a string of characters. Many everyday tasks can be accomplished with regular expressions, such as checking for the occurrence of a specific word or phrase in the body of an e–mail message, or finding specific file

types, such as `.txt` files, in a folder or directory. Regular expressions are often called "regex", "regexes", "regexps", and "RE". This primer uses the terms "regular expressions", "regex", and "regexes" equally.

# About Regex Syntax

Regular expressions use syntax elements comprised of alphanumeric characters and symbols. For example, the regex `(2)` searches for the number 2, while the regex `([1-9][0-9]{2}-[0-9]{4})` matches a regular 7–digit phone number.

There are many flavors and types of regular expression syntax. These variations are found in various tools, languages and operating systems. For example, Perl, Python, grep, sed, VI, and Unix all use variations on standard regex syntax. This primer focuses on standard regex patterns not tied to a specific language or tool. This standard syntax can be later applied to the specific language, tool or application of your choice.

# Building Simple Patterns

Complete regular expressions are constructed using characters as small building block units. Each building block is in itself simple, but since these units can be combined in an infinite number of ways, knowing how to combine them to achieve a goal takes some practice. This section shows you how to build regexes through examples ranging from the simple to the more complex.

## Matching Simple Strings

The simplest and most common type of regex is an alphanumeric string that matches itself, called a "literal text match". A literal text regex matches anywhere along a string. For example, a literal string matches itself when placed alone, and at the beginning, middle, or end of a larger string. Literal text matches are case sensitive.

***Using regexes to search for simple strings.***

***Example 1***: Search for the string "at".

- ***Regex***:

        at

- ***Matches***:

```
at
math
hat
ate
```

- *Doesn't Match*:

```
it
a-t
At
```

*Example 2*: Search for the string "email".

- *Regex*:

```
email
```

- *Matches*:

```
email
emailing
many_emails
```

- *Doesn't Match*:

```
Email
EMAILing
e-mails
```

*Example 3*: Search for the alphanumeric string "abcdE567".

- *Regex*:

```
abcdE567
```

- *Matches*:

```
abcdE567
AabcdE567ing
text_abcdE567
```

- *Doesn't Match*:

```
SPAMabCdE567
ABCDe567
```

*Note:* Regular expressions are case sensitive unless case is deliberately modified.

# Searching with Wildcards

In the previous examples, regular expressions are constructed with literal characters that match themselves. There are other characters in regex syntax that match in a more generalized way. These are called "metacharacters". Metacharacters do not match themselves, but rather perform a specific task when used in a regular expression. One such metacharacter is the dot ".", or wildcard. When used in a regular expression, the wildcard can match any single character.

***Using the wildcard to match any character.***

*Example 1*: Use a wildcard to search for any one character before the string "ubject:".

- ***Regex***:

    ```
    .ubject:
    ```

- ***Matches***:

    ```
    Subject:
    subject:
    Fubject:
    ```

- ***Doesn't Match***:

    ```
    Subject
    subject
    ```

*Example 2*: Use three dots "`...`" to search for any three characters within a string.

- ***Regex***:

    ```
    t...s
    ```

- ***Matches***:

    ```
    trees
    tEENs
    t345s
    t-4-s
    ```

- ***Doesn't Match***:

    ```
    Trees
    twentys
    t1234s
    ```

*Example 3*: Use several wildcards to match characters throughout a string.

- *Regex*:

    ```
    .a.a.a
    ```

- *Matches*:

    ```
    Canada
    alabama
    banana
    3a4a5a
    ```

- *Doesn't Match*:

    ```
    aaa
    ```

## Searching for Special Characters

In regular expression syntax, most non−alphanumerical characters are treated as special characters. These characters, called "metacharacters", include asterisks, question marks, dots, slashes, etc. In order to search for a metacharacter without using its special attribute, precede it with a backslash "\" to change it into a literal character. For example, to build a regex to search for a `.txt` file, precede the dot with a backslash `\.txt` to prevent the dot's special function, a wildcard search. The backslash, called an "escape character" in regex terminology, turns metacharacters into literal characters.

Precede the following metacharacters with a backslash "\" to search for them as literal characters:

```
^ $ + * ? . | ( ) { } [ ] \
```

*Using the backslash "\" to escape special characters in a regular expression.*

*Example 1*: Escape the dollar sign "$" to find the alphanumeric string "$100".

- *Regex*:

    ```
    \$100
    ```

- *Matches*:

    ```
    $100
    $1000
    ```

- *Doesn't Match*:

    ```
    \$100
    100
    ```

*Example 2*: Use the dot "." as a literal character to find a file called "email.txt".

- *Regex*:

      email\.txt

- *Matches*:

      email.txt

- *Doesn't Match*:

      email
      txt
      email_txt


*Example 3*: Escape the backslash "\" character to search for a Windows file.

- *Regex*:

      c:\\readme\.txt

- *Matches*:

      c:\readme.txt

- *Doesn't Match*:

      c:\\readme.txt
      d:\readme.txt
      c:/readme.txt


# Ranges and Repetition

Regex syntax includes metacharacters which specify the number of times a particular character or string must match. This group of metacharacters is called "quantifiers"; they influence the quantity of matches found. Quantifiers act on the element immediately preceding them, which could be a digit, a letter, or another metacharacter (including spaces as metacharacters not previously defined and the dot "."). This section demonstrates how quantifiers search using ranges and repetition.

## Ranges, {min, max}

Ranges are considered a "counting qualifier" in regular expressions. This is because they specify the minimum number of matches to find and the maximum number of matches to allow. Use ranges in regex searches when a bound, or a limit, should be placed on search results. For example, the range {3,5} matches an item at least 3 times, but not more than 5 times. When this range is combined with the regex, a{3,5}, the strings "aaa", "aaaa", and "aaaaa" are successfully matched. If only a single number is expressed within curly braces {3}, the pattern matches exactly three items. For example, the regex b{3} matches the string "bbb".

*Using ranges to identify search patterns.*

*Example 1*: Match the preceding "0" at least 3 times with a maximum of 5 times.

- *Regex*:

    ```
    60{3,5} years
    ```

- *Matches*:

    ```
    6000 years
    60000 years
    600000 years
    ```

- *Doesn't Match*:

    ```
    60 years
    600 years
    6003 years
    6000000 years
    ```

*Example 2*: Using the "." wildcard to match any character sequence two or three characters long.

- *Regex*:

    ```
    .{2,3}
    ```

- *Matches*:

    ```
    404
    44
    com
    w3
    ```

- *Doesn't Match*:

    ```
    4
    a
    aaaa
    ```

*Example 3*: Match the preceding "e" exactly twice.

- *Regex*:

      be{2}t

- *Matches*:

      beet

- *Doesn't Match*:

      bet
      beat
      eee

*Example 4*: Match the preceding "w" exactly three times.

- *Regex*:

      w{3}\.mydomain\.com

- *Matches*:

      www.mydomain.com

- *Doesn't Match*:

      web.mydomain.com
      w3.mydomain.com

## Repetition, ?*+

Unlike range quantifiers, the repetition quantifiers (question mark "?", asterisk "*", and plus "+") have few limits when performing regex searches. This is significant because these quantifiers settle for the minimum number of required matches, but always attempt to match as many times as possible, up to the maximum allowed. For example, the question mark "?" matches any preceding character 0 or 1 times, the asterisk "*" matches the preceding character 0 or more times, and the plus "+" matches the preceding character 1 or more times. Use repetition quantifiers in regex searches when large numbers of results are desired.

***Using repetition to search for repeated characters with few limits.***

*Example 1*: Use "?" to match the "u" character 0 or 1 times.

- *Regex*:

      colou?r

- *Matches*:

      colour
      color

- *Doesn't Match*:

      colouur
      Colour


*Example 2*: Use "*" to match the preceding item 0 or more times; use "." to match any character.

- *Regex*:

      www\.my.*\.com

- *Matches*:

      www.mysite.com
      www.mypage.com
      www.my.com

- *Doesn't Match*:

      www.oursite.com
      mypage.com


*Example 3*: Use "+" to match the preceding "5" at least once.

- *Regex*:

      bob5+@foo\.com

- *Matches*:

      bob5@foo.com
      bob5555@foo.com

- *Doesn't Match*:

      bob@foo.com
      bob65555@foo.com

**Quantifier Summary**

The following table defines the various regex quantifiers. Note that each quantifier is unique and will perform a varying minimum and maximum number of matches in order to search successfully.

| *Quantifier* | *Description* |
|---|---|
| {*num*} | Matches the preceding element *num* times. |
| {*min*, *max*} | Matches the preceding element at least *min* times, but not more than *max* times. |
| ? | Matches any preceding element 0 or 1 times. |
| * | Matches the preceding element 0 or more times. |
| + | Matches the preceding element 1 or more times. |

# Using Conditional Expressions

Conditional expressions help qualify and restrict regex searches, increasing the probability of a desirable match. The vertical bar "|" symbol, meaning "OR", places a condition on the regex to search for either one character in a string or another. Because the regex has a list of alternate choices to evaluate, this regex technique is called "alternation". To search for either one character or another, insert a vertical bar "|" between the desired characters.

*Example 1*: Use "|" to alternate a search for various spellings of a string.

- *Regex*:

    ```
    gray|grey
    ```

- *Matches*:

    ```
    gray
    grey
    ```

- *Doesn't Match*:

    ```
    GREY
    Gray
    ```

*Example 2*: Use "|" to alternate a search for either email or Email or EMAIL or e−mail.

- *Regex*:

  ```
  email|Email|EMAIL|e-mail
  ```

- *Matches*:

  ```
  email
  Email
  EMAIL
  e-mail
  ```

- *Doesn't Match*:

  ```
  EmAiL
  E-Mail
  ```

## Grouping Similar Items in Parentheses

Use parentheses to enclose a group of related search elements. Parentheses limit scope on alternation and create substrings to enhance searches with metacharacters. For example, use parentheses to group the expression `(abc)`, then apply the range quantifier `{3}` to find instances of the string "abcabcabc".

*Using parentheses to group regular expressions.*

*Example 1*: Use parentheses and a range quantifier to find instances of the string "abcabcabc".

- *Regex*:

  ```
  (abc){3}
  ```

- *Matches*:

  ```
  abcabcabc
  abcabcabcabc
  ```

- *Doesn't Match*:

  ```
  abc
  abcabc
  ```

*Example 2*: Use parentheses to limit the scope of alternative matches on the words gray and grey.

- *Regex*:

    ```
    gr(a|e)y
    ```

- *Matches*:

    ```
    gray
    grey
    ```

- *Doesn't Match*:

    ```
    gry
    graey
    ```

*Example 3*: Use parentheses and "|" to locate past correspondence in a mail–filtering program. This regex finds a 'To:' or a 'From:' line followed by a space and then either the word 'Smith' or the word 'Chan'.

- *Regex*:

    ```
    (To:|From:)(Smith|Chan)
    ```

- *Matches*:

    ```
    To:Smith
    To:Chan
    From:Smith
    To:Smith, Chan
    To:Smithe
    From:Channel4News
    ```

- *Doesn't Match*:

    ```
    To:smith
    To:All
    To:Schmidt
    ```

## Matching Sequences

You can build a regular expression to match a sequence of characters. These sequences, called "character classes", simply place a set of characters side–by–side within square brackets "[]". An item in a character class can be either an ordinary character, representing itself, or a metacharacter, performing a special function. This primer covers how to build simple character classes, prevent matches with character classes, and construct compound character classes with metacharacters.

**Building Simple Character Classes**

The most basic type of character class is a set of characters placed side–by–side within square brackets "[]". For example, the regular expression `[bcr]at`, matches the words "bat", "cat", or "rat" because it uses a character class (that includes "b","c", or "r") as its first character. Character classes only match singular characters unless a quantifier is placed after the closing bracket. For examples using quantifiers with character classes, see Compound Character Classes. The following table shows how to use simple character classes in regex searches.

*Note:* When placed inside a character class, the hyphen "–" metacharacter denotes a continuous sequence of letters or numbers in a range. For example, [a–d] is a range of letters denoting the continuous sequence of a,b,c and d. When a hyphen is otherwise used in a regex, it matches a literal hyphen.

***Using simple character classes to perform regex searches.***

*Example 1*: Use a character class to match all cases of the letter "s".

- *Regex*:

    ```
    Java[Ss]cript
    ```

- *Matches*:

    ```
    JavaScript
    Javascript
    ```

- *Doesn't Match*:

    ```
    javascript
    javaScript
    ```


*Example 2*: Use a character class to limit the scope of alternative matches on the words gray and grey.

- *Regex*:

    ```
    gr[ae]y
    ```

- *Matches*:

    ```
    gray
    grey
    ```

- *Doesn't Match*:

    ```
    gry
    graey
    ```


*Example 3*: Use a character class to match any one digit in the list.

- *Regex*:

  ```
  [0123456789]
  ```

- *Matches*:

  ```
  5
  0
  9
  ```

- *Doesn't Match*:

  ```
  x
  ?
  F
  ```

*Example 4*: To simplify the previous example, use a hyphen "–" within a character class to denote a range for matching any one digit in the list.

- *Regex*:

  ```
  [0-9]
  ```

- *Matches*:

  ```
  5
  0
  9
  ```

- *Doesn't Match*:

  ```
  234
  42
  ```

*Example 5*: Use a hyphen "–" within a character class to denote an alphabetic range for matching various words ending in "mail".

- *Regex*:

  ```
  [A-Z]mail
  ```

- *Matches*:

  ```
  Email
  Xmail
  Zmail
  ```

- *Doesn't Match*:

  ```
  email
  ```

```
mail
```

*Example 6*: Match any three or more digits listed in the character class.

- *Regex*:

  ```
  [0-9]{3,}
  ```

- *Matches*:

  ```
  012
  1234
  555
  98754378623
  ```

- *Doesn't Match*:

  ```
  10
  7
  ```

## Preventing Matches with Character Classes

Previous examples used character classes to specify exact sequences to match. Character classes can also be used to prevent, or negate, matches with undesirable strings. To prevent a match, use a leading caret "^" (meaning NOT), within square brackets, [^...]. For example, the regex [^a] matches any single character except the letter "a".

*Note:* The caret symbol must be the first character within the square brackets to negate a character class.

***Using character classes to prevent a sequence from matching.***

*Example 1*: Prevent a match on any numeric string. Use the "*" to match an item 0 or more times.

- *Regex*:

  ```
  [^0-9]*
  ```

- *Matches*:

  ```
  abc
  c
  Mail
  u-see
  a4a
  ```

- *Doesn't Match*:

```
1
42
100
23000000
```

*Example 2*: Search for a text file beginning with any character not a lower−case letter.

- *Regex*:

  ```
  [^a-z]\.txt
  ```

- *Matches*:

  ```
  A.txt
  4.txt
  Z.txt
  ```

- *Doesn't Match*:

  ```
  r.txt
  a.txt
  Aa.txt
  ```

*Example 3*: Prevent a match on the numbers "10" and "12".

- *Regex*:

  ```
  1[^02]
  ```

- *Matches*:

  ```
  13
  11
  19
  17
  1a
  ```

- *Doesn't Match*:

  ```
  10
  12
  42
  a1
  ```

**Compound Character Classes**

Character classes are a versatile tool when combined with various pieces of the regex syntax. Compound character classes can help clarify and define sophisticated searches, test for certain conditions in a program, and filter wanted e−mail from spam. This section uses compound character classes to build meaningful expressions with the regex syntax.

*Using compound character classes with the regex syntax.*

*Example 1*: Find a partial e−mail address. Use a character class to denote a match for any number between 0 and 9. Use a range to restrict the number of times a digit matches.

- *Regex*:

```
smith[0-9]{2}@
```

- *Matches*:

```
smith44@
smith42@
```

- *Doesn't Match*:

```
Smith34
smith6
Smith0a
```

*Example 2*: Search an HTML file to find each instance of a header tag. Allow matches on whitespace after the tag but before the ">".

- *Regex*:

```
(<[Hh][1-6] *>)
```

- *Matches*:

```
<H1>
<h6>
<H3  >
<h2    >
```

- *Doesn't Match*:

```
<H1
<   h2>
<a1>
```

*Example 3*: Match a regular 7−digit phone number. Prevent the digit "0" from leading the string.

- *Regex*:

```
([1-9][0-9]{2}-[0-9]{4})
```

- *Matches*:

```
555-5555
123-4567
```

- *Doesn't Match*:

```
555.5555
1234-567
023-1234
```

*Example 4*: Match a valid web–based protocol. Escape the two front slashes.

- *Regex*:

```
[a-z]+:\/\/
```

- *Matches*:

```
http://
ftp://
tcl://
https://
```

- *Doesn't Match*:

```
http
http:
1a3://
```

*Example 5*: Match a valid e–mail address.

- *Regex*:

```
[a-z0-9_-]+(\.[a-z0-9_-]+)*@[a-z0-9_-]+(\.[a-z0-9_-]+)+
```

- *Matches*:

```
j_smith@foo.com
j.smith@bc.canada.ca
smith99@foo.co.uk
1234@mydomain.net
```

- *Doesn't Match*:

```
@foo.com
```

```
        .smith@foo.net
        smith.@foo.org
        www.myemail.com
```


**Character Class Summary**

The following table defines various character class sequences. Use these alphanumeric patterns to simplify your regex searches.

| *Character Class* | *Description* |
| --- | --- |
| `[0-9]` | Matches any digit from 0 to 9. |
| `[a-zA-z]` | Matches any alphabetic character. |
| `[a-zA-z0-9]` | Matches any alphanumeric character. |
| `[^0-9]` | Matches any non–digit. |
| `[^a-zA-z]` | Matches any non–alphabetic character. |


# Matching Locations within a String

At times, the pattern to be matched appears at either the very beginning or end of a string. In these cases, use a caret "^" to match a desired pattern at the beginning of a string, and a dollar sign "$" for the end of the string. For example, the regular expression `email` matches anywhere along the following strings: "email", "emailing", "bogus_emails", and "smithsemailaddress". However, the regex `^email` only matches the strings "email" and "emailing". The caret "^" in this example is used to effectively *anchor* the match to the start of the string. For this reason, both the caret "^" and dollar sign "$" are referred to as anchors in the regex syntax.

*Note:* The caret "^" has many meanings in regular expressions. Its function is determined by its context. The caret can be used as an anchor to match patterns at the beginning of a string, for example: `(^File)`. The caret can also be used as a logical "NOT" to negate content in a character class, for example: `[^...]`.

*Using anchors to match at the beginning or end of a string.*

*Example 1*: Use "$" to match the ".com" pattern at the end of a string.

- *Regex*:

    ```
    .*\.com$
    ```

- *Matches*:

    ```
    mydomain.com
    a.b.c.com
    ```

- *Doesn't Match*:

    ```
    mydomain.org
    mydomain.com.org
    ```

*Example 2*: Use "^" to match "inter" at the beginning of a string, "$" to match "ion" at the end of a string, and ".*" to match any number of characters within the string.

- *Regex*:

    ```
    ^inter.*ion$
    ```

- *Matches*:

    ```
    internationalization
    internalization
    ```

- *Doesn't Match*:

    ```
    reinternationalization
    ```

*Example 3*: Use "^" inside parentheses to match "To" and "From" at the beginning of the string.

- *Regex*:

    ```
    (^To:|^From:)(Smith|Chan)
    ```

- *Matches*:

    ```
    From:Chan
    To:Smith
    From:Smith
    To:Chan
    ```

- *Doesn't Match*:

```
From: Chan
from:Smith
To Chan
```

*Example 4*: Performing the same search as #3, place the caret "^" outside the parentheses this time for similar results.

- *Regex*:

    ```
    ^(From|Subject|Date):(Smith|Chan|Today)
    ```

- *Matches*:

    ```
    From:Smith
    Subject:Chan
    Date:Today
    ```

- *Doesn't Match*:

    ```
    X-Subject:
    date:Today
    ```

# Searching and Replacing

Regular expressions are often used to search and replace text strings. Up until this point, the preceding examples have centered on matching a string using regex syntax. This section examines the search and replace operation as a prominent feature of regular expressions and solves standard problems using the substitution syntax.

Like with building regular expressions, there are many variations on substitution syntax depending on the language used. This primer focuses on general search and replace syntax. This standard syntax can be later applied to the specific language, tool or application of your choice.

## Building Simple Substitution Searches

Substitution searches search for and replace a pattern of text. Substitutions are performed using the `s///` operator, "s" standing for substitution. The `s///` operator takes a regular expression between the first and second front slashes, while the second and third front slashes take the replacement text.

For example:

```
s/<regex>/<substitution-string>/
```

*Use the s/// operator to search for and replace a simple text string. Note:* these searches only replace the first instance of the string found.

*Example 1*: Search for the string "email" and replace it with "e–mail".

- *Regex Substitution*:

      s/email/e-mail

- *Search for*:

      email

- *Replace with*:

      e-mail


*Example 2*: Search for an old domain name and replace it with the new domain name. Using regex syntax, escape "." and "/" characters.

- *Regex Substitution*:

      s/http:\/\/www\.old-domain\.com/http:\/\/www\/new-domain\.com/

- *Search for*:

      www.old-domain.com

- *Replace with*:

      www.new-domain.com


*Example 3*: Search for a single string starting with any lowercase letter and ending with "mail". Replace the string with "Email".

- *Regex Substitution*:

      s/[a-z]mail/Email

- *Search for*:

      email
      zmail
      xmail

- *Replace with*:

```
        Email
```

## Modifying Substitution Searches

The previous substitution examples focused on small searches, such as replacing a single lower–case word in a single line of text. Extend the scope and flexibility of substitution searches through the use of modifiers. The modifier parameter is appended to the end of the `s///` operator as follows:

```
s/<regex>/<substitution-string>/<modifier>
```

Use the modifier "i" to ignore case in alphabetic searches, "m" to allow multiple lines in a string, "s" to treat a pattern as a single line, "x" to allow for whitespace and comments, and "g" for global searching all occurances of the pattern in a file and not just the first instance found.

*Use various modifiers with the `s///` operator to search for and replace text strings.*

*Example 1*: Using the "g" modifier, search globally through all .htm instances in a file and replace them with ".html". Using "$", only substitute the ".htm" string when it appears at the end of a line. An example file where this substitution succeeds:

```
/manual/mod_python/pythonapi.htm
/manual/mod_python/more-comp.htm
/manual/mod_python/overview.htm
```

- *Regex Substitution*:

```
s/\.htm$/\.html/g
```

- *Search for*:

```
.htm
```

- *Replace with*:

```
.html
```

*Example 2*: Using the "g" modifier, remove all html tags in a file and replace the tags with an empty string.

- *Regex Substitution*:

```
s/<[^>]+>//g
```

- *Search for*:

```
<code>Tag</code>
```

- *Replace with*:

```
Tag
```

*Example 3*: Perform a case insensitive search for various instances of "login" and replace with the string "password".

- *Regex Substitution*:

```
s/LOGIN/password/i
```

- *Search for*:

```
LOGIN
login
LoGiN
Login
```

- *Replace with*:

```
password
```

## Substitution Modifier Summary

The following table defines various modifiers for the substitution operator. Modifiers change how a match is performed. Use these modifiers to expand the scope and versatility of your substitutions.

| Modifier | Meaning |
|---|---|
| i | Ignore case when matching exact strings. |
| m | Treat string as multiple lines. Allow "^" and "$" to match next to newline characters. |
| s | Treat string as single line. Allow "." to match a newline character. |
| x | Ignore whitespace and newline characters in the regular expression. Allow |

comments.

o      Compile regular expression once only.

g      Match all instances of the pattern in the target string.

# More Regex Resources

## Internet Web Sites:

*Beginner*:

- [Python Library Reference: Regular Expression Operations](), ActiveState Programmer Network (ASPN)
- [Python Regular Expressions](), *Dive into Python*, Mark Pilgrim
- [Python Cookbook](), ActiveState Programmer Network (ASPN)
- [Five Habits for Successful Regular Expressions](), The O'Reilly ONLamp Resource Center
- [Beginner's Introduction to Perl – Part 3](), The O'Reilly Perl Resource Center

*Intermediate*:

- [Rx Cookbook](), ActiveState Programmer Network (ASPN)
- [Regexp Power](), The O'Reilly Perl Resource Center

*Advanced*:

- [Power Regexps, Part II](), The O'Reilly Perl Resource Center

# Komodo and the Perl Dev Kit

Komodo provides support for ActiveState's Perl Dev Kit, so that you can build executable programs, ActiveX controls and Windows services in Perl.

After creating the desired Perl script in Komodo, select *Tools/Build Standalone Perl Application* to configure the Perl application. The *Build Standalone Perl Application* dialog box will open, giving you access to key Perl Dev Kit tools from within Komodo.

- *PerlApp* – Build an executable file from Perl scripts.
- *PerlCtrl* – Build Active X controls from Perl scripts.
- *PerlNET* – Create Perl components and applications that are compliant with Microsoft's .NET Framework.
- *PerlSvc* – Convert Perl programs to Windows services.
- *PerlTray* – Write system tray applications in Perl.

*Note:* On Linux, only the PerlApp tool is supported.

For complete instructions on building executables, controls and services in Perl, see the User Guide that accompanies the Perl Dev Kit.

As you configure options on the tabs described in the sections below, the corresponding command line string is displayed at the bottom of the *Build Standalone Perl Application* dialog box. Command line options for PerlApp, PerlCtrl, PerlSvc, PerlNET and PerlTray can be found in the User Guide that accompanies the Perl Dev Kit. Alternatively, view the Perl Dev Kit User Guide on ASPN, the ActiveState Programmer Network.

When using the PDK 'Build standalone application' feature in Komodo with Perl 5.8.0 on a Linux installation where the environment is set to use UTF–8, you must add a module 'utf8' on the modules tab. This is the equivalent of 'perlapp −−add utf8'. This does not affect Perl 5.6.x or future versions of Perl 5.8.1 or higher.

Once you have configured options using the tabs in the *Build Standalone Perl Application* dialog box, use the buttons at the bottom of the dialog box to create a build, add a script to the Toolbox, or debug a script in Komodo.

- *Add to Toolbox* – Once you have created a new script, you can click this button to add it to the Toolbox as a run command.
- *Build* – Click this button to create a new build or overwrite an existing build.
- *Debug* – If the Komodo debugging option is selected on the General tab, you can start the debugger by clicking this button.

*Feature Showcase*

- build a Perl executable

# Configuring the General Tab

The build options for the Perl Dev Kit correspond with the tools described in the Perl Dev Kit documentation, which contains detailed instructions on configuring Perl executables, services and controls.

- *Enter the name of the script to build using the PDK* – Use this field to enter the path and file name of the source Perl script. This option is equivalent to the `-script` command line argument.
- *Build the script using* – Select the type of output you wish to generate.
- *Enter the name of the target executable or control* – Use this field to specify the path and name of the output file. This option is equivalent to the `-exe` command line argument.
- *Dependencies*
  - *None* – Select this option to include all necessary files in the output file, so that it can be run on systems that do not have Perl56.dll or ActivePerl. This option is equivalent to the `-freestanding` command line argument.
  - *Perl Dll required on target* – Select this option to reduce the size of the generated executable by excluding Perl56.dll from the output file. Target systems must have the Perl56.dll installed. This setting corresponds with the `-xclude` command line argument.
  - *ActivePerl required on target* – Select this option to create an output file that will be run on systems where ActivePerl and any modules included via `use` and `require` statements are installed. This option is equivalent to the `-dependent` command line argument.
- *Verbose build information* – This option will generate detailed output messages while the output file is being built. This option corresponds to the `-verbose` command line argument.
- *Hide console (for GUI applications)* – Similar to running wperl.exe, this option is useful for building applications that run in the background. This setting corresponds with the PerlApp `-gui` command line argument, and is only available when you select the PerlApp tool.
- *Overwrite existing build* – Select this check box if you want the new build to replace the existing build. If you attempt to overwrite a build without selecting this option, a pop–up dialog box will warn that the .exe file already exists. You can then choose to overwrite the file, overwrite the file and enable the check box, or cancel the command. This option is equivalent to the `-force` command line argument.
- *Delete temp files after each run* – Freestanding Perl applications, services and controls sometimes contain embedded DLLs that are extracted and cached in the host system's temporary directory. Check this box to delete these files after each run. This setting corresponds with the `-clean` command line argument.
- *Debugging* – To debug the Perl executable, control or service as it is being built, select the desired debugger from the drop–down list. If you are not using either the Komodo or the PDK debugger, you can specify a *Hostname* and *Port* for another debugger in the fields provided.

# Configuring the Modules Tab

The Modules tab is used for adding external modules to the build, as well as trimming unwanted modules.

## Specifying Extra Modules For Your Script

To add a module to the output program, enter the name of the module in the *Module name* field and click *Add*. The new module to be added will be displayed in the list box above. Remove modules from the list box using the *Delete* and *Delete All* buttons.

This option corresponds with the `-add` command line argument.

## Specifying Modules to Trim from the Package

To remove an unwanted module from the build, enter the name of the module in the *Modules* field and click *Add*. The new module to be trimmed will be displayed in the list box above. Remove modules from the list box using the *Delete* and *Delete All* buttons.

This option corresponds with the `-trim` command line argument.


# Configuring the Files Tab

The Files tab is used to add additional files (typically data files used by the embedded program) to the output file that will be extracted when the program is run.

This option corresponds with the `-bind` command line argument.

## Adding Files

To add a file to the output program, click *Add*. In the pop−up dialog box, enter the source location of the file on your system, and the location where the file should be extracted when the output file is run.

## Editing Files

To edit a file that has been added to the output program, click *Edit*. In the dialog box, as required, alter the source location of the file on your system, and the location where the file should be extracted when the output file is run.

## Deleting Files

To remove a file that was to be added to the output program, click the file, then click *Delete*.

# Configuring the Version Tab

The Version tab is used to embed version information in the output program. It corresponds to the `-info` command line argument.

To alter any of the version options, select the desired option in the **Version field** column and enter the desired value in the field below. This information will be assembled as a version information (VERINFO) resource, and will be displayed to users when they view the properties for your script in Windows Explorer.

# Configuring the Library Paths Tab

You can use the Library Paths tab to add directories to your build. The options on the Library Path tab correspond with the command line arguments `-lib` and `-blib`.

## Specifying "lib" and "blib" Directories to Include

To add a lib or blib directory to include in your output file, Click **Add**. From the **Browse for Folder** dialog box, select the directory path to include and click **OK**. The path can contain multiple directories that are separated in the same way as in the PATH environment variable.

Use the **Delete** and **Delete All** to remove directories that you do not want to add from the "lib" and "blib" list boxes.

# Configuring the Extra Tab

The Extra tab is for adding icon files, as well as manually specifying any additional command line arguments.

## Specifying Icon files

To include .ico files in a build, Click **Add**. From the **Add Icon** dialog box, select the icon(s) you want to add and click **Open**. The complete path for the icon file will be displayed in the **Icon File** list box.

This option is equivalent to the `-icon` command line argument.

## Specifying Additional Command Line Parameters

If you want to specify any command line parameters in addition to those selected using the options in the **Build Standalone Perl Application** dialog box, you can type them in the field provided.

# Visual Package Manager (Komodo Pro)

The Visual Package Manager (VPM) is a graphical interface for the Perl Package Manager (PPM), which is included with ActiveState's [ActivePerl](#) distribution. VPM is used to install, upgrade and remove Perl modules from your ActivePerl installation.

The VPM requires ActivePerl version 631 (which includes PPM version 3). For more information about PPM, see the PPM documentation included with your ActivePerl installation or on the [ASPN](#) Web site.

- ***Repositories***: the location of the collection of Perl modules. This can be a website or a CD. Repositories are added, ordered or removed on the [Configure](#) tab. Note that not all Perl packages have PPM equivalents for every platform and Perl version. To check the [build status](#) of a package, see ActiveState's ASPN website.
- ***Local Perl Installation Structure***: the structure of the Perl installation on your local system. More than one version of Perl may be installed. During installation, you can specify the target for the package installation.

*Tutorials*

- [Perl Tutorial](#)

To launch the VPM, select ***Tools/Visual Package Manager***, or click the VPM button on the [Toolbar](#).

## Installing New Modules

Use the ***Install*** page to search for Perl modules located on the selected Repository. See [Searching for Modules](#) for a description of search syntax.

After you click the ***Search*** button, modules that match the search criteria will be displayed in a list in the lower part of the screen. Note the buttons and keyboard shortcuts that can be used to quickly navigate the list of packages.

Select the modules you want to install by clicking the check box to the left of the module list. Ensure that the desired Perl installation is selected as the ***Target***. Click ***Install*** to download and install the selected modules.

### Searching for Modules

The following examples describe the syntax for entering search strings:

***Search for 'CGI' anywhere in the package name:***

```
CGI
```

Sample results:

```
Apache-CGI
CGI-Application
CGI-ArgChecker
```

*Search for 'CGI' at the beginning of the package name:*

```
CGI*
```

Sample results:

```
CGI-ArgChecker
CGI-Application
```

*Search for all modules authored by someone with 'smith' in their name or email address:*

```
AUTHOR=smith
```

Sample results:

```
Apache-ProxyPass
Business-ISBN
```

*Search for 'compress' anywhere in the package abstract:*

```
ABSTRACT=compress
```

Sample results:

```
apache-GzipChain
IO-Zlib
```

*Search for 'CGI' in the name, or 'web' in the abstract:*

```
CGI or ABSTRACT=web
```

Sample results:

```
CGI-XMLForm
HTML-Clean
```

*Search for 'XML' in the name and either 'parser' in the name or 'pars' in the abstract, but not with 'XPath' in the name:*

```
XML and (parser or ABSTRACT=pars) and not XPath
```

Sample results:

```
XML-Node
```

```
XML-Parser-EasyTree
```

***PPM Server 3.0 repositories only: search by module name, even if unrelated to the containing package:***

```
Data::Grove
```

Sample results:

```
libxml-perl
```

***Browse all packages in the repository:***

```
*
```

Sample results:

```
Affix-Infix2Postfix
AI-Fuzzy
[many more...]
```

## Upgrading Existing Modules

The **Upgrade** panel displays a list of all installed modules that have upgrades available on the selected repository. All modules are initially selected.

## Removing Installed Modules

The **Remove** page lists all installed modules. Modules can be selected for removal. If a module is a prerequisite for another module, the former can only be removed if the latter is selected as well.

Modules considered *precious* are not listed, as they are necessary for VPM (and PPM) to function correctly. It would not be possible to reinstall them (via VPM/PPM) once they are removed.

## Configuring the VPM

The *Configure* page is used to add package repositories, and to view, enable, disable or re-order existing repositories.

## Adding a Repository

- *Name*: enter a name by which this repository will be known.
- *Location*: enter the URL where the repository is located.
- *Username*: if the repository requires you to log in, enter your user name in this field.
- *Password*: if the repository requires you to log in, enter your password in this field.
- *Repository Order*: when you search for modules, repositories are searched in the order they appear on this page. Specify where the new repository should be added to the existing repository list.

After configuring repository information, click *Add* to add the repository to the list.

Repositories are displayed in the bottom section of the *Configure* page.

# Interpolation Shortcuts

Interpolation shortcuts are codes embedded in run commands, snippets or templates that, at "execution" time, get replaced with values. For example, the path and name of the current file can be inserted via an interpolation shortcut when a run command is executed.

Interpolation shortcuts embedded in run commands are inserted via the Run Command dialog box. When using interpolation shortcuts in snippets or templates, insert the interpolation code using bracketed syntax. Run commands can be stored in a project or the Toolbox for frequent use.

## Interpolation Code List

The following table contains all of the interpolation shortcut codes available in Komodo.

| Code | Description |
| --- | --- |
| **%** | a literal percent sign (%); for example, `Path = C:\temp; %%PATH%%` inserts the directory "temp" at the beginning of the PATH statement |
| **f** | the basename of the current file |
| **F** | the full path and name of the current file |
| **L** | the line where the editing cursor is located within the current file |
| **d** | the base directory of the current file |
| **D** | the entire directory path of the current file |
| **P** | the full path of the active project |
| **p** | the directory path of the active project |
| **w** | the word under the cursor in the editor |
| **W** | URL−escaped word under cursor; replaces characters that are not valid in a query string, such as spaces and ampersands |
| **s** | the current selection; interpolates the text that is currently selected in the editor |
| **S** | URL−escaped selection; replaces characters that are not valid in a query string, such as spaces and ampersands |
| **perl** | the perl interpreter specified in Komodo's Perl preference |
| **php** | the php interpreter specified in Komodo's PHP preference |
| **python** | the python interpreter specified in Komodo's Python preference |
| **tclsh** | the tclsh interpreter specified in Komodo's Tcl preference |

| wish | the wish interpreter specified in Komodo's Tcl preference |
| --- | --- |
| browser | the browser specified in Komodo's Web Browser preference |
| guid | a new GUID (Global Unique Identifier) |
| date * | the current date |
| ask * | ask the user for the value when invoked |
| path * | special Komodo directories |
| pref * | values from Komodo preferences |
| debugger * | runtime properties of the debugger system |

* Codes marked with asterisks have special options, *not* the options described in the Basic Interpolation Options section. The options for these codes are described below.


# Basic Interpolation Code Syntax

Interpolation code blocks come in two forms: bracketed and non–bracketed. Run commands use the non–bracketed format. Snippets and templates use the bracketed format.


## Non–Bracketed Syntax

The syntax for a non–bracketed interpolation code is:

```
%(<code><backref>:<options>...)
```

where <code> is one of the codes shown in the table above, <backref> is a number and <options>... depend on the specific code. Back–references and options are discussed in separate sections. The following are examples of non–bracketed interpolation code:

```
%(perl)
%w
%guid2
%(ask:Your Name:Trent Mick)
```

The parentheses are optional if the code block does not contain spaces. For example, the following two commands are equivalent:

```
%ask:Name:Trent
```

```
%(ask:Name:Trent)
```

## Bracketed Syntax

The syntax for a bracketed interpolation code is:

```
[[%(<code><backref>:<options>...)]]
```

where <code> is one of the codes shown in the table above, <backref> is a number and <options>...
depend on the specific code. Back–references and options are discussed in other sections. The following
are examples of bracketed syntax:

```
[[%perl]]
[[%w]]
[[%guid2]]
[[%ask:Your Name:Trent Mick]]
```

With bracketed interpolation codes, the parentheses are always optional. The double brackets enclose
spaces, making parentheses unnecessary. For example, both of the following commands are valid:

```
[[%ask:Your Name:Trent Mick]]
[[%(ask:Your Name:Trent Mick)]]
```

Bracketed interpolation code blocks permit some excess space immediately adjacent to the double
brackets. For example the following are equivalent:

```
[[%ask:Your Name:Trent Mick]]
[[ %ask:Your Name:Trent Mick]]
[[%(ask:Your Name:Trent Mick) ]]
[[ %(ask:Your Name:Trent Mick) ]]
```

# Basic Interpolation Options

The following table shows the standard options available for *most* interpolation codes. These options do
not apply to the codes displayed with an asterisk in the Interpolation Codes table.

| Option | Syntax | Description |
|--------|--------|-------------|
| orask | %(<code>:**orask**:<question>) | If a value for code cannot be determined automatically, then the user is prompted when the command is invoked. question is text that will be displayed when the user is asked to enter a value. |

| | | |
|---|---|---|
| else | %(<code>:**else**:<default>) | If a value for code cannot be determined automatically, then the default is used. |

# Date Code

A date interpolation code will be replaced with the current date, formatted according to a given optional format or the default format.

## Date Code Syntax

The syntax of the date code is as follows:

```
%(date<backref>:<optional-format>)
[[%(date:<optional-format>)]]
```

As noted in the Basic Interpolation Code Syntax section, the parentheses are optional. The <backref> optional parameter is discussed in the Back−references section. The following examples are valid:

```
%date
[[%date]]
%(date)
%date:%H:%M:%S
[[%date:%d/%m/%Y %H:%M:%S]]
```

## Date Code Format Option

If no <optional−format> is specified in a date code, the default date format is used. Configure the default date format using Komodo's Internationalization preferences.

If this format is not appropriate, you can specify the format with a string in accordance with the spec for the "format" argument of Python's *time.strftime()* method [1]_. The table is reproduced here for convenience.

| Directive | Meaning |
|---|---|
| **%a** | Locale's abbreviated weekday name. |
| **%A** | Locale's full weekday name. |
| **%b** | Locale's abbreviated month name. |

| %B | Locale's full month name. |
|---|---|
| %c | Locale's appropriate date and time representation. |
| %d | Day of the month as a decimal number [01,31]. |
| %H | Hour (24–hour clock) as a decimal number [00,23]. |
| %I | Hour (12–hour clock) as a decimal number [01,12]. |
| %j | Day of the year as a decimal number [001,366]. |
| %m | Month as a decimal number [01,12]. |
| %M | Minute as a decimal number [00,59]. |
| %p | Locale's equivalent of either AM or PM. |
| %S | Second as a decimal number [00,61]. |
| %U | Week number of the year (Sunday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Sunday are considered to be in week 0. |
| %w | Weekday as a decimal number [0(Sunday),6]. |
| %W | Week number of the year (Monday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Sunday are considered to be in week 0. |
| %x | Locale's appropriate date representation. |
| %X | Locale's appropriate time representation. |
| %y | Year without century as a decimal number [00,99]. |
| %Y | Year with century as a decimal number. |
| %Z | Time zone name (or by no characters if no time zone exists). |
| %% | A literal "%" character. |

For more information about Python 2.2's time access and conversions, visit
http://www.python.org/doc/2.2/lib/module–time.html

# Ask Code

An "ask" code will prompt the user for the value to use as its replacement. The user is prompted using a pop–up dialog box. If a code snippet or run command includes more than one "ask" code, then the pop–up dialog box will request values for all of these codes.

## Ask Code Syntax

The "ask" code prompts users to enter input in the Interpolation Query dialog box before running a command. The complete syntax for the %(ask) code is:

```
%(ask[:NAME:[DEFAULT]])
```

...where "NAME" is an optional name to insert in the Interpolation Query dialog box and "DEFAULT" is an optional default value that appears in the dialog box.

Use the `%(...:orask)` modifier with other interpolation codes to prompt for input if no value can be determined. The syntax for a modified shortcut is:

```
%(SHORTCUT:orask[:NAME])
```

...where "SHORTCUT" is the shortcut as displayed in the shortcut drop−down list and "NAME" is an optional name to insert in the Interpolation Query dialog box. See the Run Command Tutorial for examples of %(ask) and %(...:orask) shortcuts.

As noted in the Basic Interpolation Code Syntax section, usage of parentheses depends on the context. The following examples are valid:

```
%ask
[[%ask:Name]]
%ask:Name:Joe
%(ask:What is Your Name:Joe Smith)
```

## Ask Code Options

An "ask" code takes two optional parameters. The first, <optional−question>, is the text to display to the user when prompting for the value for that code. The second, <optional−default>, is a default string to preload the text entry field in which the user specifies a value. For example:

```
%(ask<backref>:<optional-question>:<optional-default>)
[[%(ask<backref>:<optional-question>:<optional-default>)]]
```

The <backref> optional parameter is discussed in the Back−references section.

## The Query Dialog for "ask"−modified and "orask"−modified Codes

When a list of strings are interpolated using interpolation shortcuts, the user may be prompted to supply values for certain codes. The user is asked via the *Interpolation Query* dialog box. This dialog will contain one or more text entry fields with labels describing the data expected for that field. These labels are determined from the specific interpolation code (both the "ask" code and the "orask" modifier include an option to specify a label).

The label or question fields maintain recently−entered results that are displayed as a drop−down list of

matching recent entries.

# Path Code

The "path" codes are used to provide special directory paths based on the installation of Komodo that is currently running. These include such items as the common data directory, which may be necessary if you are building run commands that you intend to work on shared files.

## Path Code Syntax

The syntax of the path code is as follows:

```
%(path<backref>:<pathName>)
[[%(path<backref>:<pathName>)]]
```

## Path Code Options

The path code takes one required parameter, "pathName". The pathName may be one of the following:

| Path Name | Meaning |
|-----------|---------|
| **userDataDir** | User–specific data directory where Komodo stores various information and files. |
| **hostUserDataDir** | The userDataDir contains some host–specfic directories. This is most common on systems where user directories are shared and used from more than one computer. |
| **commonDataDir** | The common data directory contains data and files that are shared between multiple users. |
| **installDir** | This is the directory where Komodo is installed. |

# Debugger Code

The "debugger" codes are used to provide runtime values from the debugger subsystem in Komodo. These codes can be used to provide debugging information to applications such as the PerlApp component in ActiveState's Perl Developer Kit.

## Debugger Code Syntax

The syntax of the debugger code is as follows:

```
%(debugger<backref>:<value>)
[[%(debugger<backref>:<value>)]]
```

## Debugger Code Options

The debugger code takes one required parameter, "value". The value may be one of the following:

| Debugger Value | Meaning |
| --- | --- |
| **address** | The hostname or address Komodo is running on. |
| **port** | The TCP/IP port number that the debugger system is listening on. |
| **proxyAddress** | The hostname or address of a debugger proxy that Komodo is using. |
| **proxyPort** | The TCP/IP port number of a debugger proxy that Komodo is using. |
| **proxyKey** | A session key, typically retrieved from the USER environment variable, that the proxy uses to match debug sessions with a specific running instance of Komodo. |

# Pref Code

**This is a very advanced feature and is subject to change in future versions of Komodo.**

The "pref" codes are used to provide values from Komodo's preferences, which are configured in the Preferences dialog box (*Edit*|*Preferences*). Komodo's preference system is undocumented, but you may examine your preference settings in the file "prefs.xml" located in your user data directory (~./komodo/*VERSION* on linux, C:\Documents and Settings\*USERNAME*\Application Data\ActiveState\Komodo\*VERSION* on Windows).

## Pref Code Syntax

The syntax of the pref code is as follows:

```
%(pref<backref>:<prefName>)
[[%(pref<backref>:<prefName>)]]
```

Preference names may change between versions of Komodo.

# Back–References

Back–references are particularly useful for code snippets. You can use back–references to interpolate the same value any number of times in the snippet. Back–references make it possible to prompt the user for an input value only once, and then insert that value multiple times. For example, you could create a snippet that prompts for a value, which would then be entered at various places in the snippet text. Without back–referencing, the user would be prompted as many times as there were instances of the interpolation in the snippet.

## Back–Reference Syntax

You create a back–referenced set of codes by suffixing an interpolation code with a number. The syntax for back–reference is as follows:

```
%(<code><backref>:<options>...)
[[%(<code><backref>:<options>...)]]
```

For example:

```
%(ask1:Name:Trent)
%w1:else:Foo
[[%guid1]]
```

All interpolation code blocks with the same code name and reference number are part of the same back–reference set. All members of the same back–reference set will be replaced with the first code block in that set. For example, consider this run command:

```
echo Hi there %(ask1:Name:Trent). That name again is %ask1
```

This would generate a Query dialog prompting for *one* entry, "Name", with a default of "Trent". Whatever value the user entered for "Name" would then be inserted in *two* places in the command, resulting in the following command:

```
echo Hi there Bill. That name again is Bill
```

Another useful application of back–references is the "guid" code. A guid code is replaced with a new GUID (Globally Unique Identifier). Sometimes it is desirable to have the same GUID inserted in more than one place in a file. In snippets, this can be done by using a code "%guid1" instead of just "%guid" wherever you want the GUID inserted.

---

# Customizing Komodo

Komodo's preferences are used to set the default behavior of Komodo. Preferences can be set for various aspects of Komodo functionality, such as editor behavior, preferred language interpreters, the Komodo workspace layout, etc.

Some preferences can also be configured on a per–file basis. For example, the configuration of line endings, indentation style and word wrap can be configured for individual files. File–specific settings override the default preferences described in this section. To configure file–specific defaults, see File Properties and Settings in the File section of the Komodo documentation.

## Appearance Preferences

Use the Appearance preferences to customize the default layout of the Komodo workspace. The functions described below can also be changed using keyboard shortcuts; see Key Bindings for more information. To customize the Komodo workspace, select *Edit/Preferences/Appearance*. Configure the following options:

*Toolbar Configuration*

- *Show button text*: Descriptive text displayed beneath toolbar icons.
- *Show standard toolbar*: Toolbar with commonly used functions, for example, Open and Save.
- *Show debug toolbar*: Toolbar with debugging functions, for example, Step Over and Step Through.
- *Show source code control toolbar*: Toolbar with source code control functions, for example, Update and Revert.
- *Show macro toolbar*: Toolbar containing commands for recording macros.

*Initial Page Configuration*

- *Show Komodo Start Page on startup*: Select to display the Start Page in the Editor Pane when Komodo launches.
- *Hide Tutorials Pane*: Select to hide the Tutorials section of the Komodo Start Page.
- *Hide Quick Links Pane*: Select to hide the Quick Links section of the Komodo Start Page.
- *Hide Tip of the Day Pane*: Select to hide the Tip of the Day section of the Komodo Start Page.

*Most Recently Used*

- *Number of Projects*: The number of projects displayed on Komodo's Start Page, and on the Recent Projects menu.
- *Number of Files*: The number of files displayed on Komodo's Start Page, and on the Recent Files menu.

# Code Intelligence Preferences

Use the Code Intelligence preferences to enable the Code Browser, Object Browser, and Python AutoComplete and CallTips functionality. Also, run wizards to scan language installations and custom directories to build the Code Intelligence database. To configure Code Intelligence preferences, select *Edit|Preferences|Code Intelligence*.

- *Code Intelligence*: Enables or disables code intelligence functionality (Code Browser, Object Browser, CallTips, and AutoComplete). Select the check box to enable Code Intelligence (enabled by default). De−select the check box to disable Code Intelligence. Click *OK* to save changes.
- *Scan language installations*: A wizard that scans specified language installations to update the Code Intelligence database. An updated database increases the accuracy of Code Intelligence features (for example, the Object Browser). Note that Python requires this database for CallTips and AutoComplete to operate fully. To run the wizard:
    1. Click *Scan language installations* to start the wizard.
    2. Select the languages you wish to use with Code Intelligence, and then click *Next*.
    3. Verify that the listed languages are correct. If correct, click *Build Now* to construct the Code Intelligence Database. Building the database may take several minutes, depending on the number of languages selected. (Alternatively, Click *Back* to edit the language list. Click *Cancel* to exit the wizard.)
    4. Click *Finish* to exit the wizard.
- *Scan custom directories*: A wizard that scans specified directories to update the Code Intelligence database. An updated database increases the accuracy of Komodo Code Intelligence features. To run the wizard:
    1. Click *Scan custom directories* to start the wizard.
    2. Click *Add directory* to browse for and select the directory to add. Repeat to add further directories. Click *Next*.
    3. Verify that the listed directories are correct. If correct, click *Build Now* to construct the database. Building the database may take several minutes, depending on the number of directories selected. (Alternatively, Click *Back* to edit the directory list. Click *Cancel* to exit the wizard.)
    4. Click *Finish* to exit the wizard.
- *Enable automatic AutoComplete and CallTip triggering while you type*: Access AutoComplete and CallTip functionality while programming in Komodo. Select the check box to enable automatic triggering (enabled by default). De−select the check box to disable automatic triggering. Click *OK* to save changes.
  Note that AutoComplete and CallTips can be manually invoked via the associated key binding when the cursor is placed on the code fragment in the editor.
- *Enable re−scanning of the current file while editing*: Background file scanning updates the Code Intelligence database while a file is being edited. Select the check box to enable re−scanning (enabled by default). De−select the check box to disable file background scanning. Click *OK* to save changes.
  Note that scanning may be CPU intensive for large files. If slowdowns are experienced, disable this feature and use *Refresh Status* (*File|Refresh Status*) to manually re−scan the current file and to update the Code Intelligence database.

# Debugger Preferences

To customize general debugging functions, select **Edit/Preferences/Debugger**. For language−specific settings (such as interpreter selection), see the Language preference.

*Debugging Session Startup*

- ***When starting a new debug session***: Specify whether Komodo should ***Ask me what files to save*** (which displays a list of changed files); ***Save all modified files*** (whereby all modified files are automatically saved); or ***Save no files*** (whereby the debugging session starts without saving any files).
- ***When receiving a remote debugging connection***: Specify whether Komodo should ***Ask me to allow connection*** (Komodo prompts to allow the connection request); ***Allow the connection*** (Komodo accepts the connection without prompting); or ***Refuse the connection*** (Komodo refuses the connection without prompting).
- ***Skip debugging options dialog***: To block the display of the debugger dialog box when the debugger is invoked, check this box. Using the 'Ctrl' key in conjunction with a debugger command key toggles the value specified here. However, commands invoked from the ***Debugger*** drop−down menu always use the convention specified here.

*Debugging Session Shutdown*

- ***Confirm when closing debugger session tab***: If the debugger is running when you attempt to close the Debug tab, this check box determines whether you are prompted to halt the debug session and close the tab, or whether this happens without prompting.

*Debugger Connection Options*

- ***Listen for debug connections on port***: The port Komodo listens on for debugger sessions. The default is 9000. Set the port to 0 to use a free port provided by the system. On multi−user systems where multiple instances of Komodo are running, using port 0 to select a system−specified port prevents you from having to change the port specification from one debugging session to the next.
- ***Enable Debugger Proxy***: Select this check box to enable a debugger proxy.
- ***Proxy Listener Address***: The address of the proxy Komodo uses to listen for connections. It supports remote debugging on a multi−user system. By default, the DBGP Proxy uses port 9000 to listen for remote debuggers and port 9001 to listen for connections from Komodo. The proxy must be started separately at the command line.
- ***Proxy Key***: In a multi−user system, the key identifies which instance of Komodo requires the connection. By default, set to the USER or USERNAME environment variables on the system where Komodo is running.
- ***Try to find files on the local system when remote debugging***: By default, when Komodo performs remote debugging, it retrieves a read−only copy of the file to be debugged from the debug engine. When this check box is selected, however, Komodo first searches for the debugger

file on the local system. While it is probably safe to leave this check box selected for all of your remote debugging, there is a slight possibility that Komodo retrieves the wrong file if remote debugging is performed on another machine. If, by chance, there is a file on your local system with the same name and location as the file on the remote system, Komodo uses the local file. This would only happen if the names and locations were identical (e.g., if both machines contained a file called "C:\foo\bar\baz.pl").

# Editor Preferences

To configure editing preferences, select *Edit/Preferences/Editor*.

*General Preferences*

- *Show whitespace characters*: Display or hide whitespace characters in the editor. Spaces are displayed as dots; tab characters appear as right arrows.
- *Show end−of−line characters*: This option sets the default for displaying end of line markers. Display can also be toggled using the *View/View EOL Markers* menu option.
- *Show line numbers*: This option sets the default for displaying line numbers. If enabled, line numbers are displayed on the left side of the Editor Pane. Line numbers can also be toggled using the *View/View Line Numbers* menu option.

Options set through the Preferences dialog box are the default for all files opened in Komodo. Some display characteristics can be assigned to individual files.

*Confirmation Dialogs*

When files that are opened in the Komodo editor are changed by another application, Komodo can be configured to respond in various ways:

- *Detect when files are changed outside the environment*: When this option is enabled, Komodo pays attention to changes made to files outside the Komodo environment.
- *If files have been changed*: When files are changed outside Komodo, select whether Komodo should *Ask me what files to reload* (prompt for reload confirmation); *Reload all files* (reload without prompting); or *Reload no files* (do nothing).
- *If files have been deleted*: When files are deleted outside Komodo, select whether Komodo should *Ask me what files to close* (prompt for close confirmation); *Close all files* (close without prompting); or *Close no files* (do nothing).

If *Ask me what files to reload Ask me what files to close* are selected, the prompt is displayed when:

- changing between tabs in the editor
- switching back to Komodo from another application
- saving a file
- deleting a file

*Scrolling*

The Scrolling setting determines the number of lines that are be displayed above or below the editing cursor. As the editing cursor moves, the number of lines specified here are displayed between the cursor and the top or bottom of the Editor Pane. You can also set the horizontal scroll bar width by entering the desired size in pixels.

*Incremental Search*

These options set the defaults for the Incremental Search feature.

- *Matches Case*: Specify whether Incremental Search should be case sensitive.
- *Uses*: Specify the search syntax type. *Plain Text* exactly matches the search string; *Regular Expressions* interprets the search text as a regular expression; *Wildcard* interprets asterisk and question mark characters as wildcards.

# Configuring Key Bindings

Most Komodo functions can be invoked via key bindings. These key bindings can be customized. To view an HTML list of the key bindings currently in effect, select *Help/List Key Bindings*. Refer to the Key Binding List for a list of the default key bindings.

On Unix systems, key bindings defined in the window manager (including default key bindings) take precedence over Komodo key bindings. If certain keys or key combinations do not work as expected in Komodo, check the window manager's key binding scheme. In the case of conflicts, change either the Komodo key bindings or the window manager key bindings.

To configure key binding defaults, select *Edit/Preferences/Editor/Key Bindings*. By default, menu key bindings are accessed using 'Alt' key combinations. For example, the *File* menu is opened via 'Alt'+'F'. Select *Remove Alt−<letter> shortcuts from menus* to disable menu access via these key bindings. The 'Alt' key still activates the *File* menu.

*Key Binding Schemes*

Key binding "schemes" are sets of pre−configured key bindings. The *Default* scheme is consistent with common Windows key bindings. The *Emacs* scheme contains many of the key bindings associated with the Emacs editor. The Emacs scheme is not a comprehensive set of Emacs key bindings. Only the most commonly used Emacs functions are included. Some of the default Emacs key bindings use 'Alt' key combinations that are also used to access Komodo menus. To disable the menu access, select *Remove Alt−<letter> shortcuts from menus*.

Pre−configured schemes cannot be modified. When you attempt to modify a key binding, you are prompted to make a copy of the scheme before making changes.

*Modifying Key Bindings*

To alter or view a specific key binding, scroll the **Commands** list or enter characters in the filter field. If multiple key bindings are assigned to a single command, the **Current Key Sequence** field displays as a drop−down list. Click the **Clear** button to delete the key binding displayed for the selected command; click **Clear All** to delete all key bindings for the selected command.

To add a new key binding for the selected command, enter the desired key binding in the **New Key Sequence** field. If the key sequence is already assigned to another command, the current assignment is displayed in the **Key Sequence Already Used By** field. Click **Change** to update the key binding displayed in the **Current Key Sequence** field; click **Add** to make the new key binding an additional key binding. If the key binding is already assigned, the original assignment is cleared.

*Key Bindings for Custom Components*

Custom key bindings can be assigned to the following types of components:

- Open Shortcuts
- URLs
- Run Commands
- Macros
- Snippets
- Templates

When the key binding associated with a component is invoked, it has the same action as double−clicking the component in the Toolbox or Project Manager.

To assign a key binding to a component, or to alter or delete an existing key binding, right−click the desired component in the Toolbox or Project Manager to display the Properties dialog box, then click the **Key Binding** tab. Configure as described above.

## Configuring Indentation

From the **Edit** menu, select **Preferences**, then click **Editor|Indentation**.

- **Auto−Indent Style**: Choose from one of three indentation styles:
  - ♦ **Use Smart Indent**: Komodo automatically anticipates logical indentation points, based on language cues (such as open braces).
  - ♦ **Indent to first non−empty column**: Komodo maintains the current level of indentation.
  - ♦ **Don't auto−indent**: Select to prevent all forms of automatic indentation.
- **Auto−adjust closing braces**: Komodo automatically aligns closing braces with corresponding opening braces.
- **Show indentation guides**: Select to display indentation markers (grey vertical lines). An indentation marker is displayed every time the number of spaces on the left margin equals the

value specified in the Number of spaces per indent field.

- *Allow file contents to override Tab settings*: If selected when files are open, Komodo uses the indentation settings saved in the file, possibly overriding the other preferences. If de−selected, Komodo uses the preference configuration regardless of the indentation values in the file.
- *Prefer Tab characters over spaces*: Komodo displays Tab characters wherever possible, according to the values specified in the *Number of spaces per indent* and the *Width of each Tab character* fields. When the 'Tab' key is pressed, Komodo inserts indentation up to the next indent width. If the new indentation is a multiple of the Tab width, Komodo inserts a Tab character. *Example*: With a Tab width of 8 and an indent width of 4, the first indent is 4 spaces, the second indent is a Tab character, and the third indent is a Tab character plus 4 spaces.

Tab and indent widths are specified as follows:

- *Number of spaces per indent*: Number of spaces Komodo inserts on the left margin when indenting a line of code.
- *Width of each Tab character*: Number of spaces that are equal to a Tab character.
- *Fold mark style*: Use the drop−down list to select the style of node used in code folding.
- *Use horizontal line on folds*: Displays collapsed code with fold marks; a thin line also spans the width of the Editor Pane.
- *'Backspace' decreases indentation in leading whitespace*: If this option is enabled, pressing 'Backspace' clears an entire indentation, rather than a single space, if there is nothing between the editing cursor and the left margin. For example, if the number of spaces per indent is set to four, and there are five spaces between the left margin and the editing cursor, pressing 'Backspace' once clears one space; pressing 'Backspace' a second time clears four spaces.
- *Restore fold state on document load (slows down file opening)*: If this option is enabled, the current state of code folding is remembered when a file is closed, and reinstated when the file is next opened.

Options set through the Preferences dialog box are the default for all files opened in Komodo. Some indentation characteristics can be assigned to individual files.

# Smart Editing

## Background Syntax Checking

Background syntax checking validates code against the language interpreter as you type. (If Code Intelligence is enabled for Python, the code intelligence database is used to validate Python code.) Syntax errors and warnings are underlined in the Editor Pane. See Background Syntax Checking for more information.

To turn background syntax checking on or off, select *Preferences/Editor/Smart Editing*.

The level of background syntax checking for Perl is determined by the setting on the Perl Language preference page.

### Configuring Word Completion

The Komodo editor maintains an index of words in the current file. Instead of re–entering words that already exist in the current file, you can use the Complete Word function to finish words. If you are using the default key binding scheme, word completion is invoked from the keyboard by pressing 'Ctrl'+'Space'. To configure Komodo to invoke word completion with the 'Tab' key, select the check box labeled *Use tab character to complete words like Ctrl+Space*.

### Configuring Word Wrap

If word wrap is enabled, lines are automatically "wrapped"; that is, when a line exceeds the width of the Editor Pane it wraps to the next line. This is merely a display characteristic – no end–of–line marker is inserted. Use the *Word wrap long lines* option to configure this behavior.

*Note*: For lines that have been wrapped automatically, the behavior of the 'Home' and 'End' keys is slightly different. Pressing 'Home' or 'End' moves the cursor to the beginning or end of the current line. Pressing the same key a second time moves the cursor to the previous or next end–of–line marker.

### Configuring Edge Lines

The edge line is a vertical line that indicates a column marker.

- *Show edge line / Highlight characters beyond edge line*: Select to show where the line wraps, and to highlight characters beyond the wrap column. With fixed–width fonts, a line is drawn at the column specified. With proportional–width fonts, those characters beyond the specified column are drawn on a colored background. The line or background color is configured on the Fonts and Colors preference page.
- *Edge line column*: Specify the column position of the vertical marker.

Options set through the Preferences dialog box are the default for all files opened in Komodo. Some Smart Editing features can be assigned to individual files.

## Save Options

To automatically fix whitespace errors when saving files:

- *Clean trailing whitespace and EOL markers*: Eliminates unnecessary empty space between text and EOL markers, and fixes inappropriate EOL markers.

- *Ensure file ends with EOL marker*: Adds an EOL marker to the last line in a file if one does not already exist.

Based on the specified *Minutes between auto−save*, Komodo saves backup copies of all files open in the editor. When Komodo is shut down normally, the backup copies are deleted. If Komodo is shut down abnormally (such as through a system crash), Komodo prompts to restore the backup copy when the file is next opened. If you respond "Yes", the backup copy of the file, rather than the (older) disk copy, is opened in the editor.

When files without extensions are saved, Komodo can be configured to prompt for an action. Configure the *If filename has no extension* drop−down list:

- *Ask me what to do*: Komodo prompts you with a dialog box to decide what to do when a particular file is saved without an extension.
- *Add appropriate extension*: Komodo automatically adds an extension based on file content.
- *Leave filename alone*: Komodo does nothing when a file is saved without an extension.

# File Associations

Komodo's file associations determine the functionality of editing features such as AutoComplete and code coloring. Use the *File Associations* preference to associate file extensions and characteristics with particular languages.

*Editing the Language Associated with a File Pattern*

To edit the language associated with a file pattern:

1. Select the desired extension from the *Patterns* list.
2. From the *Language* drop−down list, select the language to associate with the selected file pattern.

To remove an association, select the desired pattern and click *Remove*.

*Adding a New File Association*

To add a new file pattern/language association:

1. Enter the desired pattern in the *Pattern* field. The pattern consists of the wildcards and the naming convention. Typically, file associations are made by the filename extension; for example, a Perl script has the extension ".pl". The pattern for a Perl script is therefore "*.pl".
2. Select the language to associate with the pattern from the *Language* drop−down list.

*Use File Content to Determine Language*

Komodo can be configured to identify the language of a file based on its contents rather than its extension. The following characteristics can be used to override the file associations settings for syntax checking and debugging configuration.

- *XML Declarations*: The *Use XML Declarations* option checks for XML declarations that specify the language of a file (e.g. <!DOCTYPE html PUBLIC "–//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1–strict.dtd"> for XHTML 1.0).
- *Shebang (#!...) Line*: The *Use shebang line* option checks for a "#!/..." line at the top of a file that specifies the interpreter (e.g. `#!/usr/bin/perl`).
- *Emacs–Style Mode Variable*: When this check box is selected, as Komodo opens files, it checks for an embedded Emacs "mode" specification used to set the syntax checking and debugging configuration.

# Fonts and Colors Preferences

Komodo's Fonts and Colors Preference is used to customize the display of text in the Editor Pane. To modify the font and color preferences, from the *Edit* menu, select *Preferences*, then click *Fonts and Colors*.

The *Sample Text* window at the top of the Fonts and Colors page offers a preview of the current scheme. If multiple schemes are configured, select the desired scheme from the *Scheme* drop–down list.

To create a new scheme:

1. Select the scheme that you want to base your new scheme upon.
2. Click the *New* button and enter a name for the new scheme.
3. Make any necessary changes using the controls on the Fonts, Colors, Common Syntax Coloring, and Language–Specific Coloring tabs.
4. Click *OK* to save the new scheme.

Schemes are added to the *Scheme* drop–down list. Remove the selected scheme by clicking the *Delete* button. System schemes appear in bold and cannot be deleted.

## Fonts

The *Fonts* tab is used to configure the display characteristics for fixed–width and/or proportional fonts. Note that the default font characteristics configured on this tab are not overridden by any language–specific font configurations.

To create a scheme that affects characters in specific encodings, select the type of encoding from the *Encoding* drop–down list. If you do not specify the encoding, Komodo uses the system's default encoding.

The *Fonts* tab contains two identical sets of font controls, one for fixed–width fonts, and the other for proportional fonts. Select the *Prefer Fixed* or *Prefer Prop.* option button to set the desired font type and associated settings.

- *Font*: Select specific fixed–width or proportional fonts from the drop–down lists. (On Linux and Solaris, the same list of fonts is displayed in both drop–down lists. On GTK, there is no programmatic way to identify whether a font is proportional or not; therefore, you must know the properties of the individual fonts when modifying these values.)
- *Font Size*: Select the size of the font from the drop–down list.
- *Bold*: If you want the default font to be displayed in bold, click the "B".
- *Italic*: If you want the default font to be displayed in italics, click the "I".
- *Font Color*: Set the font color by clicking on the "Fore" color box. Select the desired color from the color picker dialog box. Click the small arrow button to set the font color using the system color palette.
- *Background Color*: Set the background color for the Editor Pane by clicking on the "Back" color box. Select the desired color from the color picker dialog box. Click the small arrow button to set the background color using the system color palette.
- *Reset*: Click this button to return the font and color preferences to Komodo's original, default settings.

## Colors

Use the *Color Choice* drop–down list on the *Colors* tab to configure general color properties for the Editor Pane. After selecting an interface component from the list, click the box to the right of the drop–down list to choose a color from the color palette, or click the small arrow button to select a color from the system color palette.

The following interface elements can be configured:

- *Current Line Background Color*: The color configured here does not take effect unless the *Highlight Current Line* check box is selected. This sets the highlighting color of the line in which the editing cursor is positioned.
- *Cursor Color*: Sets the color of the editing cursor.
- *Selection Background Color*: The background of text that has been selected in the Editor Pane (by double–clicking, or clicking and dragging), is colored according to this setting.
- *Selection Text Color*: This option is only available if the *Override Text Color in Selection* check box is selected. Regardless of other color configurations, all text that has been selected in the Editor Pane (by double–clicking, or clicking and dragging), is displayed in the color specified in this setting.
- *Active Breakpoints Color*: Sets the color of the breakpoint at which the debugger is currently stopped.
- *Pending Breakpoints Color*: Sets the color of breakpoints at which the debugger has yet to stop.
- *Bookmark Color*: Sets the color of the bookmarks that are inserted in the margin to the left of the Editor Pane.

- *Debugger Current Line Background Color*: Sets the background color of the line highlighted by the debugger.
- *Debugger Calling Line Background Color*: Sets the color of lines that call subroutines. Caller line coloring is applied only when you have changed the stack position to view the line that calls the current line. View caller lines in the *Call Stack* drop−down list box on the *Variables* tab on the *Debug* tab.
- *Edge Line/Background Color of Text Too Far*: If [Word Wrap](#) is enabled, use this option to set the color of the word wrap column marker, as well as the highlighted characters beyond the wrap column. If using fixed−width fonts, a line is drawn at the specified column. If using a proportional−width font, characters beyond the specified column are drawn on a colored background.

The *Override Text Color in Selection* check box activates the "Selection Text Color" setting described above. The *Highlight Current Line* check box activates the coloring specified in the "Current Line Background Color" setting described above.

## Common Syntax Coloring

Some language elements are common to a number of programming languages. The element colors specified on the *Common Syntax Coloring* tab applies to all languages that use these elements. Select an element from the *Element Type* drop−down list and use controls described below to set the font characteristics. Note that the font characteristics configured on this tab are overridden by any [language−specific](#) font configurations.

- *Face*: Select the typeface of the font from the drop−down list. You can choose either "Fixed−width" or "Proportional".
- *Size*: Select the size of the font from the drop−down list.
- *Bold*: If you want the default font to be displayed in bold, click the "B".
- *Italic*: If you want the default font to be displayed in italics, click the "I".
- *Font Color*: Set the font color by clicking on the foreground color box. Select the desired color from the color picker dialog box, or click the small arrow button to select a color from the system color palette.
- *Background Color*: Set the background color for the Editor Pane by clicking on the background color box. Select the desired color from the color picker dialog box, or click the small arrow button to select a color from the system color palette.
- *Reset*: Click this button to return the font and color preferences to Komodo's original, default settings.

## Language−Specific Coloring

The colors configured on the *Language−Specific Coloring* tab apply to elements that appear in a specific language. Select a language from the *Language* drop−down list and an element from the *Element Type*

drop−down list, then use the controls described below to set the font characteristics.

- *Face*: Select the typeface of the font from the drop−down list. You can choose either "Fixed−width" or "Proportional".
- *Size*: Select the size of the font from the drop−down list.
- *Bold*: If you want the default font to be displayed in bold, click the "B".
- *Italic*: If you want the default font to be displayed in italics, click the "I".
- *Font Color*: Set the font color by clicking on the foreground color box. Select the desired color from the color picker dialog box, or click the small arrow button to select a color from the system color palette.
- *Background Color*: Set the background color for the Editor Pane by clicking on the background color box. Select the desired color from the color picker dialog box, or click the small arrow button to select a color from the system color palette.
- *Reset*: Click this button to return the font and color preferences to Komodo's original, default settings.

# GUI Builder Preferences

To set the default language for GUI Builder projects, select it from the drop−down list. If *Ask each time* is specified, Komodo prompts every time the GUI Builder is invoked.

Komodo communicates with the GUI Builder using the port indicated in the *TCP/IP Port used* field. Alter the port number as desired.

# Interactive Shell Preferences

The Interactive Shell is an implementation of the language interpreter's shell within the Komodo environment. These preferences set the default behavior for interactive shell functionality.

- *Preferred Language*: Specify which language interpreter's shell is launched when the interactive shell is invoked.
- *Session Control*:
  - ♦ *Close tab when interactive shell session ends*: If this option is selected, the *Shell* tab closes when the *Stop* button is clicked. Otherwise, the tab remains visible (although you must invoke another interactive shell session to use the shell).
  - ♦ *Confirm when closing interactive shell*: When you attempt to close the *Shell* tab before stopping the session (by clicking the *Stop* button), this option determines whether you are prompted for confirmation. The confirmation dialog box has an option to disable the warning; to re−enable the warning, set this field to *Ask me each time*.
- *Working Directory*: This option sets the "current" directory for the interactive shell session. Specify the desired directory.

# Internationalization Preferences

Language encodings provide support for files containing characters in non−ASCII character sets.

Encodings are determined in the following order:

1. *File Preference*: If a specific encoding has been assigned to a file via the file's Properties and Settings context menu, the assigned encoding is always used when that file is opened.
2. *Auto−Detect*: If the *Auto−Detect File Encoding when Opened* box is checked, Komodo analyzes the existing encoding of the file by first looking for a Byte Order Marker (BOM), then by checking for an XML declaration, and then by performing heuristic analysis on the file's contents. If an encoding can be determined, it is applied.
3. *Language−specific Default Encoding*: Specific encodings can be assigned to programming languages. (Komodo determines the programming language of a file based on the File Association preferences.) If an encoding is associated with a programming language, that encoding is used. Check *Signature (BOM)* to embed a Byte Order Marker (BOM) at the beginning of the file. If the specified encoding is set to the default encoding, the *System Encoding or Custom Encoding* is used.
4. *System Encoding or Custom Encoding*: If the *Use Encoding Defined in Environment* box is checked, Komodo uses the encoding specified in the operating system. The following system variables are checked:
   - ◆ *Windows*: The Control Panel's "Regional Settings" (Windows 98, ME, and NT); "Regional Options" (Windows 2000); "Regional and Language Options" (Windows XP).
   - ◆ *UNIX*: LC_CTYPE, LANG and LANGUAGE.
   To use a different encoding, uncheck this box and select the desired encoding from the *Custom Encoding* drop−down list.

When you create a new file, only the third and fourth methods described above are used to set the file's encoding.

The following settings override all other encoding settings except the *File Preference* setting.

- *Allow XML Declaration to Override Auto−Detection*: Komodo always uses the XML encoding declaration contained in the XML file when opening XML files (if applicable).
- *Allow HTML META tag to Override Auto−Detection*: Komodo uses the charset setting defined in META tags in HTML documents.
- *Allow 'coding:' tag to Override Auto−Detection*: If the file contains a "coding: <encoding_name>" directive within the first two lines, that encoding is used.

The *Date & Time* format determines the display format of the date and time for items listed on the Start Page, and for the Current File settings display.

# Language Help Settings

Use the *Language Help* page in Komodo Preferences (*Edit*/*Preferences*/*Language Help*) to configure context–sensitive language look–up.

## Configuring Reference Locations

The *Language Lookup Commands* section of the Language Help page displays the default URL for language–specific help. (The `%(browser)` string is an interpolation shortcut.) If you are using the default key binding scheme, 'Shift'+'F1' opens a browser window and looks up the address of the sites specified here. The site is selected according to the type of file currently active in the Editor Pane. (To configure file association, see File Associations.)

The *General Help* field is used to specify a help location that does not specifically apply to a language (or applies to a language not available in the above list).

To reset any of the help settings to their original value, click *Reset* beside the pertinent field.

## Using Language Help

In the Editor Pane, double–click to select the keyword that you want to look up. Then, if you are using the default key binding scheme, press 'Shift'+'F1' to invoke a browser window and look up the keyword on the site configured in the Preferences. Press 'Ctrl'+'F1' to perform the lookup using the site configured in the *General Help* field on the Language Help page.

# Language Configuration

To configure the languages supported by Komodo, select *Edit*/*Preferences*/*Languages*, then select the desired language.

## Configuring Perl

- *Use this interpreter*: Select *Find on Path* to use the first Perl interpreter that occurs in the system's `PATH` variable. The paths to interpreters found in the `PATH` variable are available from the drop–down list; select a specific interpreter as desired. Alternatively, click *Browse* and navigate the filesystem to select the desired interpreter.
- *Background Syntax Checking*: Perl syntax checking is configurable; the degree of syntax checking is determined by switches sent to the interpreter. Specify the desired level of syntax checking by selecting the corresponding interpreter switch combination from the drop–down list. If a setting that uses "taint" mode is selected, the `PERL5LIB` environment variable is ignored; syntax checking is not performed on modules located in directories specified via `PERL5LIB`.

- ***Debugger Logging***: If this option is enabled, the Komodo debugger logs the debugging session to a file in the directory specified in the ***Debugger Log Path*** field (or the directory specified in the system's `TEMP` variable, if no directory is specified). This is primarily for debugging the debugger, as opposed to gaining additional insight on the debug session itself. The debugger log file is named *perl−dbgp.log*. The contents of the log file are overwritten each time the debugger is invoked.
- ***Additional Perl Import Directories***: Directories specified in this field are inserted at the beginning of Perl's `@INC` array (in the same manner as Perl's "I" command−line argument). Modules in the specified directories are used for debugging, syntax checking and during interactive shell sessions.

## Configuring PHP

Click the ***PHP Debugger Configuration Wizard*** button to configure the location of the PHP interpreter, to modify the `php.ini` file for Komodo debugging, and to install the required debugging extensions. To manually configure PHP debugging, refer to Debugging PHP for instructions.

- ***Use this interpreter***: Select ***Find on Path*** to use the first PHP interpreter that occurs in the system's `PATH` variable. The paths to interpreters found in the `PATH` variable are available from the drop−down list; select a specific interpreter as desired. Alternatively, click ***Browse*** and navigate the filesystem to select the desired interpreter.
- ***Path to alternate PHP configuration file***: The `php.ini` file must be modified to support Komodo debugging. To specify an different `php.ini` than the one configured by the ***PHP Debugger Configuration Wizard***, enter the path in this field, or use the ***Browse*** button. See Debugging PHP for information about manually configuring the `php.ini`.

*Note*: Be sure your `php.ini` configuration file is located in your operating system directory. If you used the PHP Windows installer, this file should be in the correct location. To verify, on Windows 2000/NT the `php.ini` file should be in `\winnt`; on Windows 98/Me the `php.ini` file should be in `\windows`. On Windows XP, the system directory is either `\winnt` or `\windows`, depending on whether XP was a native installation or was an upgrade from a previous Windows version.

### Sharing PHP Preferences and Files

Use Komodo's shared support functionality to share PHP preferences, run commands, code snippets, templates, .tip files, or other items that have special usefulness within your PHP programming group. See Configuring Shared Support for more information.

## Configuring Python

- ***Use this interpreter***: Select ***Find on Path*** to use the first Python interpreter that occurs in the system's `PATH` variable. The paths to interpreters found in the `PATH` variable are available from

the drop−down list; select a specific interpreter as desired. Alternatively, click *Browse* and navigate the filesystem to select the desired interpreter.
- *Additional Python Import Directories*: Directories specified in this field are inserted at the beginning of Python's PYTHONPATH environment variable. Modules in the specified directories are used for debugging, syntax checking and during interactive shell sessions.

## Configuring Tcl

Komodo provides the ability to interact with both the standard Tcl interpreter ("Tclsh") and the Tcl interpreter that supports the Tk widget library ("Wish"). Komodo's Tcl integration also supports logging and syntax checking. Extended Tcl editing support and the Tcl debugging libraries are included with a subscription to ASPN Tcl.

- *Use this Wish interpreter*: Select *Find on Path* to use the first Wish interpreter that occurs in the system's PATH variable. The paths to interpreters found in the PATH variable are available from the drop−down list; select a specific interpreter as desired. Alternatively, click *Browse* and navigate the filesystem to select the desired interpreter.
- *Use this Tclsh Interpreter*: As described above, specify the desired Tclsh interpreter.
- *Enable Debugger Log*: If this option is enabled, the Komodo debugger logs the debugging session to a file in the directory specified in the *Debugger Log Path* field (or the directory specified in the system's TEMP variable, if no directory is specified). This is primarily for debugging the debugger, as opposed to gaining additional insight on the debug session itself. The debugger log file is named *tcl−dbgp.log*. The contents of the log file are overwritten each time the debugger is invoked.
- *Additional Tcl Include Directories*: Directories specified in this field are inserted at the beginning of Tcl's TCLLIBPATH environment variable. Modules in the specified directories are used for debugging, syntax checking and during interactive shell sessions.

### Tcl Syntax Checking

To specify Tcl syntax checking:

- *Warning messages to suppress*: The warning messages listed in this dialog box can be disabled. This prevents Komodo's syntax checking functionality from reporting these warnings.
- *Error messages to suppress*: The error messages listed in this dialog box can be disabled. This prevents Komodo's syntax checking functionality from reporting these errors.
- *Additional options*: Configure the level of error and warning checking by using the switches −W1 (display parsing and syntax errors), −W2 (display parsing and syntax errors, and usage warnings), −W3 (display parsing and syntax errors, portability warnings, upgrade warnings, performance warnings, and usage warnings), and −Wall (displays all messages and errors (the default)). Additionally, specific warning and error messages can be suppressed using the −suppress *error* switch.

- *Force checking for specific Tcl/Tk Version*: To use a version of Tcl other than the default (8.4) for warning and error checking, select the desired version from the drop–down list.

*Sharing Tcl Preferences and Files*

Use Komodo's shared support functionality to share Tcl preferences, run commands, code snippets, templates, .tip files, or other items that have special usefulness within your Tcl programming group. See Configuring Shared Support for more information.

## Configuring HTML

Komodo works in conjunction with HTML Tidy to provide configurable syntax checking for HTML files. The following options can be configured:

- *Error Level*: *Errors Only* displays all HTML errors with a red underline; *Errors and Warnings* displays both errors and warnings with a red underline.
- *WAI Accessibility Conformance level*: The Web Accessibility Initiative (WAI) provides HTML developers with guidelines for making web content accessible to those with disabilities. These guidelines include methods for making content understandable and navigable (for example, adding "alt" text to an "img" tag for those who cannot view images). WAI accessibility levels are:
    - ◆ *Off*: WAI accessibility is off. No WAI–related syntax errors are reported.
    - ◆ *Priority 3*: The lowest WAI conformance level. One or more groups will have difficulty accessing the information in this document.
    - ◆ *Priority 2*: Satisfying this level removes significant barriers to accessing content in this document.
    - ◆ *Priority 1*: The highest WAI conformance level. A web content developer must satisfy this level for the greatest content accessibility.
- *Configuration File*: Tidy functionality can be customized via a custom configuration file. See teaching Tidy about new tags on the W3C site for information on building a custom configuration file. To specify a custom Tidy configuration file, click *Browse* beside the *Configuration File* text box to locate the configuration file on your filesystem.

## New Files Preferences

When the *New* button is used to create a new file, Komodo, by default, opens a text file in the Editor Pane. To alter the default, select the desired file type from the drop–down list. To specify the end–of–line marker for new files, select the desired marker from the drop–down list.

The Komodo templates used to create new files *(File|New|New File)* support the same Interpolation Shortcut codes as snippets and run commands. Prior to Komodo Version 2.5, only a limited set of variables could be used (for example, to embed the current date and time in files created from custom

templates). The new Interpolation Shortcuts are more powerful but are backward–incompatible.

Select the first check box under *Templates* in New File Preferences to display a warning prompt whenever you attempt to create a new file using a template containing the old type of template code.

Enter a number in the *Number of recent templates to remember* field to specify how many recent template names appear on the *File/New* drop–down menu.

The encoding for new files is determined by the configuration of the Internationalization preference.

# Printing Preferences

- *Print Line Numbers*: Check this box to print the line numbers.
- *Print in Color*: To print in the colors displayed in the Editor Pane, check this box.
- *Wrap long lines at n characters*: Set the column at which lines will wrap. Specify "0" characters for no line wrapping.
- *Scale font sizes from screen to print by n*: Specify the number of times larger or smaller the printed font size will be in relation to its size on screen. The default is "1.5". Specify "1" to print the current font size.

# Projects and Workspace Preferences

*Workspace*

Use the *When starting Komodo* field to specify the display when Komodo is opened.

- *Ask me whether to restore workspace*: Komodo prompts to open recent files and projects.
- *Restore last workspace*: Komodo displays the workspace exactly as it was when you last quit Komodo (including expanded tabs and open files).
- *Do not restore last workspace*: Komodo displays the default workspace (the Start Page and no expanded tabs).

*Opening and Closing Projects*

These options specify the relationship between projects and files that are open in the Editor Pane.

When opening a project, set Komodo to:

- *Ask me what to do*: Komodo prompts whether the files that were open when the project was last closed should be re–opened.
- *Open recent files*: Komodo automatically opens the files that were open when the project was last closed.

- **Open no files**: Komodo opens the project without opening any files.

When closing a project, set Komodo to:

- **Ask me what to do**: Komodo prompts whether open files associated with the project should be closed.
- **Close all open files in project**: Komodo automatically closes open files associated with the project.
- **Close no files**: Komodo closes no files.

### File Status Updates in Project Manager

The **Update file status automatically** option enables a periodic check of the read/write status and the source code control status of components stored in the Project Manager and the Toolbox.

Status refresh can also be performed manually; see Refreshing Project Status for more information.

### Importing Files From Disk

Specify the defaults for the **Import from File System** option, available in the option in the Project Manager, the Toolbox, and in folders stored in either a project or the Toolbox. These defaults can be overridden in the **Import from File System** dialog box.

- **Filenames to include**: Specify the filenames to include. Use wildcards ("*" and "?") to specify groups of files. Separate multiple file specifications with semicolons. If the field is left blank, all files in the specified directory are imported.
- **Filenames to exclude**: Specify the file and directory names to exclude. Use wildcards ("*" and "?") to specify groups of files. Separate multiple file specifications with semicolons. If the field is left blank, no files in the specified directory are excluded.
- **Import Subdirectories Recursively**: To import subdirectories located beneath the directory specified for the import, check **Import Subdirectories Recursively**. Specify how Komodo should handle subdirectories by selecting one of the following options:
  - ♦ **Import directory structure**: If the **Import Subdirectories Recursively** box is checked and this option is selected, Komodo creates folders within the project that represent imported directories. Thus, the directory structure is preserved within the project.
  - ♦ **Make a folder per language**: If this option is selected, imported files are organized into folders according to the language indicated by file pattern in the filename. File associations are configured in the Komodo Preferences. Each folder is named after the associated language, for example, "Perl files", "XML files", etc. Files that don't correspond to a known file pattern are stored in a folder called "Other files".
  - ♦ **Make one flat list**: If this option is selected, all the imported files are placed directly under the project or folder from which the **Import from File System** command was invoked.

### Triggering Macros

Macros can be configured to execute when specific Komodo events occur (such as before a file is saved or after a file is closed). To disable this feature, uncheck *Enable triggering of macros on Komodo events*.

# Configuring Proxies

By default, Komodo's language–specific Help accesses content from the Internet. If your system is behind a proxy, you must configure the proxy's IP address and port number to access this content. If your system is not behind a proxy, select *Direct connection to the Internet* (the default). Otherwise, select *Manual proxy configuration* and configure the following options:

- *HTTP Proxy*: Enter the IP address of your network's proxy server.
- *Port*: Enter the port number on the proxy server that is configured to access the Internet.
- *No Proxy for*: To exclude specific domains from being routed through the proxy, enter the desired domain names in this field. If there are multiple domains to exclude, separate the domain names with a comma.

# Servers Preferences

Use the *Servers* page to configure FTP servers and accounts for remote file access. To access the Servers page, select *Edit/Preferences/Servers*. Note that you can also manually connect to a server when opening or saving remote files.

See Opening Remote Files for information about working with remote files.

If no servers have been previously configured, enter access information as described below and click the *Add* button. If there are prior server configurations, click the *New Server* button to clear the fields. To alter an existing configuration, select the configuration from the drop–down list, make the desired changes, then click the *Update* button. To delete a configuration, select the desired configuration and click the *Delete* button.

- *Remote Accounts*: Previous server configurations can be accessed via this field.
- *Server Type*: The server type is "FTP".
- *Name*: Enter a name for the account. The value in this field is displayed in the "Remote Accounts" drop–down list box, and is used as the Server name in the Remote File dialog box.
- *Hostname*: Enter the name of the FTP server. The name may be in the format "ftp.server.com", or may be the name of a machine within a local domain.
- *Port*: By default, FTP uses port 21.
- *User Name*: If you require an account to use the FTP server, enter the user name in this field. If access to the FTP server is anonymous, enter "anonymous".
- *Password*: If you require an account to use the FTP server, enter the account password in this field. If access to the FTP server is anonymous, the password is usually an email address (such as

"user@host.com").
  - *Default Path*: To specify the directory that displays when you connect to the server, enter the
    path in this field.
  - *Anonymous Login*: If the server allows anonymous login, check this box.


# Shared Support Preferences

Komodo's shared support functionality is used to configure components on one machine and distribute
them for use on other machines. Shared support is implemented via a "Common Data Directory", which
stores the shared components. The following components can be shared:

  - [templates](#)
  - [Shared Toolbox](#)
  - Tcl .tip files ([syntax definition](#) files)
  - .pcx files (checker extension files that define exact syntax information)
  - .pdx files (debugger extension files)
  - preferences

To configure shared support, select *Edit/Preferences/Shared Support*.

To access shared components, Komodo users must have "read" access rights to shared files in both the
Common Data Directory and the [Shared Toolbox](#) (if the directory is not the same as the Common Data
Directory). To alter shared components, users must also have "write" rights.

To Configure the Common Data Directory:

The Common Data Directory default locations:

  - *Windows*: `C:\Documents and Settings\All Users\Application
    Data\ActiveState\Komodo\3.0`
  - *Unix*: `/etc/komodo/3.0`

To specify a custom location for the Common Data Directory:

  1. On the *Edit* menu, select *Preferences/Shared Support*.
  2. Click *Use custom Common Data Directory location*.
  3. Click *Choose* to select a new location.
  4. Click *OK*.

## Sharing .tip, .pcx and .pdx Files

Through Shared Support, .tip files (which provide syntax checking for PHP and Tcl) can be made available site–wide. All .tip files should be stored along with the default .tip information in the *tcl* subdirectory of the Common Data Directory.

The other file types that can be shared are .pcx files, which can be used to extend the command information supported by the TDK Checker and Komodo Tcl linter, and .pdx files, which are debugger extension files that define debugging functions, such as spawnpoints. Like .tip files, .pcx and .pdx files are stored in the *tcl* subdirectory of the Common Data Directory.

## Sharing Preferences

Shared preferences are used to set a default preference configuration that is shared between multiple Komodo users. An organization or user group can specify defaults like the language type for new files, default tab widths, and other Komodo settings.

There are three levels of preference recognition in Komodo:

1. user preferences
2. shared preferences (common)
3. default preferences (factory)

In a shared configuration, user preferences always override the shared preferences. Shared preferences always override the default preferences.

To configure shared preferences, set the desired preferences in one instance of Komodo. (This sets user preferences for that Komodo installation.) Then, edit the *prefs.xml* file that stores the preferences (located by default in `C:\Program Files\ActiveState Komodo` on Windows, and `etc/komodo/` on Unix). (Make a backup copy of *prefs.xml* before editing it.) In *prefs.xml*, make the following changes:

- Change the value of `commonDataDirMethod` to `custom`.
- Change the value of `customCommonDataDir` to the path to the Common Data Directory.

Copy *prefs.xml* to the Common Data Directory. When other Komodo sessions (configured to use the same Common Data Directory) are started, the preferences in the Common Data Directory are used.

Because user preferences override both default and shared preferences, ensure that user preferences are not configured for items defined in the shared preferences. For example, if the shared preference contains a tab size definition, and a user's personal preference contains a tab size definition, the user's preference is used, not the shared preference.

# Source Code Control Preferences

Use Komodo's Source Code Control integration to perform the most common SCC repository tasks from within Komodo, including checking files out, comparing them to the repository version, and checking files back in. See Source Code Control for information about using SCC functions within Komodo.

- *Show SCC Output Tab on Commands*: Select the desired action from the drop–down list to specify whether the SCC tab is displayed when SCC commands produce output.
- *Method used to display 'diff' output*: Specify whether the output from the SCC diff command should be displayed in a separate window, or within a new tab in the Komodo Editor Pane.

## CVS Integration

Configure these options to use CVS source code control integration.

- *CVS Integration*: Check this box if you are using a CVS source code repository.
- *CVS executable used*: Choose the path to the desired CVS executable file from the drop–down list, or click *Browse* to navigate to the file location.
- *Check for status changes from outside of Komodo*: If this box is checked, Komodo checks to see if the status of files that are open in the editor has changed from the status they had at the last check. Specify the interval at which Komodo should check the file status in the field below.
- *Do recursive status checks*: When checking the CVS status of files in a project, check this box to recurse the directories. If this box is not checked, only the status of files in the current directory are checked.
- *Diff options*: When you use the option *Diff (Compare Files)*, the comparison is performed according to the style specified here. Any CVS diff options may be specified. For a complete list of options, refer to the CVS Manual.
- *Do not warn about CVS external protocols (CVS_RSH) at startup*: If you are using an external protocol (such as RSH) to connect to the CVS repository, check this box if you do not want a warning displayed when you start Komodo.

## Perforce Integration

Configure these options to use Perforce source code control integration.

- *Perforce Integration*: Check this box if using a Perforce source code repository.
- *Perforce executable used*: Use the drop–down list or the *Browse* button to specify the path to the Perforce executable file.
- *Check for status changes from outside of Komodo*: If this box is checked, Komodo checks to see if the status of files that are open in the editor has changed from the status it had at the last check. Specify the interval at which Komodo should check the file status in the field below.

- *Do recursive status checks*: When checking the status of files in a project, check this box to recurse the directories. If this box is not checked, only the status of files in the current directory is checked.
- *Show diff in Komodo*: When you use the option *Diff (Compare Files)*, the comparison is performed according to the style specified here. Refer to the Perforce Manual for a complete description of the options. (Alternatively, on the command line, enter `p4 help diff`.)
- *Use external diff tool*: If you want to use a diff tool other than Perforce, it must be specified in this field. The location of the diff tool must also be included in your system's PATH environment variable.
- *Automatically open files for edit before save*: Select an option from the drop−down list to determine what Komodo does if you attempt to save a file that has not been checked out of Perforce.

# Web and Browser Preferences

- *Web Browser*: Specify the browser that Komodo should launch when a web−based language query or the web browser preview is invoked. Select the desired browser from the list, or use the *Browse* button to navigate to the desired browser. If you do not specify a browser, Komodo uses the system's default browser.
- *Preview in Browser*: Choose the method Komodo uses to preview code in the selected web browser:
    - *Preview in Komodo tab, other tab group*: This option splits the Editor Pane to display the browser preview in a separate pane.
    - *Preview in Komodo tab, same tab group*: This option displays the browser preview in the Editor Pane.
    - *Preview in external browser*: This option opens the default browser (specified in the Web and Browser Preferences drop−down list) in a separate window.

# Windows Integration Preferences

Windows Integration preferences set system−wide file associations on the Windows platform. By configuring file associations, Komodo becomes the default editor for specific file types. When one of these files is invoked (for example, by double−clicking the filename in Windows Explorer), Komodo is automatically launched (if not already running), and the file is loaded in the Editor Pane.

When a file extension is added to the "Edit with Komodo" association, the context menu displayed when the filename is right−clicked in Window Explorer contains an "Edit with Komodo" option.

To configure file associations:

1. Select *Edit/Preferences/Windows Integration*.

2. Click *Configure common associations*. The *Setup Common Komodo File Associations* dialog box opens.
3. Select the file extensions for which Komodo should be the default editor, and the files extensions that should have the "Edit with Komodo" context menu option.

Individual file extensions may be added and deleted via the lists.

If another application overrides the associations configured by Komodo, click *Re−apply settings to system* to reset the Komodo associations.

---

# Feature Showcase

These quick demos highlight a variety of Komodo features that help you write code quickly and accurately. Take a look at some advanced search functions; explore your code using the code analysis tools; create custom workspace components for code reuse and project management.

## Editing

- Preview Cascading Style Sheets
- Reuse a Code Fragment
- Create a Prompting Snippet
- Code Completion Snippet

## Code Analysis

- Code Descriptions in the Code Browser
- Object Browser
- View Construct Scope

## Debugging

- Break on a Variable Value
- Debug an XSLT Program

## Search

- Fast String Finder
- Incremental Search
- Open/Find Toolbar

## Tools

- Google Run Command
- Interactive Shell
- Build a Perl Executable with PerlApp
- Test a Regular Expression

## Project and Workspace

- Create a Custom Toolbar
- Directory Shortcut
- Export/Import Project Package
- Filesystem in Project

- [Custom Keybindings](#)
- [Custom Template in a Project](#)

# Feature Showcase: Fast String Finder

Two keystrokes to find the next occurrence of a string in the current document – two more keystrokes to find all occurrences of the string in the current file's directory. Use the Open/Find Toolbar for fast searches.

Before you start: If necessary, display the Open/Find toolbar (*View*/*Toolbars*/*Open/Find*).
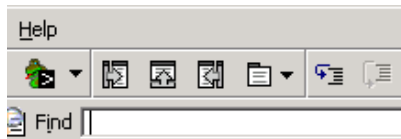


'Alt'+'I' positions the cursor in the *Find* field.





Enter the search string.
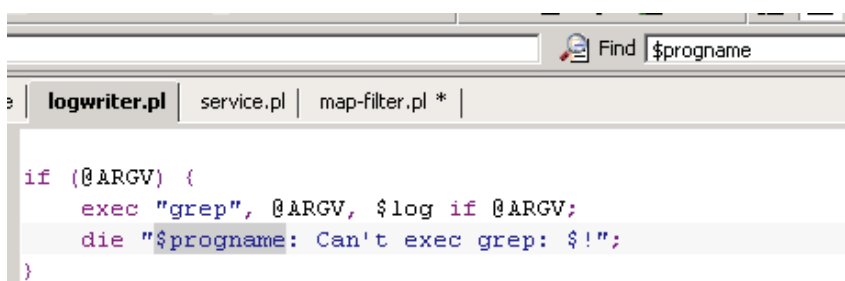




'Enter' searches for the string in the current file.
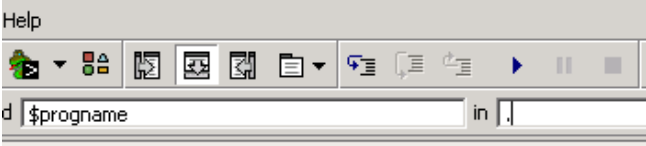
'Tab' positions the cursor in the *Find In* field. Enter a period to search the directory of the current file.

↓



'Enter' displays all matches in the directory of the current file.

# Feature Showcase: Custom Toolbar

Use custom toolbars and icons to personalize the Komodo workspace. This showcase describes how to create a custom toolbar with a custom icon.
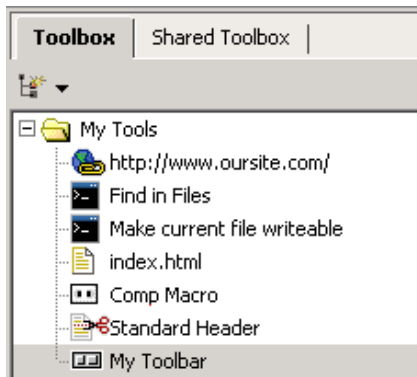


Select *Toolbox|Add|New Custom Toolbar*. Name the toolbar.

↓
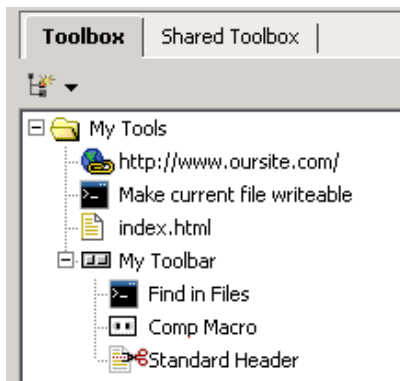


Drag and drop items onto the toolbar.

↓
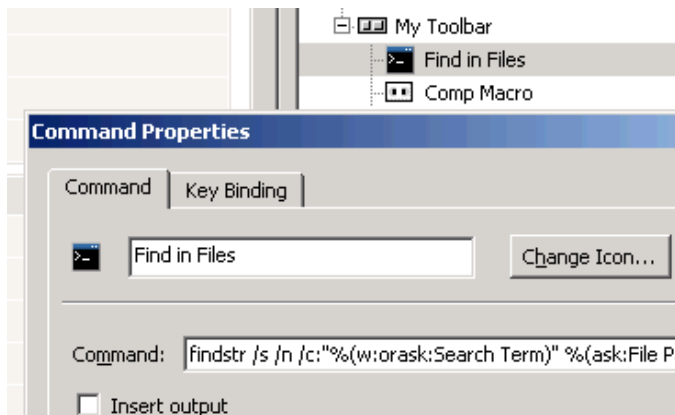
Right−click to access an item's Properties. Click *Change Icon*.

↓
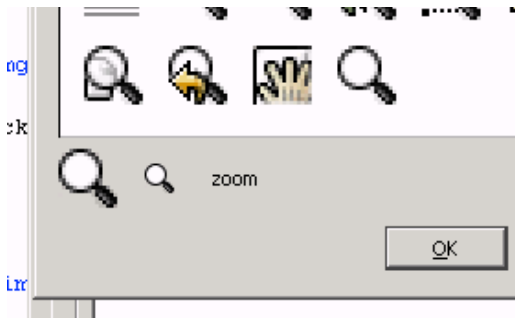


Choose an icon.

↓
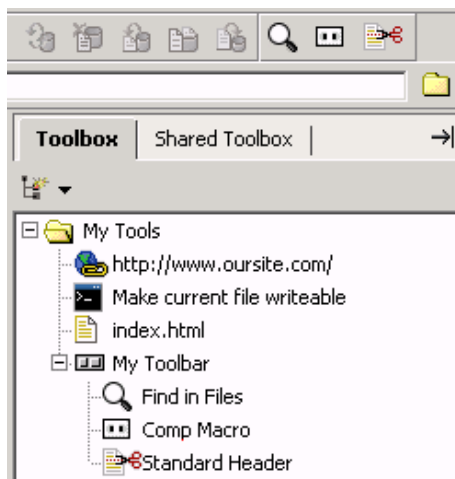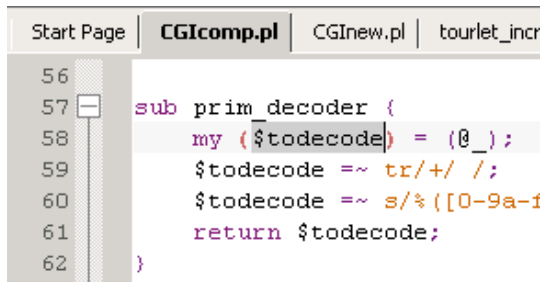
Use the new custom toolbar.
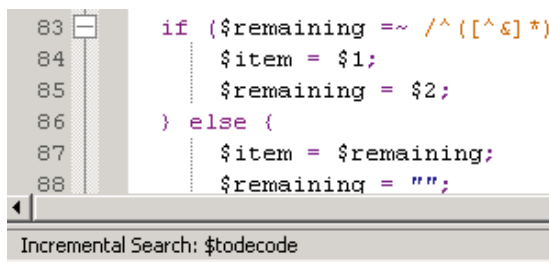
# Feature Showcase: Incremental Search

Use Incremental Search to quickly find all occurrences of a string in all open files. (The key bindings mentioned below are part of the default key binding scheme.)



Select the string.

↓



'Ctrl'+'T' invokes incremental search on the selected string.

↓



Press 'Ctrl'+'T' to find the next occurrence of the string. Continue pressing 'Ctrl'+'T' to find all occurrences in the current file.

↓



Press 'Ctrl'+'Page Down' to change to the next open file. Press 'Ctrl'+'I' to find the string.

# Feature Showcase: Find and Open Files with the Open/Find Toolbar

Use the Open/Find Toolbar to open files via fast keystroke–based filesystem navigation.

Before you start: If necessary, display the Open/Find toolbar (*View*/*Toolbars*/*Open/Find*), and open a file in the Editor Pane.
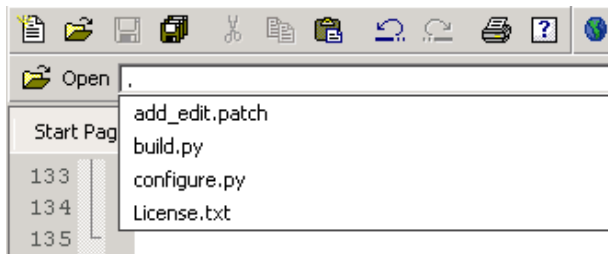
Click in the **Open** field to position the cursor. Enter a period to display the files in the directory where the current file is stored.

↓
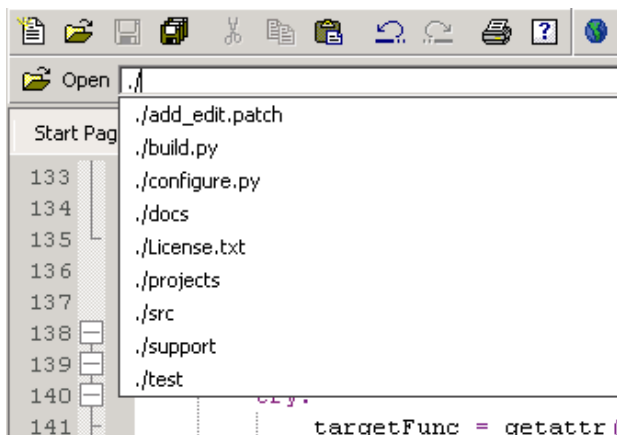
Enter a frontslash to display directories.

↓
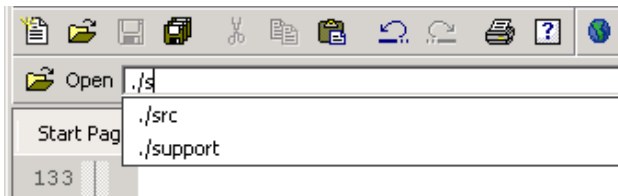
Enter characters to filter the display.

↓
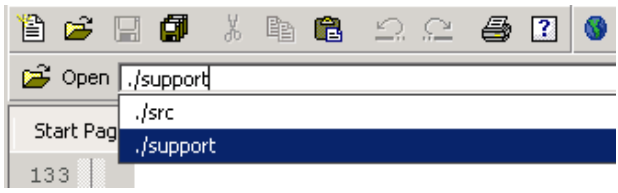


Use the arrow keys to navigate the list.

↓
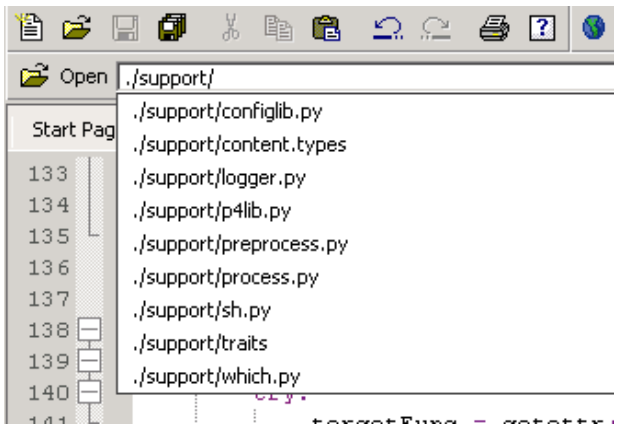


Enter a frontslash after a directory to display the directory contents.

↓
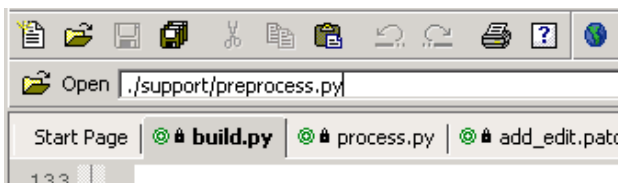
Select the desired file from the list. Press 'Enter' to update the *Open* field, and 'Enter' to open the file in the Editor Pane.

# Feature Showcase: Code Completion Snippet

Create a snippet that contains the structure for a code object. In this example, a common usage of Perl's `if` syntax is stored in a Toolbox snippet.

Before you start: If necessary, display the Toolbox tab (*View/Tabs/Toolbox*).
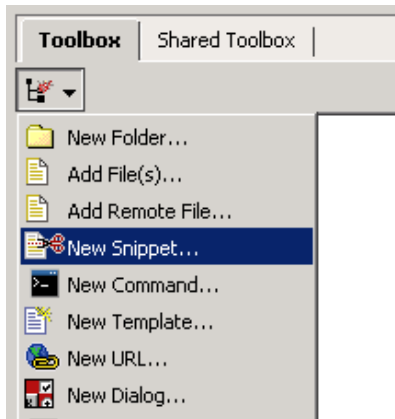


Click in the *Add* button to create a new snippet in the Toolbox.
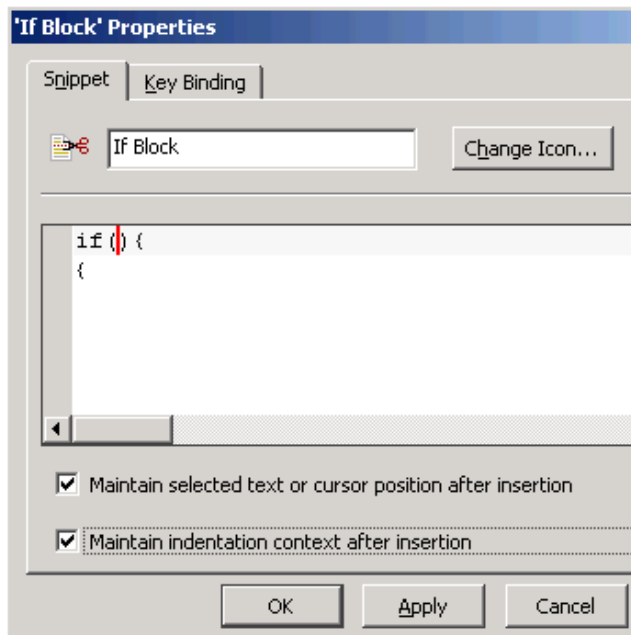


Configure the snippet.

↓

```
Start Page | prynetcfg.pl *
66
67      $def = "" unless defined $def;
68
69      chomp($prompt);
70
71 ⊟    if(){
72 ⊟    {
```

Double−click the snippet to insert the contents at the cursor position in the Editor Pane.

# Feature Showcase: Preview Cascading Style Sheets

Use Komodo's browser preview to view the effects of CSS changes as you edit.

Before you start: Open a CSS file.
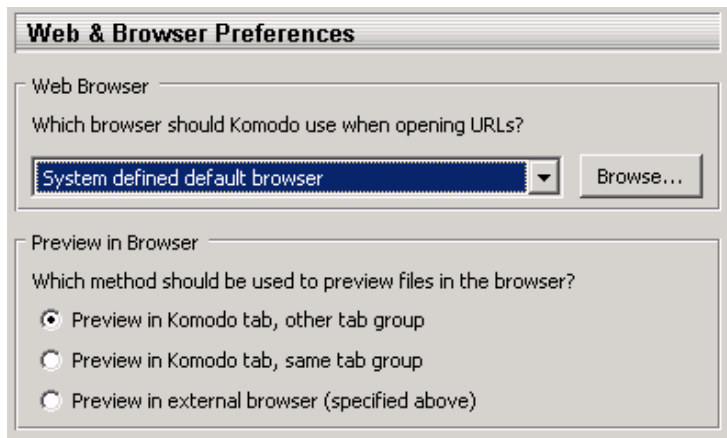


Set the web and browser preference to *Preview in Komodo tab, other tab group*.

↓
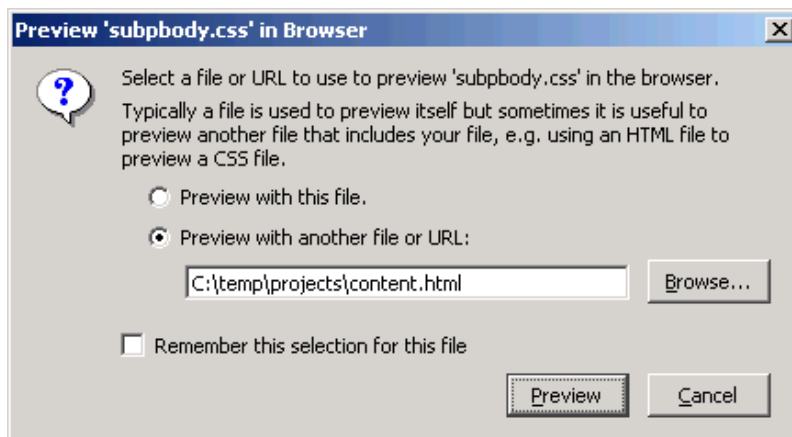


Click the **Web Preview** button and specify a file which uses the CSS file.

↓
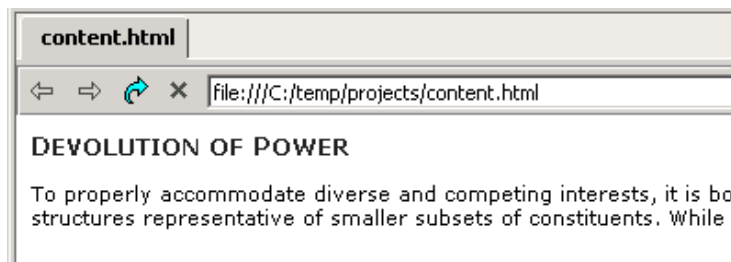
The preview displays the CSS file via the specified HTML file.

↓
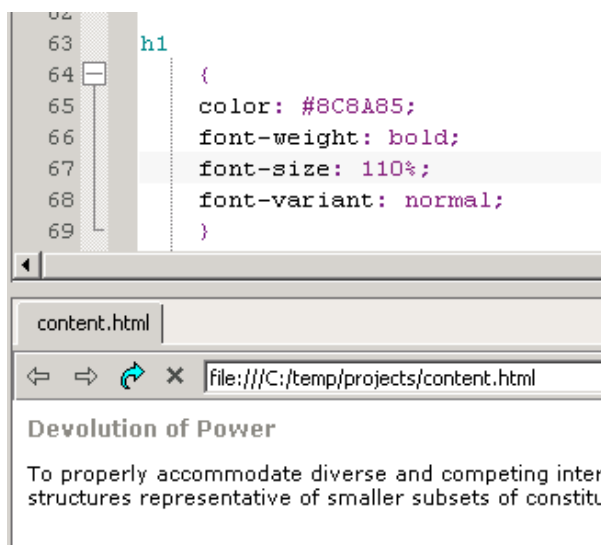


Alter the CSS file. When the changes are saved, the preview is automatically updated.

# Feature Showcase: Snippet that Prompts for Input

Create a snippet that prompts for input. This showcase uses an interpolation shortcut to prompt for a string that is interpolated into the snippet.

Before you start: If necessary, display the Toolbox tab (*View/Tabs/Toolbox*).



Click in the *Add* button to create a new snippet in the Toolbox.

↓
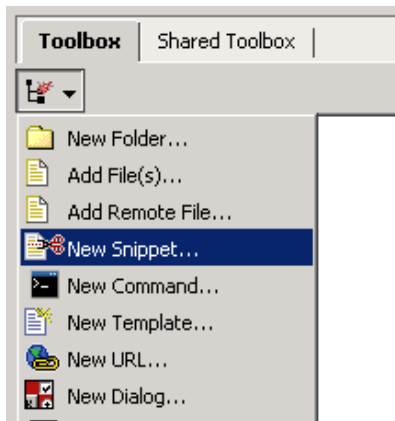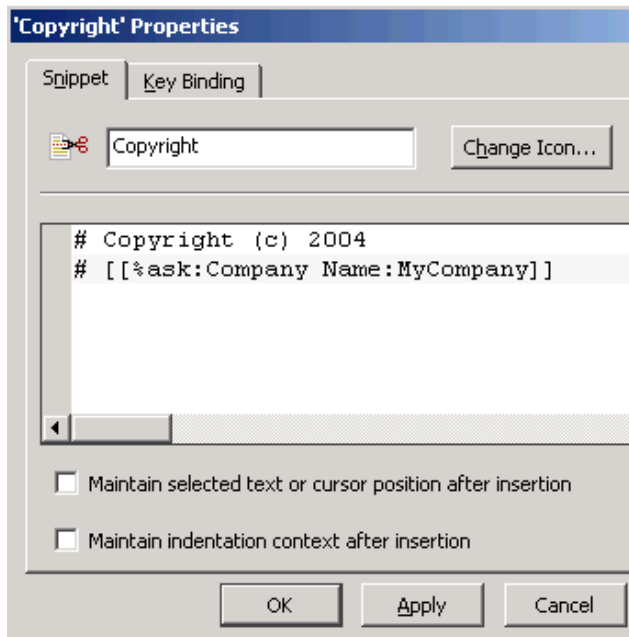
Configure the snippet. The `%ask` segment is an interpolation shortcut that prompts for an entry when the snippet is used.

↓
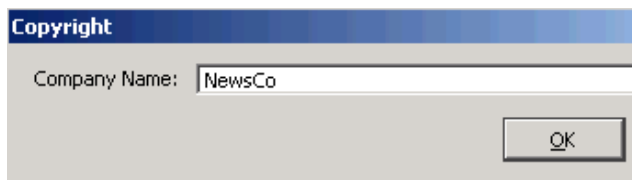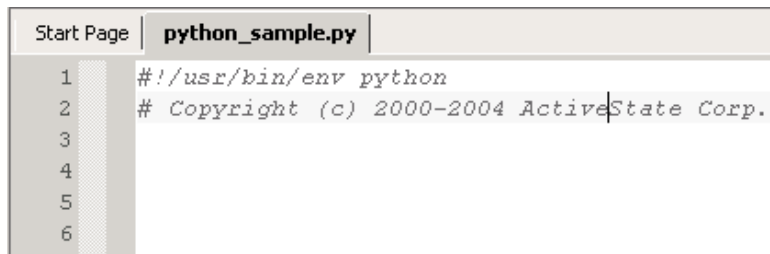


Double−click the snippet. A dialog box is displayed prompts for the "Company Name" value.

↓



The value is interpolated into the comment configured in the snippet.

# Feature Showcase: Google Run Command

Create a run command that launches a Google search on the term under the editing cursor. The "%(browser)" interpolation shortcut loads the browser configured in Komodo's preferences; the "%W" shortcut interpolates the word under the cursor in the editor pane.



In the Editor Pane, place the cursor within the word you want to search.

↓
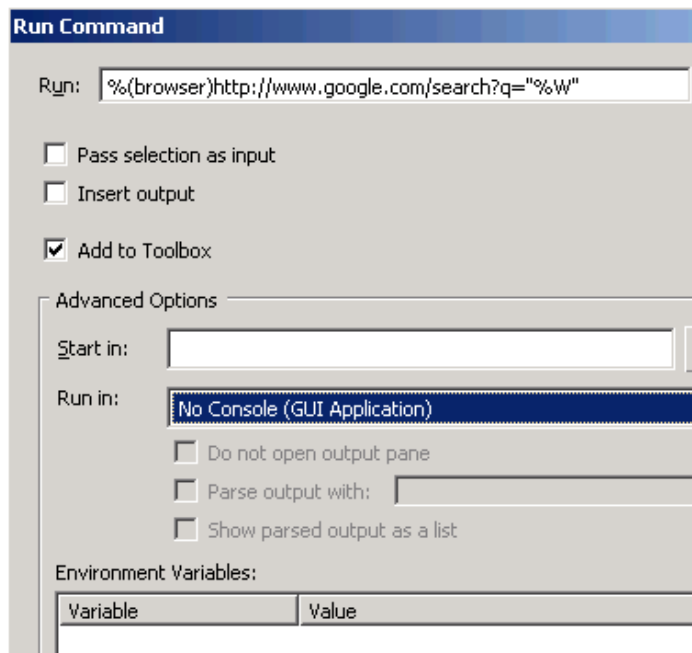


Open the Run Command dialog box (***Tools/Run Command***). Configure as shown.

↓
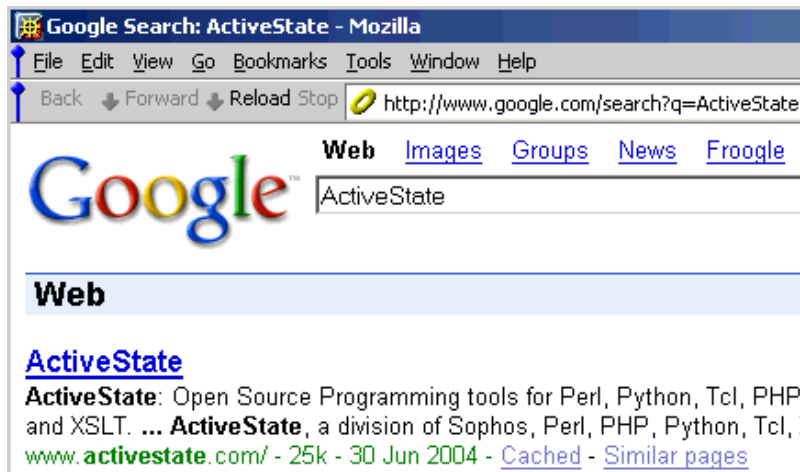
When you click *Run*, the Google search results for the term are displayed in the browser.
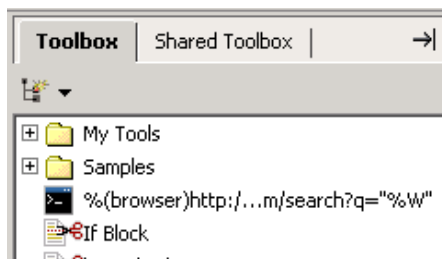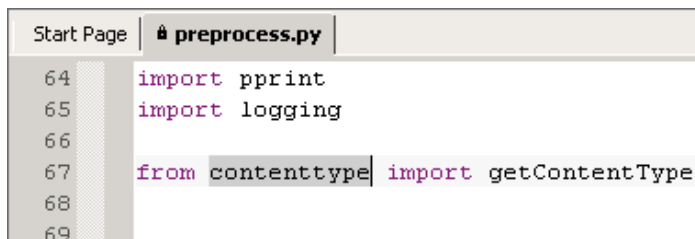




The run command is stored in the Toolbox for re–use.

# Feature Showcase: Using the Interactive Shell

Press 'F12' to switch between the Editor Pane and the interactive shell. This showcase uses the sample program `preprocess.py`, described in the Python tutorial, located by default in *install dir\Komodo x.x\samples\python_tutorials*.
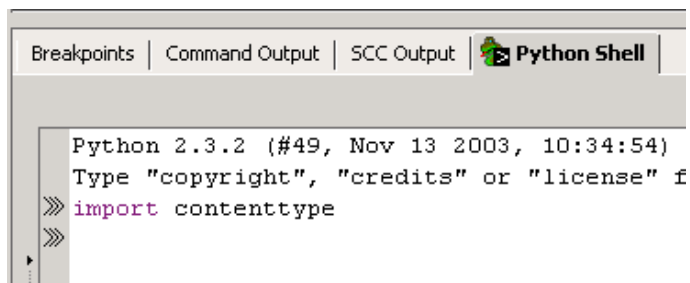
Before you start: Configure the interactive shell preferences to load the Python shell by default.

The key bindings mentioned below are part of the default key binding scheme.
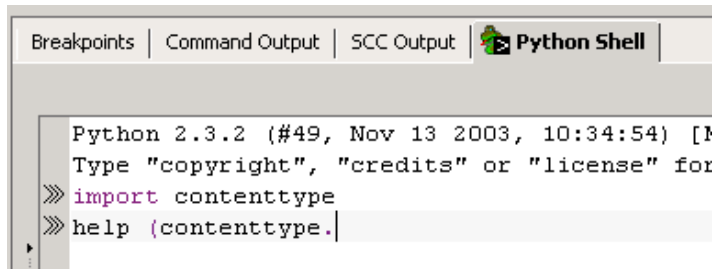


On line 67 of `preprocess.py`, select `contenttype`.
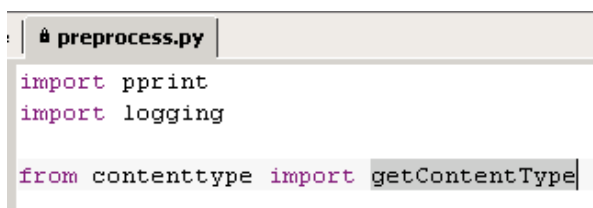




Press 'F12' to open the shell. Enter `import` and press 'Ctrl'+'V'. Press 'Enter' to load the module.
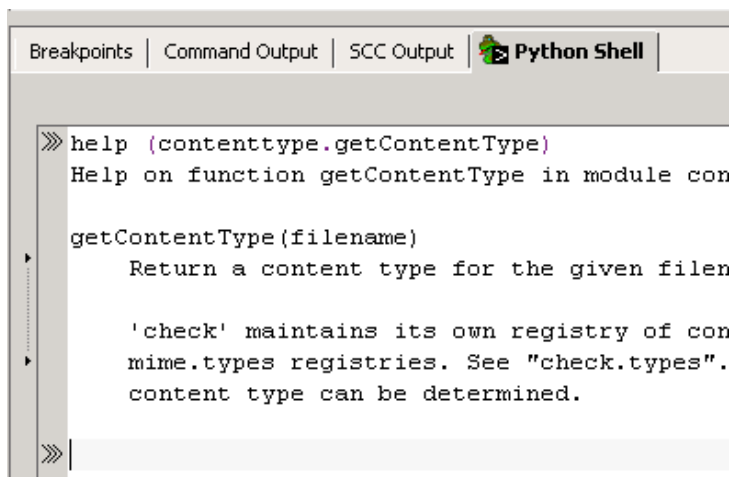
Enter `help (` and press 'Ctrl'+'V' again to paste `contenttype`. Add a period.

↓



Press 'F12' to switch back to the Editor Pane. Select `getContentType`.

↓



Press 'F12' to switch back to the interactive shell. Press 'Ctrl'+'V' to paste `getContentType`, then enter a closing parenthesis and press 'Enter'.

# Feature Showcase: Store a Filesystem Layout in a Project

To mirror a filesystem structure within a project (or within a folder in a project or the toolbox), use the Import from Filesystem function. Folders will be created for each directory, containing the same files as on the filesystem.
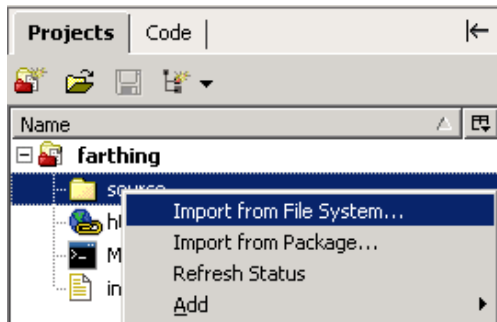


Right–click a folder in a project and select *Import from File System*.

↓
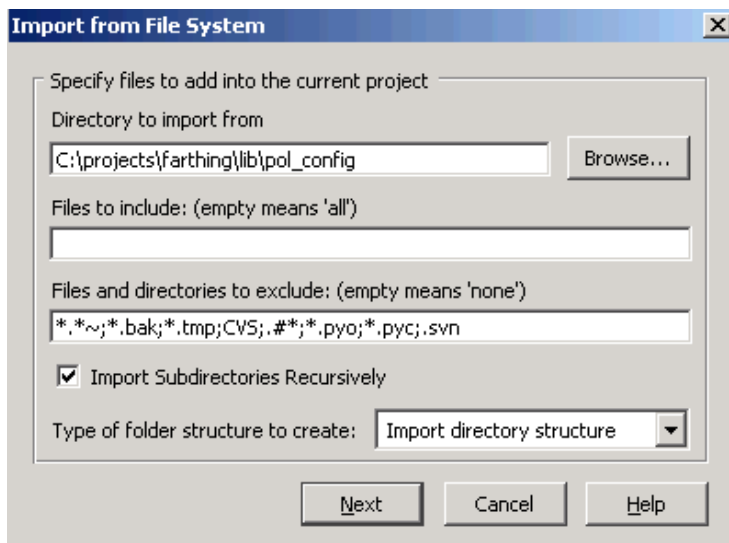


Select the directory, specify inclusion and exclusion parameters, and select *Import directory structure* as the folder structure.

↓
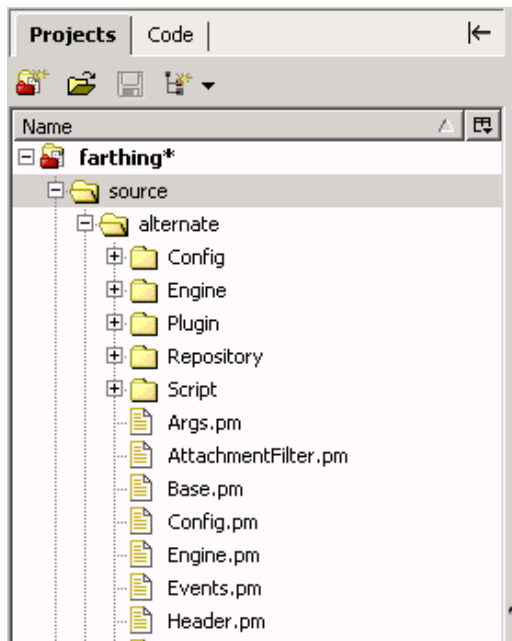
The imported file system has the same structure as the disk.

# Feature Showcase: Using Conditional Breakpoints

Conditional breakpoints are used to pause the debugger when specific events occur, such as when a variable equals a certain value, an exception occurs, or a function completes execution. This showcase uses a variable in the Perl sample program; on the Start Page, select **Open Sample Project**, then double−click *perl_sample.pl* in the Project Manager.
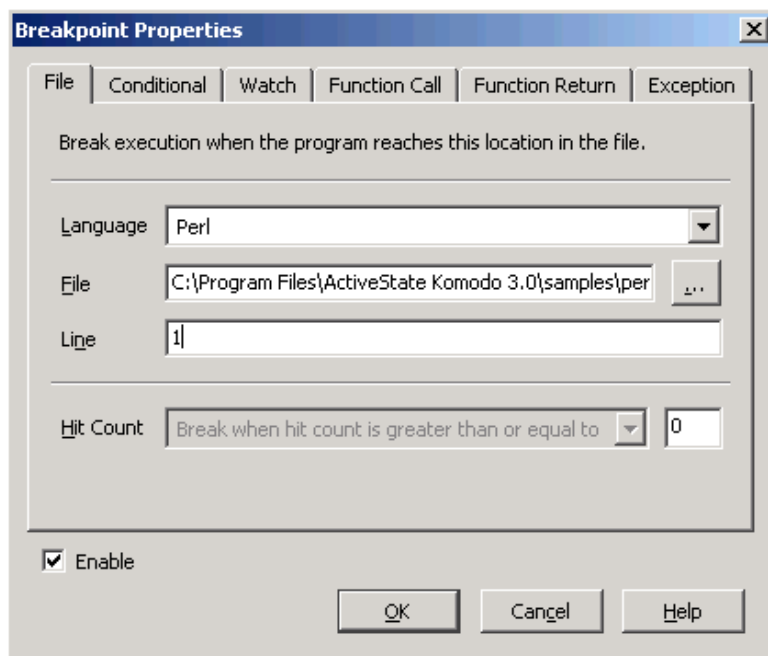


Invoke the Breakpoint Properties dialog box (**Debug/Add\Edit Breakpoint**).

↓
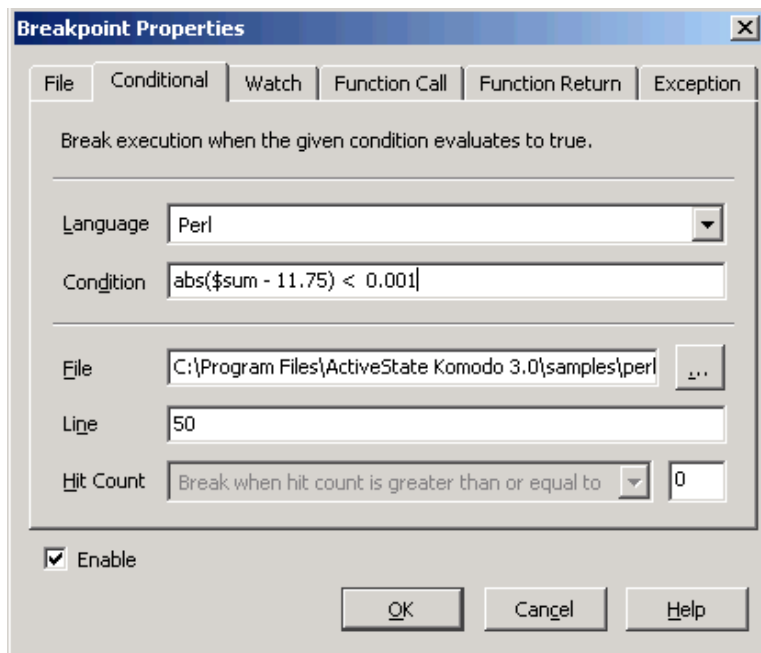
On the *Conditional* tab, configure a breakpoint as shown. The break will occur on line 50 when the `$sum` variable is equal to $11.75.

↓
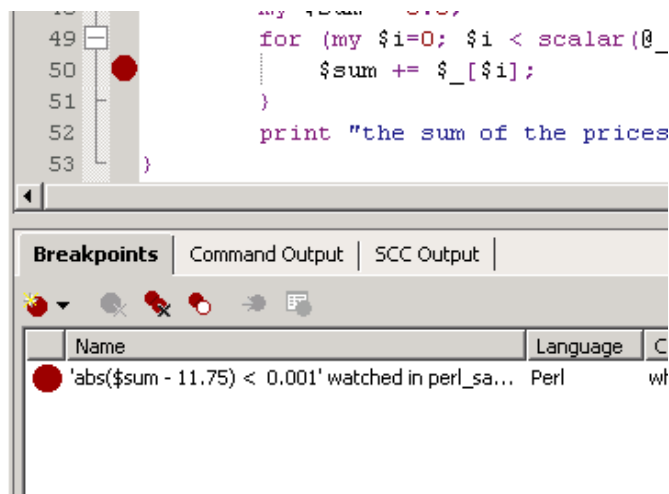


The breakpoint is displayed on the margin of the program file, and on the *Breakpoints* tab.

↓

```
49  ⊟              for (my $i=0; $i < scalar(@_); $i++) {
50  ◆                  $sum += $_[$i];
51  ⊢              }
52                 print "the sum of the prices is $sum\n
```

| Breakpoints | Command Output | SCC Output | **II Debug: perl_sample.pl** |
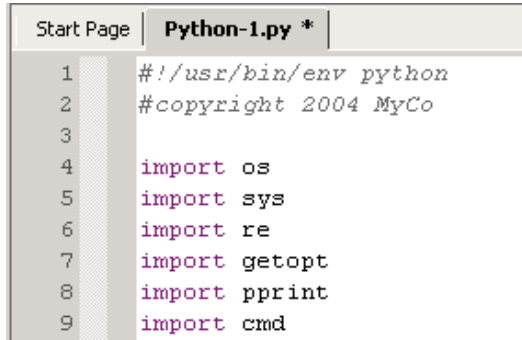
Break at C:\Program Files\ActiveState Komodo 3.0\samples\perl_sample.pl, line 50.

| Name | Type | Value |
|------|------|-------|
| $_ | string | |
| ⊞ @_ | array | |
| $_[$i] | int | 7 |
| $i | int | 2 |
| $sum | float | 11.75 |

Run the debugger. Execution pauses on line 50, when the $sum variable is equal to $11.75.

# Feature Showcase: Store a Custom Template in a Project

Create a custom template for creating new files with custom elements. Store the template in a project for re−use.



Create a file with the template contents.

↓
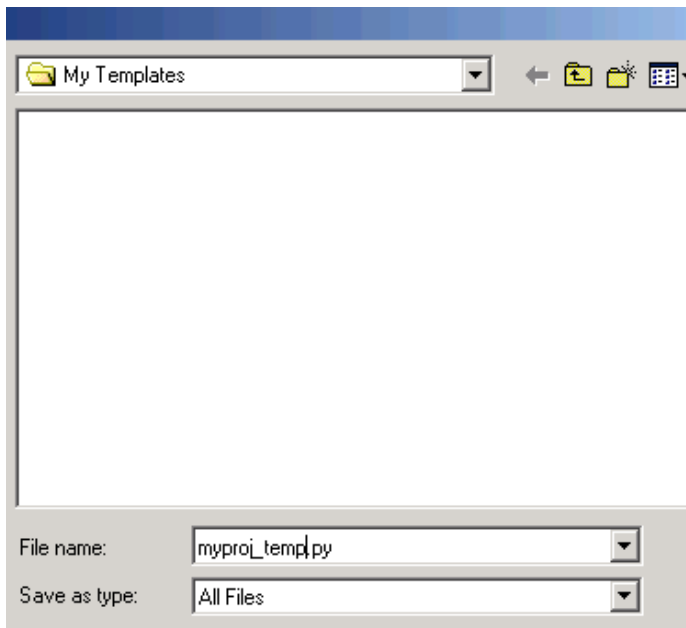


Select *File/Save as Template*. Close the file in the editor.

↓
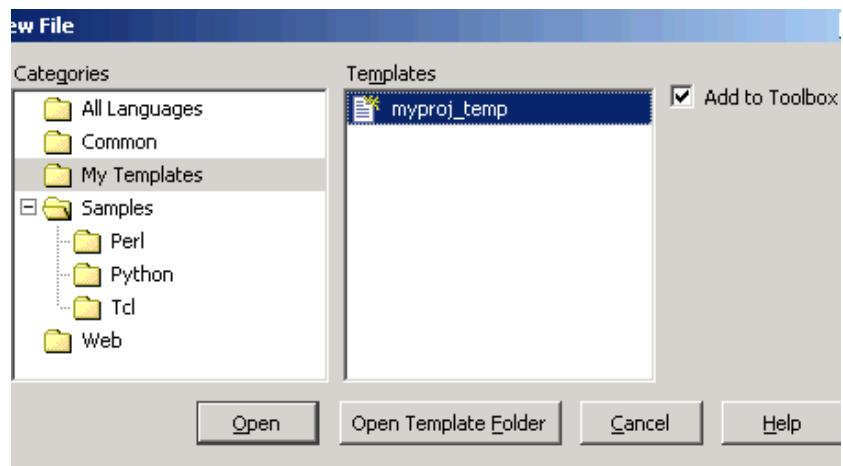
Select *File*/*New*/*New File*. Select the template file and check *Add to Toolbox*.

↓
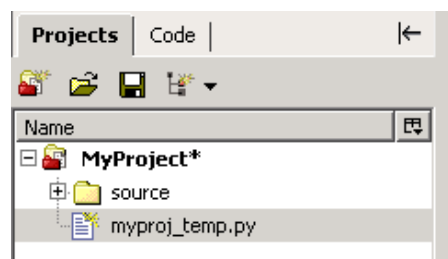


Drag and drop the template file from the Toolbox to a project.

# Feature Showcase: Build a Perl Executable

Building a stand−alone executable from a Perl script in Komodo requires ActiveState's Perl Dev Kit (PDK). The PerlApp packaging tool is run from within Komodo using the *Build Standalone Application* option in the *Tools* menu.

Before you start: Open the *perl_sample.pl* file contained in the sample project.
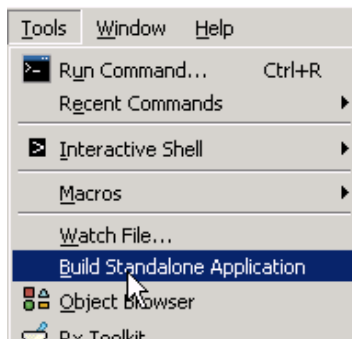


From the *Tools* menu, click Build Standalone Application.

↓

In the Build Standalone Perl Application dialog box, accept the defaults and click ***Build***.

↓

```
Breakpoints | Command Output | SCC Output |
'C:\Perl\bin\perlapp.exe --script perl_sample.pl --exe perl_sample.exe --freestanding --fo

+++ C:\Perl\lib\auto\File\Glob\autosplit.ix
+++ C:\Perl\lib\strict.pm
+++ C:\Perl\lib\vars.pm
+++ C:\Perl\lib\warnings.pm
+++ C:\Perl\lib\warnings\register.pm
+++ perl56.dll (C:\Perl\bin\perl56.dll)

Please run `perlapp.exe --help DIAGNOSTICS` for more
about the generated error and warning messages.

Created 'perl_sample.exe'

Ready
```
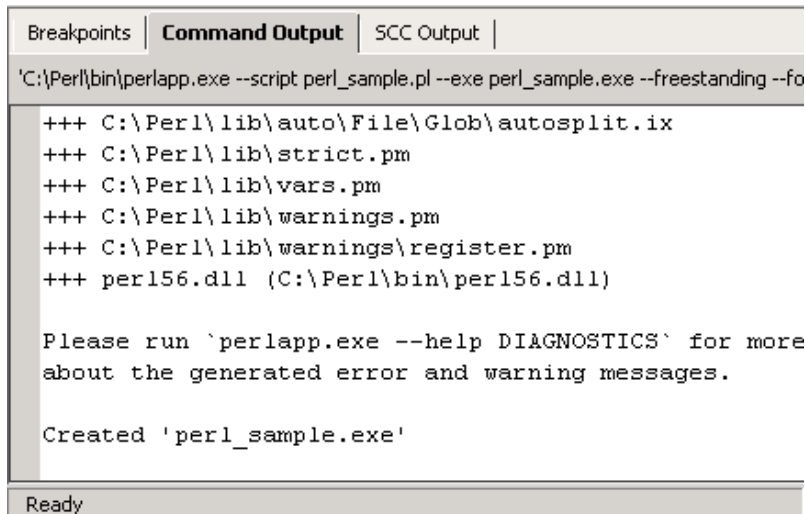
The **Output** tab displays the results of the build process.

↓



```
dlg_tcl_check.ui        3 KB   Komodo GUI
dlg_tcl_check_ui.tcl    4 KB   TCL File
HTMLSnippetHelp.txt     2 KB   Text Docum
javascript_sample.js    2 KB   JScript Scrip
perl_sample.exe       429 KB   Application
perl_sample.pl          2 KB   Perl File
php_sample.php    Type: Application    PHP File
python_sample.    Size: 428 KB         Python File
```
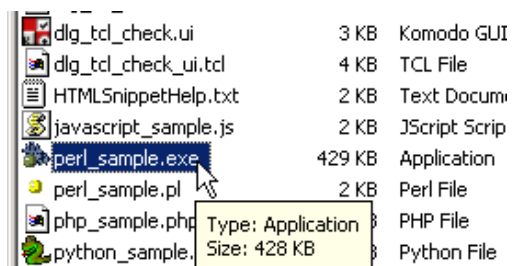
The *perl_sample* executable created is a console application. Double−clicking the executable opens a console window, displays the script output, and closes the window immediately upon completion. To view the script output, run the executable from the command line.

↓

Command string (executed in the 'C

C:\Perl\bin\perlapp.exe --script per
--freestanding --verbose

Add to Toolbox    Build

Status: Build was Successful.
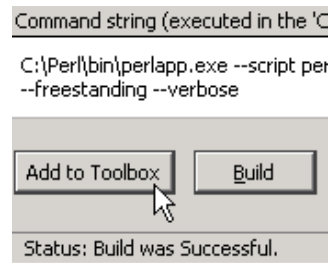
The build options set in the Build Standalone Perl Application dialog box can be saved as a ***Run Command*** in the ***Toolbox***.

# Feature Showcase: Shortcut to Commonly Used Directory

Komodo's Open Shortcut is a fast method for locating files in deeply nested directories. Create a shortcut in the Toolbox or Project Manager that opens the desired directory.
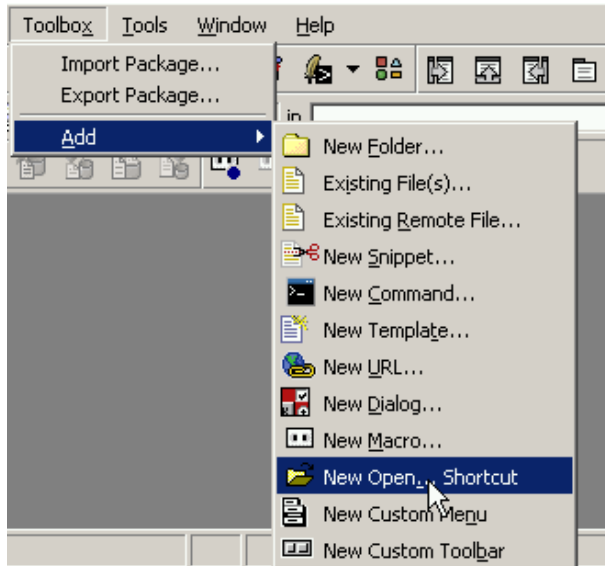


Select *Toolbox|Add|New Open...Shortcut*.

↓
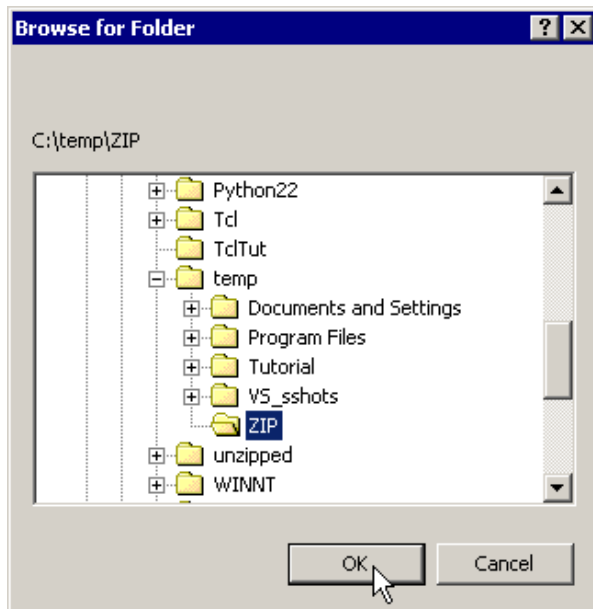
Select a directory, and click OK.

↓
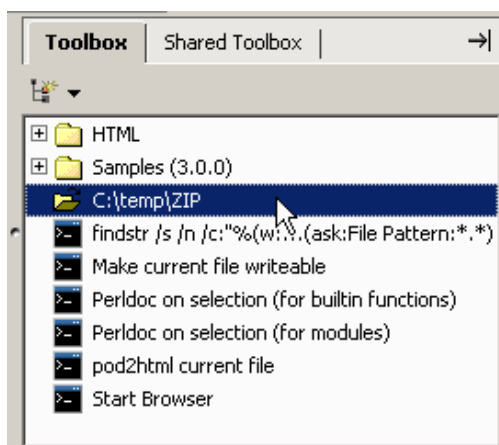


Double−click the shortcut in the Toolbox. the Open File dialog box will open to the specified directory.

# Feature Showcase: Reuse Code Fragments

Create snippet from a code fragment. Store it in the Toolbox for reuse.



In the Editor Pane, select the code.
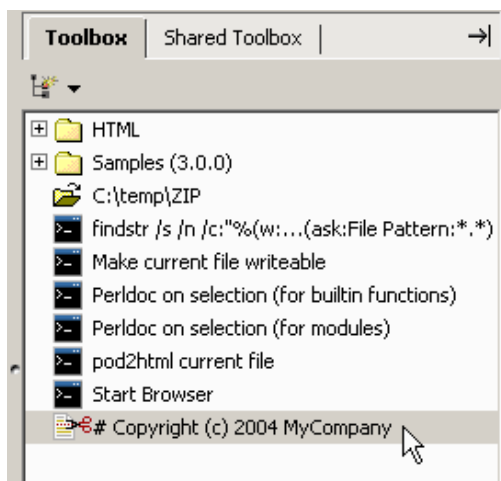




Drag and drop the selected code onto the **Toolbox** tab.

# Feature Showcase: View Code Descriptions in the Code Browser

Use the Code Browser to view the help documentation embedded in source files.

Before you start: Open the *preprocess.kpf* project, located under *<komodo−installdir>\samples\python_tutorials\*
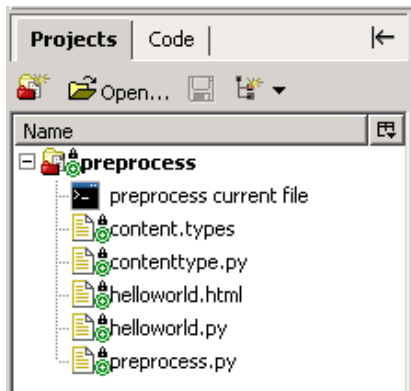


Open the `preprocess.kpf` project. View the project in the ***Projects*** tab.
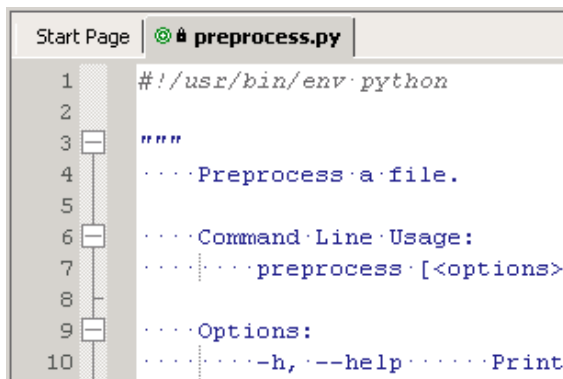
↓



Double−click the ***preprocess.py*** file. The file opens in the Editor Pane in a ***preprocess.py*** tab.

↓

Click the *Code* tab.

↓
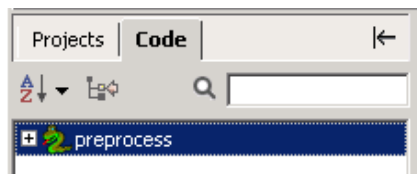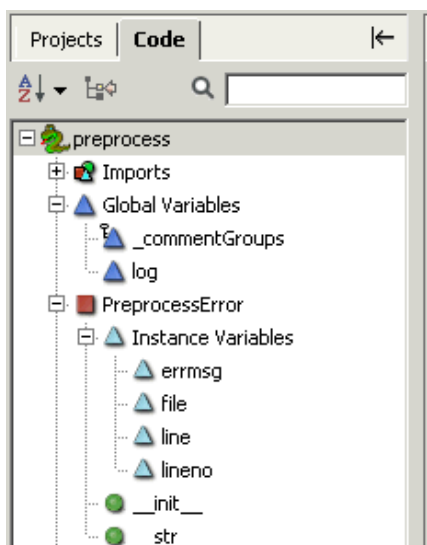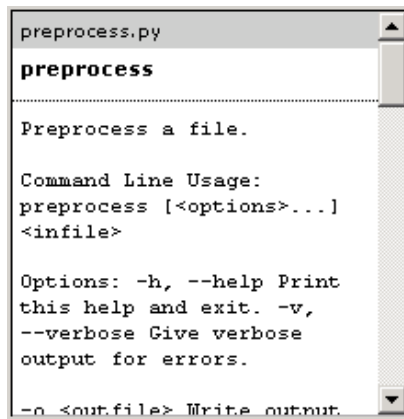


Click the '+' to expand the *preprocess* tree hierarchy. The tree contains all code constructs (variables, methods, imports, etc.) in the *preprocess.py* program.

↓



Click the *Show/Hide Description* button.

↓

```
preprocess.py
```

**preprocess**

```
Preprocess a file.

Command Line Usage:
preprocess [<options>...]
<infile>

Options: -h, --help Print
this help and exit. -v,
--verbose Give verbose
output for errors.

-o <outfile> Write output.
```

View the *Code Description* pane to read the help documentation embedded in the *preprocess.py* source file.

# Feature Showcase: View the Scope of a Code Construct

Use the Scope Indicator to locate the current scope of a code construct (variable, namespace, method, etc.) within the Code Browser's tree hierarchy.

Before you start: Open the *preprocess.kpf* project, located under *<komodo−installdir>\samples\python_tutorials\*. Be sure that line numbers are enabled in Komodo (***View/View Line Numbers***).



Open the `preprocess.kpf` project. View the project in the ***Projects*** tab.

↓



Double−click the ***preprocess.py*** file. The file opens in the Editor Pane in a ***preprocess.py*** tab.

↓

Click the *Code* tab.

↓
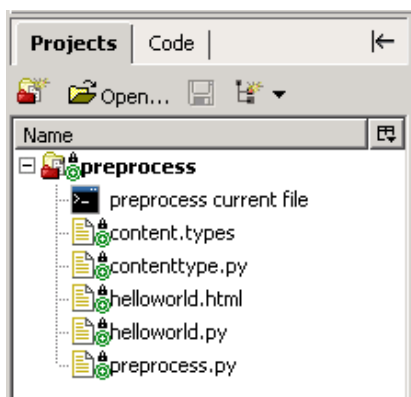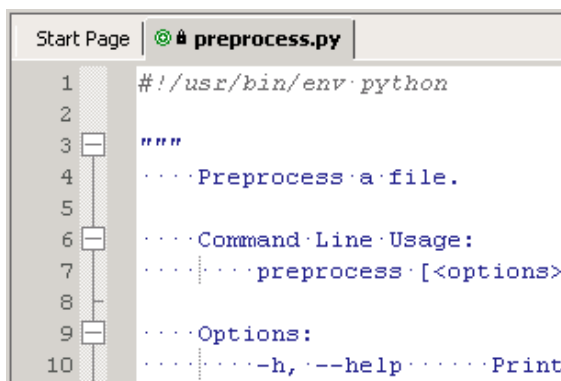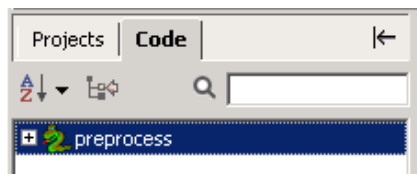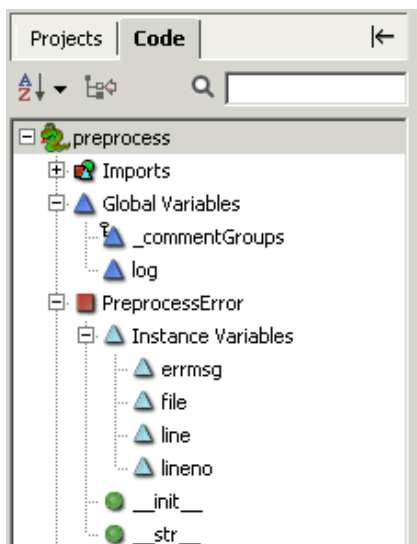


Click the '+' to expand the preprocess tree hierarchy. The tree contains all code constructs (variables, methods, imports, etc.) in the *preprocess.py* program.

↓

```
128
129 ⊟    def preprocess(infile, outfile=sys.stdout, defines={}):
130         ····"""Preprocess the given file.
```

Scoll down to line 129 in the Editor Pane.

↓

```
129 ⊟    def preprocess(infile, outfile=sys.stdout, defines={}):
130         ····"""Preprocess the given file.
```

Use the cursor to highlight the preprocess variable.

↓
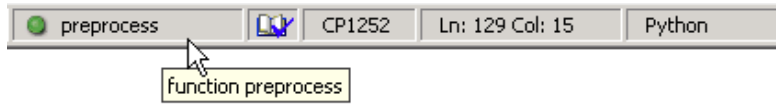


View the Status Bar. Notice the Scope Indicator displays the `preprocess` variable name and type.

↓
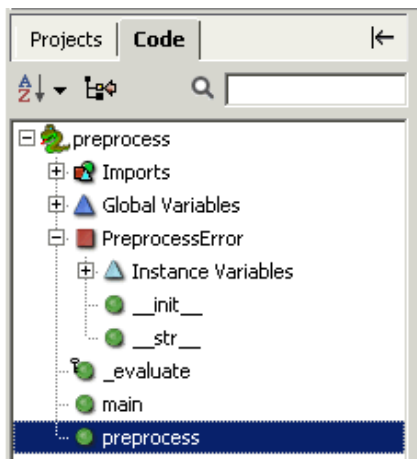


Double−click the ***preprocess*** variable on the Scope Indicator. The Code Browser displays `preprocess` in the tree hierarchy. The current scope of the `preprocess` variable is located.

# Feature Showcase: Find Code Constructs

Use the [Object Browser](#) to search for code constructs (variable, namespace, method, etc.), and locate all instances of that construct in an open project.

Before you start: Open the *preprocess.kpf* project, located under *<komodo−installdir>\samples\python_tutorials\*. Be sure that line numbers are enabled in Komodo (***View/View Line Numbers***).
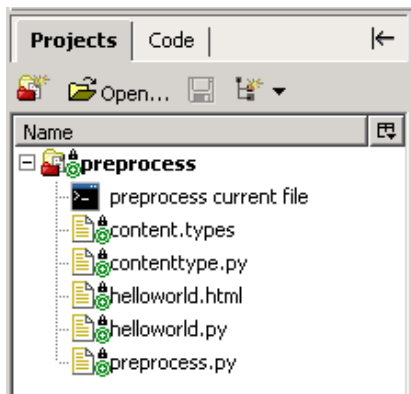


Open the `preprocess.kpf` project. View the project in the ***Projects*** tab.

↓



From the ***Tools*** menu, select ***Object Browser***.

↓
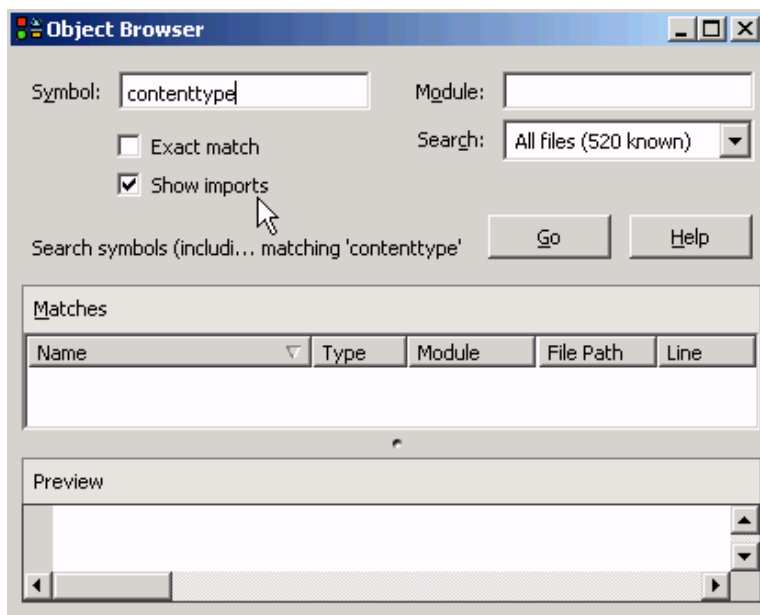
In the *Symbol* text box, enter `contenttype`. Select *Show imports*. Click *Go*.

↓
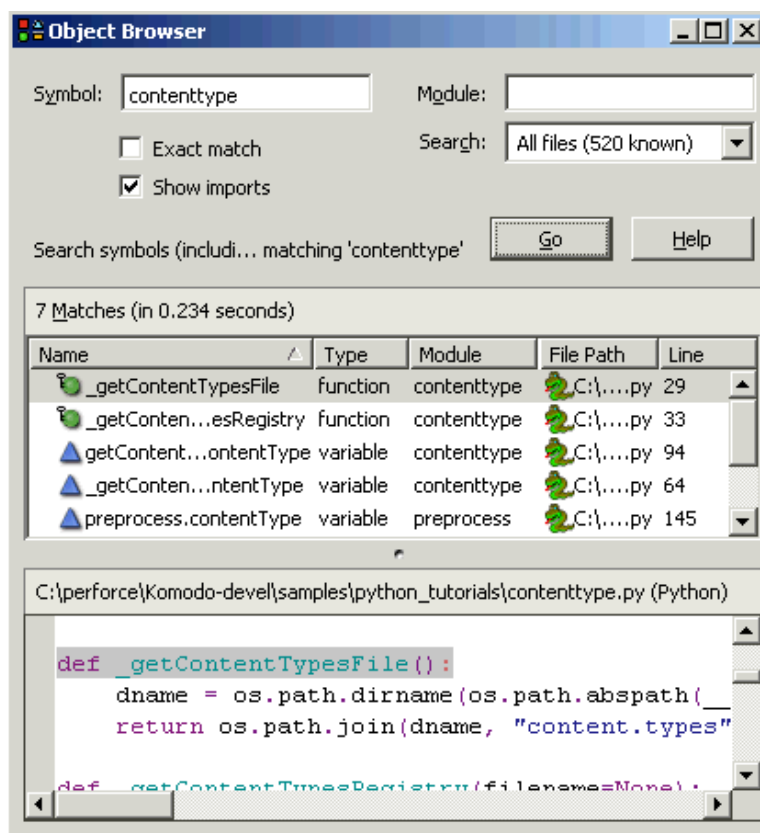
The **Object Browser** searches the Code Intelligence database and finds all symbols (including imports) matching `contenttype`.

↓



The **Matches** pane displays a tree of symbol nodes that outline the general program structure of found search criteria.

↓



In the **Matches** pane, scroll down to the `getContentType` node. Click the **getContentType** node. The **Preview** pane displays the file where `getContentType` is declared.

↓
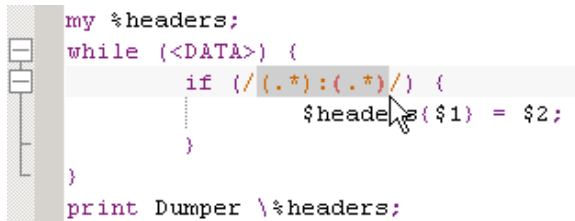


In the **Matches** pane, double−click the **getContentType** node. The file *contenttype.py* opens in the Editor

Pane at the position (line 85) where *getContentType* is declared.

# Feature Showcase: Test a Regular Expression with the Rx Toolkit

Use the **Rx Toolkit** to create, analyse and debug regular expressions in a program.

Before you start: Open the *rx_sample.pl* file contained in the sample project.

```
my %headers;
while (<DATA>) {
        if (/(.*):(.*)/) {
                $headers{$1} = $2;
        }
}
print Dumper \%headers;
```

From the `while (<DATA>)` block, select the regular expression between the "/" characters.

↓

```
Tools   Window   He
```

In the **Toolbar**, click the **Regular Expression Toolkit** button.

↓
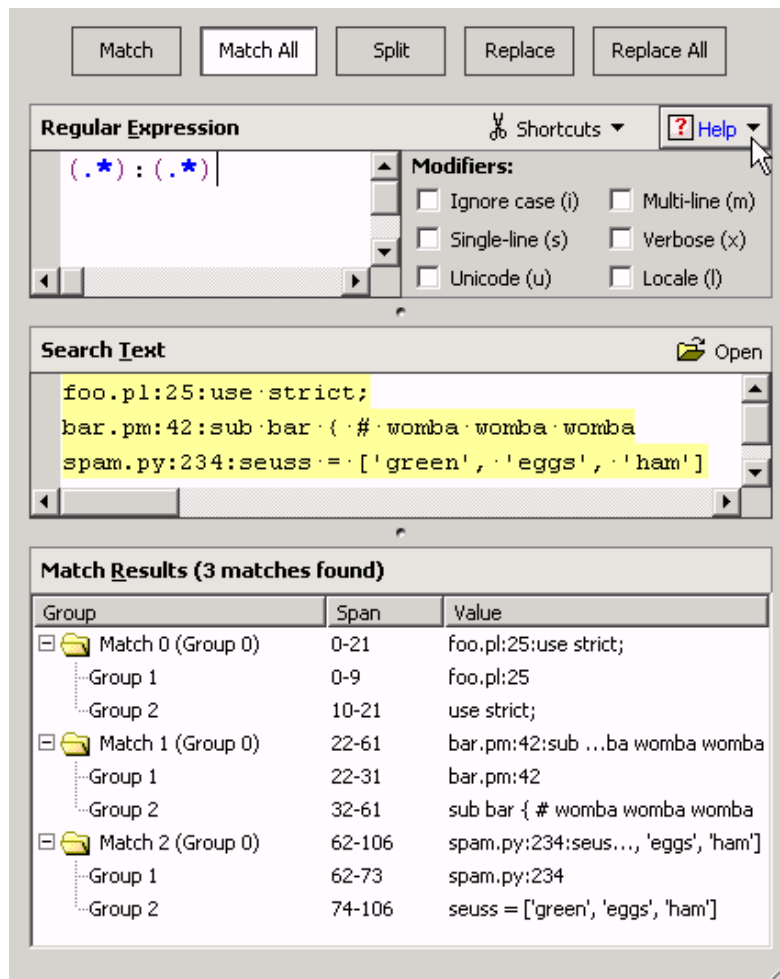
Clear the previous contents by clicking **Help/Load Sample Regex and Search Text**. Copy the selected expression to the **Regular Expression** field.

↓
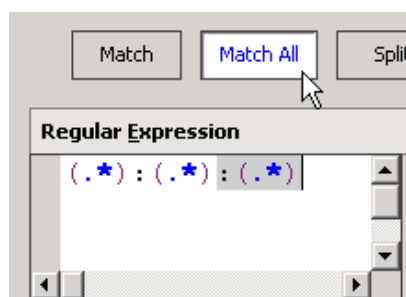


In the **Regular Expression** pane, add an additional "**: ( . * )**" and click **Match All**.

↓



| Group | Span | Value |
| --- | --- | --- |
| ⊟ 📂 Match 0 (Group 0) | 0-21 | foo.pl:25:use strict; |
| Group 1 | 0-6 | foo.pl |
| Group 2 | 7-9 | 25 |
| Group 3 | 10-21 | use strict; |
| ⊟ 📂 Match 1 (Group 0) | 22-61 | bar.pm:42:su... womba womba |
| Group 1 | 22-28 | bar.pm |
| Group 2 | 29-31 | 42 |
| Group 3 | 32-61 | sub bar { # w... womba womba |

Match Results (3 matches found)

The *Match Results* pane now displays three groups for each regular expression match.

# Feature Showcase: Assign a Key Binding to a Toolbox Item

Assign a key binding to a frequently used component in the toolbox or a project.
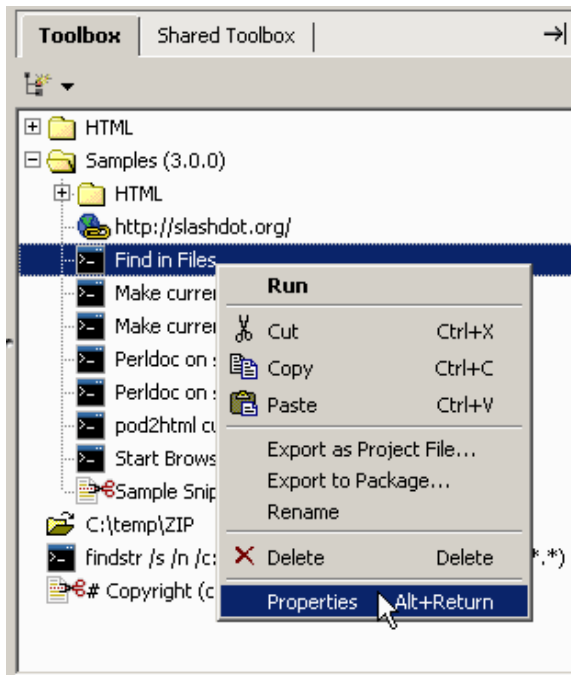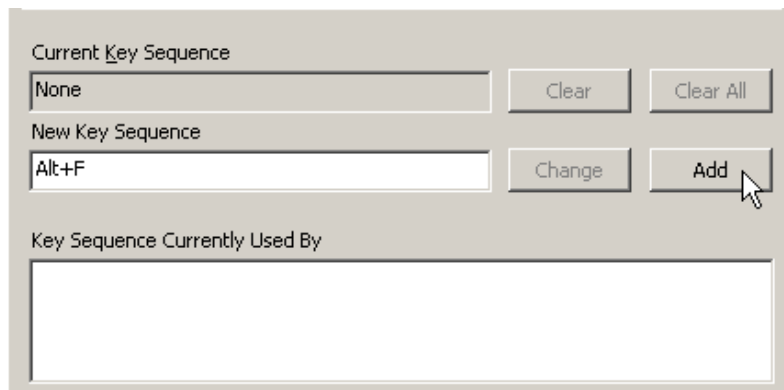


In the Toolbox, right–click the item, and select **Properties**.
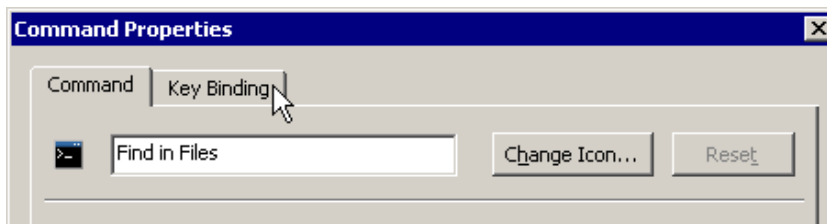
↓



Click the **Key Binding** tab.

↓

In the *New Key Sequence* field, press the desired keys (if sequence already in use, it is indicated in the *Key Sequence Currently Used By* field). Click *Add*, and then click *OK*.

# Feature Showcase: Distributing a Project in a Package

Komodo projects and the components they contain can be exported to a package file for distribution to other Komodo users or for the sake of archiving. Exported packages can be imported by instances of Komodo running on other machines.
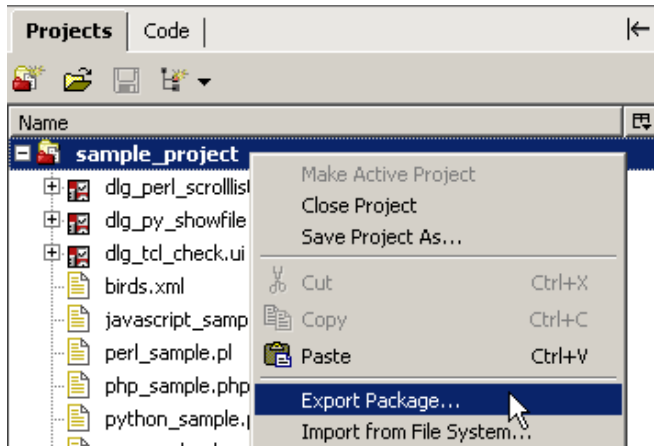


On the *Projects* tab, right–click the project name and select *Export Package*.

↓
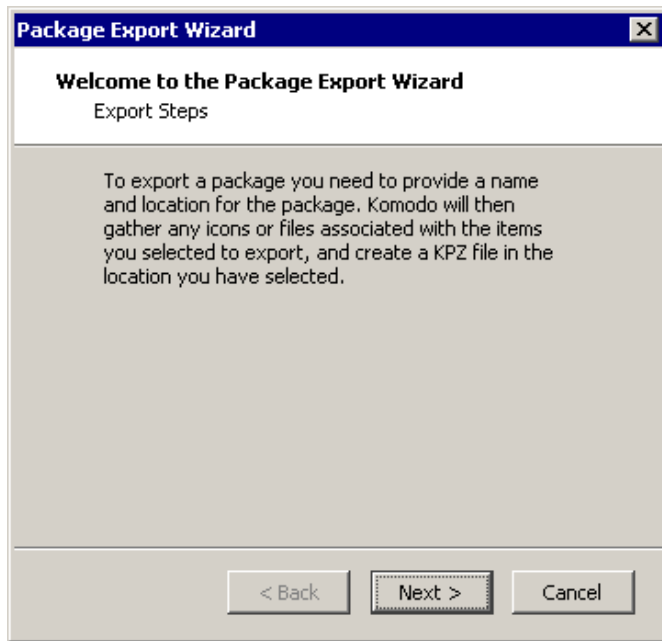
In the Package Export Wizard, click *Next*.

↓
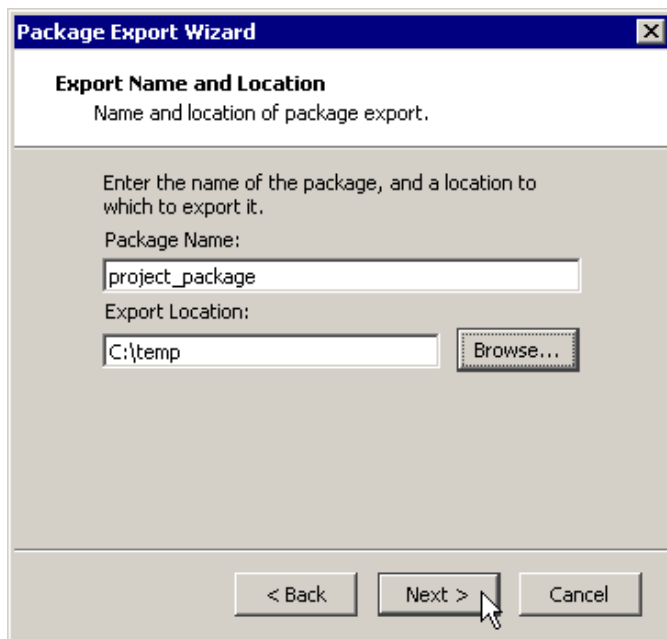


Enter a *Package Name* and an *Export Location*. Click *Next*. Click *Finish*.

↓
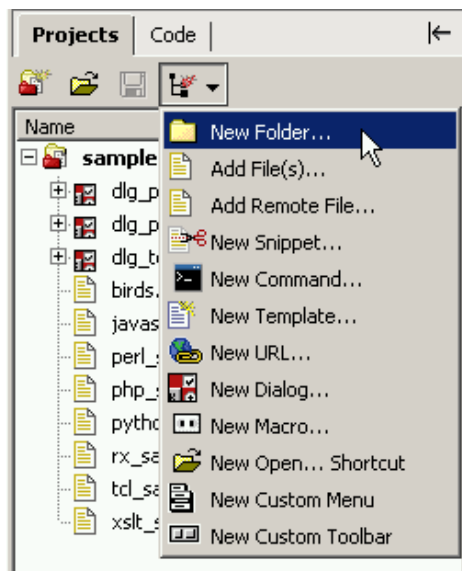
On the *Projects* tab, click the "Add Item" button and select *New Folder*.

↓
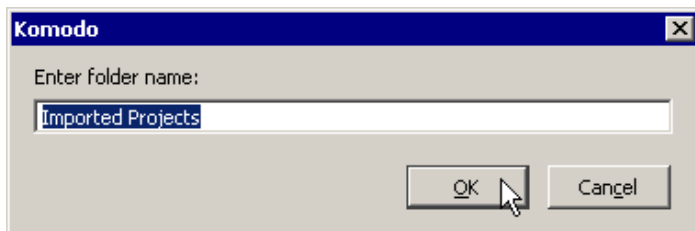


Name the folder "Imported Projects" and click *OK*.

↓
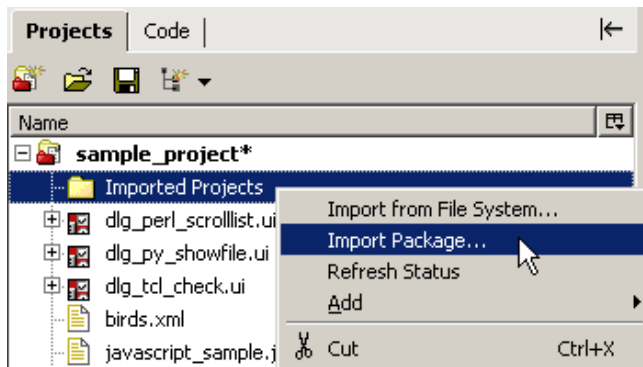
On the *Projects* tab, right–click the *Imported Projects* folder and select *Import Package*.

↓
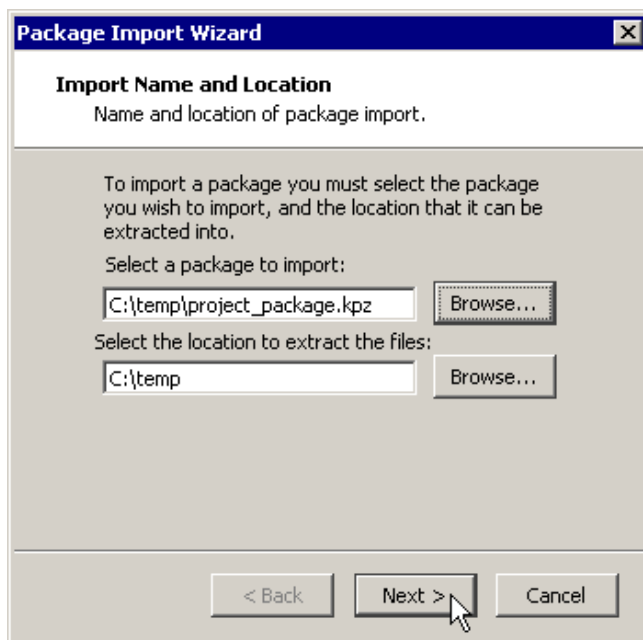


In the Package Import Wizard, enter the package to import, and the location where the files will be extracted. Click *Next*. Click *Finish*.

↓
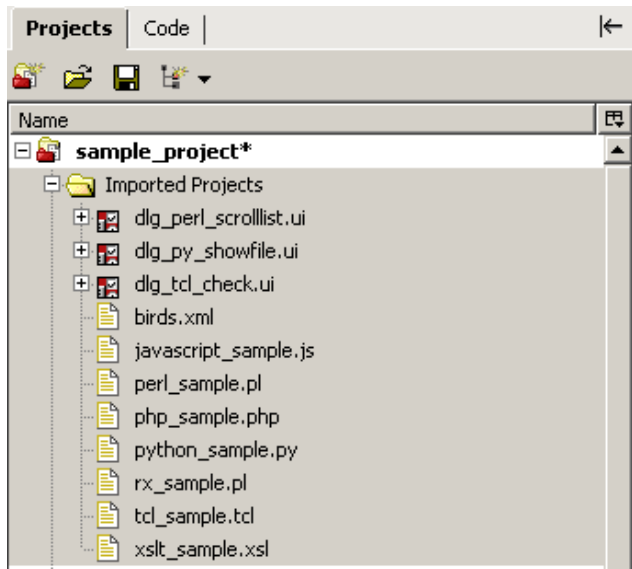
The files contained in the imported package are copied to the location on disk that you specified in the Package Import Wizard.

# Feature Showcase: Debug an XSLT Program

When debugging XSLT programs in Komodo, view the execution location of the XSLT file and the XML input file at the same time.

Before you start: Open the *xslt_sample.xsl* file contained in the sample project.
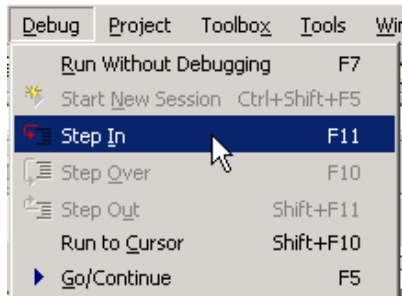


On the **Debug** menu, click **Step In**.

↓
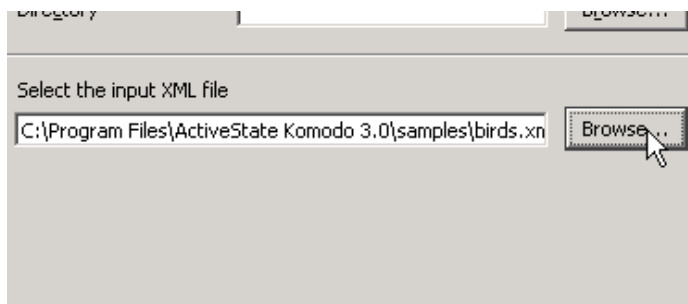


In the Debugging Options dialog box, enter *birds.xml* (also in the Komodo Sample Project) as the input file. Click **OK**.

↓
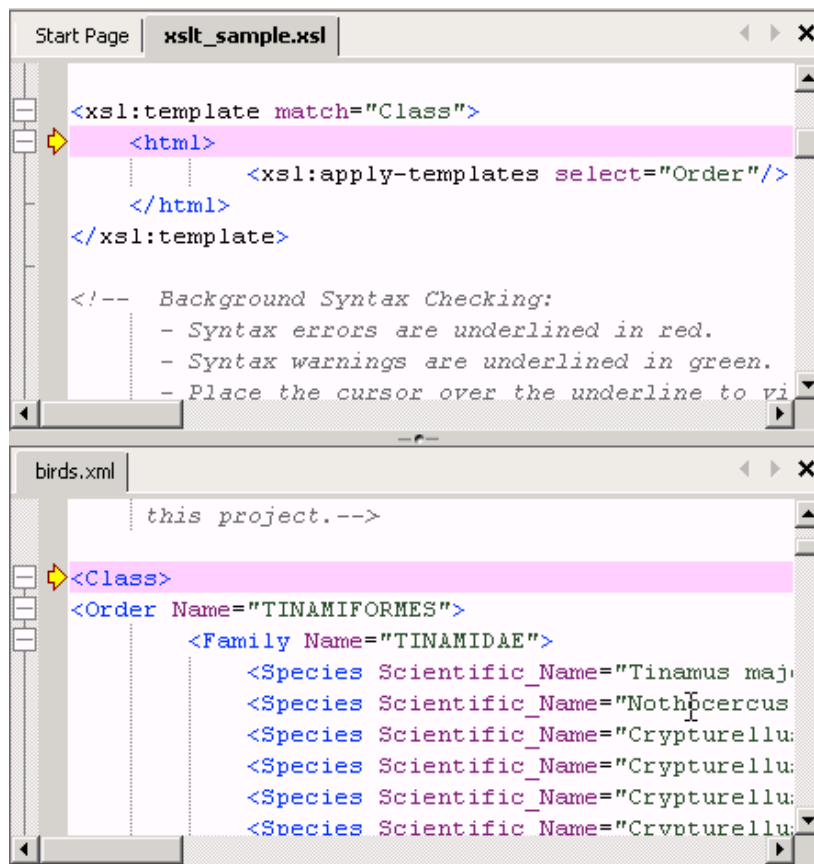
Komodo displays split editor pane with yellow arrows showing point of execution in both the XML and XSLT files. Step through the code using the *Step In* button ('F11'), or set breakpoints and use the *Go/Continue* button ('F5').

↓
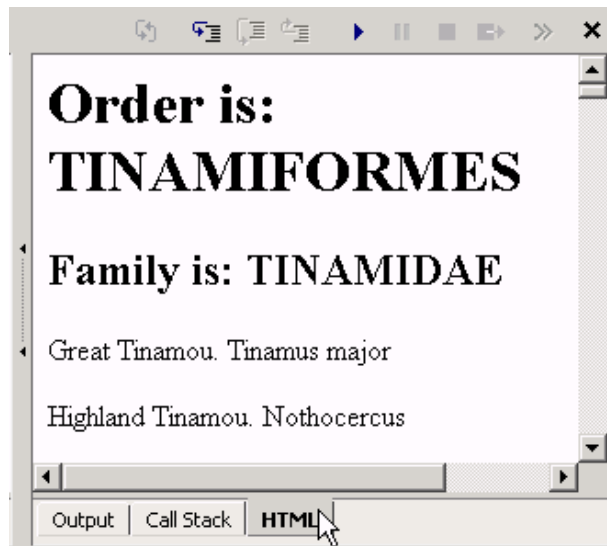
When debugging is complete the results of the transformation appear in the Output tab. Select the **HTML** tab to preview the rendered results.

# Perl Tutorial

## Perl Tutorial Overview

### Before You Start

This tutorial assumes...

- ...that ActivePerl build 623 or greater is installed on your system. ActivePerl is a free distribution of the core Perl language. See Komodo's Installation Guide for configuration instructions.
- ...that you have a connection to the Internet.
- ...that you are interested in Perl. You don't need to have previous knowledge of Perl; the tutorial will walk you through a simple program and suggest some resources for further information.

### Perl Tutorial Scenario

You have exported a number of email messages to a text file. You want to extract the name of the sender and the contents of the email, and convert it to XML format. You intend to eventually transform it to HTML using XSLT. To create an XML file from a text source file, you will use a Perl program that parses the data and places it within XML tags. In this tutorial you will:

1. ***Install a Perl module*** for parsing text files containing comma−separated values.
2. ***Open the Perl Tutorial Project*** and associated files.
3. ***Analyze parse.pl*** the Perl program included in the Tutorial Project.
4. ***Generate output*** by running the program.
5. ***Debug the program*** using the Komodo debugger.

## Installing Perl Modules Using VPM or PPM

One of the great strengths of Perl is the wealth of free modules available for extending the core Perl distribution. ActivePerl includes the Perl Package Manger (PPM) that makes it easy to browse, download and update Perl modules from module repositories on the internet. These modules are added to the core ActivePerl installation.

Komodo Professional Edition includes the Visual Package Manager (VPM), a graphical interface for PPM.

## Running the Visual Package Manager (Komodo Pro only)

The Text::CSV_XS Perl module is necessary for this tutorial. To install it using VPM:

1. Select *Tools/Visual Package Manager*, or click the VPM button on the Toolbar. The VPM *Install* tab opens in a browser.
2. In the *Search* field, enter:
   ```
   Text::CSV_XS
   ```

3. Click *Search*. Modules that match the search criteria are displayed in a list in the lower part of the screen.
4. Select the check box next to Text−CSV_XS and click *Install*. VPM connects to the default repository, downloads the necessary files and installs them.

## Running the Perl Package Manager (Komodo Personal)

The Text::CSV_XS Perl module is necessary for this tutorial. To install it using PPM:

1. Open the Run Command dialog box. Select *Tools/Run Command*.
2. In the *Run* field, enter the command:

   ```
   ppm install Text::CSV_XS
   ```

3. Click the *Run* button to run the command. PPM connects to the default repository, downloads the necessary files and installs them.

## About PPM and VPM

- PPM can be run directly from the command line with the `ppm` command. Enter `ppm help` for more information on command−line options.
- By default, PPM and VPM access the Perl Package repository at http://ppm.activestate.com. The ActiveState repository contains binary versions of most packages available from CPAN, the Comprehensive Perl Archive Network.
- More information about PPM is available on ASPN. PPM documentation is also included with your ActivePerl distribution.
- On Linux and Solaris systems where ActivePerl has been installed by the super−user (i.e. `root`), most users will not have permissions to install packages with VPM or PPM. Run `ppm` as root at the command line to install packages.

*Perl Pointer* It is also possible to install Perl modules without VPM or PPM using the CPAN shell. See the CPAN FAQ for more information.

# Opening Files

## Open the Perl Tutorial Project

Select *File*/*Open*/*Project* and choose `perl_tutorial.kpf` from the `/samples/perl_tutorial`
subdirectory of the Komodo installation. The files included in the tutorial project are displayed on the
Projects tab in the Left Pane. No files open automatically in the Editor Pane.

## Open the Perl Tutorial Files

On the Projects tab, double–click the files `parse.pl`, `mailexport.xml` and `mailexport.txt`.
These files will open in the Editor Pane; a tab at the top of the pane displays their names.

## Overview of the Tutorial Files

- *mailexport.txt* This file was generated by exporting the contents of an email folder (using the
  email program's own Export function) to a comma–separated text file. Notice that the key to the
  file contents are listed on the first line. The Perl program will use this line as a reference when
  parsing the email messages.
- *parse.pl* This is the Perl program that will parse mailexport.txt and generate mailexport.xml.
- *mailexport.xml* This file was generated by parse.pl, using mailexport.txt as input. When you run
  parse.pl (in Generating Output), this file will be regenerated.


# Analyzing the Program

## Introduction

In this step, you will examine the Perl program on a line–by–line basis. Ensure that Line Numbers are
enabled in Komodo (*View*/*View Line Numbers*). Ensure that the file "parse.pl" is displayed in the
Komodo Editor Pane.

## Setting Up the Program

### Line 1 – Shebang Line

- begins every Perl program
- associates the program with a Perl interpreter (in this case, the first one in the system's PATH
  variable)
- warning messages are enabled with the "–w" switch

*Komodo Tip* notice that syntax elements are displayed in different colors. You can adjust the display
options for language elements in the Preferences dialog box.

**Lines 2 to 4 – External Modules**

- these lines load external Perl modules used by the program
- Perl module files have a ".pm" extension; "use strict" uses the "strict.pm" module, part of the core Perl distribution
- "use Text::CSV_XS" refers to the module installed in Step One

## Writing the Output Header

### Lines 6 to 7 – Open Files

- input and output files are opened; if the output file does not exist, it is created
- scalar variables, indicated by the "$" symbol, store the files
- "strict" mode (enabled by loading "strict.pm" in line 2) requires that variables be declared using the format "my $variable"

*Perl Pointer* scalar variables store "single" items; their symbol ("$") is shaped like an "s", for "*s*calar".

### Lines 9 to 13 – Print the Header to the Output File

- "<<" is a "here document" indicator that defines the string to be printed
    - ♦ the text "EOT" is arbitrary and user–defined, and defines the beginning and end of the string
    - ♦ the second EOT on line 13 indicates the end of output
- lines 10 and 11 are data that will be printed to the output file

## Setting Up Input Variables

### Lines 15 to 16 – Assign Method Call to Scalar Variable

- the result of the method call "new" is assigned to the scalar variable $csv
- the method "new" is contained in the module Text::CSV_XS
- (`{binary => 1}`) tells the method to treat the data as binary

*Perl Pointer* good Perl code is liberally annotated with comments (indicated by the "#" symbol).

### Lines 18 to 19 – Method "getline"

- the method "getline" is contained in the module Text::CSV_XS, referenced in the $csv scalar variable
- "getline" reads the first line of mailexport.txt (referenced in the $in variable), parses the line into fields, and returns a reference to the resulting array to the $fields variable

## Starting the Processing Loop

**Line 21 – "while" Loop**

- the "while" statement is conditional
  - the condition is "1', so the program endlessly repeats the loop because the condition is always met
  - the logic for breaking out of the loop is on line 25
  - the loop is enclosed in braces; the opening brace is on line 21, the closing brace on line 51

*Komodo Tip* Click on the minus symbol to the left of line 21. The entire section of nested code will be collapsed. This is Code Folding.

*Komodo Tip* click the mouse pointer on line 21. Notice that the opening brace changes to a bold red font. The closing brace on line 51 is displayed the same way.

**Lines 22 to 25 – Extracting a Line of Input Data**

- the "getline" function extracts one line of data from the input file and places it in the $record scalar variable
- if "getline" returns an empty array, the input file has been fully processed and the program exits the loop and proceeds to line 52

*Perl Pointer* variable arrays store lists of items indexed by number; their symbol ("@") is shaped like an "a", for "*a*rray".

# Converting Characters with a Regular Expression

**Lines 27 to 31 – "foreach"**

- "foreach" cycles through the elements stored in the @$record array
- the regular expressions on lines 29 and 30 find the characters "<" and "&", and replace them with their character entity values ("<" and "&" are reserved characters in XML)

*Komodo Tip* Komodo's Rx Toolkit is a powerful tool for creating and debugging regular expressions. See Regular Expressions Primer for more information.

# Combining Field Reference and Field Data

**Lines 33 to 35 – hash slice**

- line 35 combines the @$record array with the field reference generated in line 19

*Perl Pointer* variable hashes are indicated by the symbol "%", and store lists of items indexed by string.

# Writing Data to the Output File

**Lines 37 to 50 – Writing Data to the Output File**

- one line at a time, lines from the input file are processed and written to the output file

- portions of the data line (stored in the $record scalar variable) are extracted based on the corresponding text in the field reference (the first line in the input file, stored in the $fields variable)

## Closing the Program

### Line 51 – Closing the Processing Loop

- at line 51, processing will loop back to the opening brace on line 21
- the logic to exit the loop is on line 25

### Lines 52 to 54 – Ending the Program

- line 52 prints the closing tag to the XML file
- lines 53 and 54 close the input and output files

# Run the Program to Generate Output

To start, you will simply generate the output by running the program through the debugger without setting any breakpoints.

1. *Clear the contents of mailexport.xml* Click on the "mailexport.xml" tab in the Editor Pane. Delete the contents of the file – you will regenerate it in the next step. Save the file.
2. *Run the Debugger* Click on the "parse.pl" tab in the editor. From the menu, select *Debug/Go/Continue* (or 'F5'). In the Debugging Options dialog box, click *OK* to accept the defaults.
3. *View the contents of mailexport.xml* Click on the "mailexport.xml" tab in the editor. Komodo informs you that the file has changed. Click *Yes* to reload the file.

# Debugging the Program

In this step you'll add breakpoints to the program and "debug" it. Adding breakpoints lets you to run the program in chunks, making it possible to watch variables and view output as it is generated. Before you begin, ensure that line numbering is enabled in Komodo (*View/View Line Numbers*).

1. *Set a breakpoint*: On the "parse.pl" tab, click in the grey margin immediately to the left of the code on line 9 of the program. This will set a breakpoint, indicated by a red circle.
2. *Run the Debugger*: Select *Debug/Go/Continue* (or 'F5', or use the Debug Toolbar). In the Debugging Options dialog box, click *OK* to accept the defaults. The debugger will process the program until it encounters the first breakpoint.

***Komodo Tip*** Debugger commands can be accessed from the Debug menu, by shortcut keys, or from the Debug Toolbar. For a summary of debugger commands, see Debugger Command List.

3. ***Watch the debug process***: A yellow arrow on the breakpoint indicates the position at which the debugger has halted. Click on the "mailexport.xml" tab. Komodo informs you that the file has changed. Click ***Yes*** to reload the file.
4. ***View variables***: In the Bottom Pane, see the ***Debug*** tab. The variables "$in" and "$out" appear in the ***Locals*** tab.
5. ***Line 9 – Step In***: Select ***Debug/Step In***. "Step In" is a debugger command that causes the debugger to execute the current line and then stop at the next processing line (notice that the lines between 9 and 16 are raw output indicated by "here" document markers).
6. ***Line 16 – Step In***: On line 16, the processing transfers to the module Text::CSV_XS. Komodo opens the file CSV_XS.pm and stops the debugger at the active line in the module.
7. ***Line 61 – Step Out***: Select ***Debug/Step Out***. The Step Out command will make the debugger execute the function in Text::CSV_XS and pause at the next line of processing, which is back in parse.pl on line 19.
8. ***Line 19 – Step Over***: Select ***Debug/Step Over***. The debugger will process the function in line 19 without opening the module containing the "getline" function.

***Komodo Tip*** What do the debugger commands do?

- **Step In** executes the current line of code and pauses at the following line.
- **Step Over** executes the current line of code. If the line of code calls a function or method, the function or method is executed in the background and the debugger pauses at the line that follows the original line.
- **Step Out** when the debugger is within a function or method, Step Out will execute the code without stepping through the code line by line. The debugger will stop on the line of code following the function or method call in the calling program.

9. ***Line 21 – Set Another Breakpoint***: Click in the grey margin immediately to the left of the code on line 21 to set another breakpoint.
10. ***Line 21 – Step Out***: It appears that nothing happened. However, the debugger actually completed one iteration of the "while loop" (from lines 21 to 51). To see how this works, set another breakpoint at line 37, and Step Out again. The debugger will stop at line 37. On the Debug Session tab, look at the data assigned to the $record variable. Then Step Out, and notice that $record is no longer displayed, and the debugger is back on line 21. Step Out again, and look at the $record variable – it now contains data from the next record in the input file.
11. ***Line 37 – Stop the Debugger***: Select ***Debug/Stop*** to stop the Komodo debugger.

***Perl Pointer*** Did you notice that output wasn't written to mailexport.xml after every iteration of the while loop? This is because Perl maintains an internal buffer for writing to files. You can set the buffer to "autoflush" using the special Perl variable "$|".

# More Perl Resources

## ASPN, the ActiveState Programmer Network

ASPN, the ActiveState Programmer Network, provides extensive resources for Perl programmers:

- Free downloads of *ActivePerl*, ActiveState's Perl distribution
- Searchable *Perl documentation*
- Trial versions of *Perl tools*, like the Perl Dev Kit and Visual Perl
- The *Rx Cookbook*, a collaborative library of regular expressions for Perl

## Documentation

There is a wealth of documentation available for Perl. The first source for language documentation is the Perl distribution installed on your system. To access the documentation contained in the Perl distribution, use the following commands:

- Open the *Run Command* dialog box (*Tools*/*Run Command*), and then type `perldoc perldoc`. A description of the "perldoc" command will be displayed on your screen. Perldoc is used to navigate the documentation contained in your Perl distribution.

## Tutorials and Reference Sites

There are many Perl tutorials and beginner Perl sites on the Internet, such as:

- Introduction to Perl, a course developed by the University of Missouri
- learn.perl.org, which provides book reviews, tips, and access to Perl news lists and books

# PHP Tutorial

## Overview

### Before You Start

This tutorial assumes:

- [PHP 4.3.1](#) or greater is installed on your system. See Komodo's [Installation Guide](#) for configuration instructions.
- You are interested in PHP. You don't need previous knowledge of PHP; the tutorial will walk you through a simple program and suggest some resources for further information.

### PHP Tutorial Scenario

This tutorial examines a PHP program that implements a form on a website – in this case, a guest book where site visitors can log comments. In addition to providing an overview and working example of PHP, the tutorial introduces Komodo's [CGI Debugging](#) functionality. In this tutorial you will:

1. ***[Open the PHP Tutorial Project](#)*** and associated files.
2. ***[Analyze guestbook.php](#)***, the PHP program included in the PHP Tutorial Project.
3. ***[Run the program and generate HTML output](#)*** by running the program.
4. ***[Debug the program](#)*** using the Komodo debugger.

See [Debugging Programs](#) for more information on this Komodo functionality.

# Opening the Tutorial Project

On the ***File*** menu, click ***Open/Project*** and navigate to the *php_tutorial.kpf* project file on your filesystem (*install dir\Komodox.x\samples\php_tutorials* in Windows and *install dir/Komodo x.x/samples/php_tutorials* in Unix. All files included in the tutorial project are displayed on the [Projects tab](#) in the Left Pane.

### Overview of the Tutorial Files

The following components are included in the *php_tutorial.kpf* project file:

- ***guestbook.php***: This PHP program writes data from an HTML form to a data file, then extracts the contents of the data file and formats it as HTML.

## Open the PHP Tutorial File

On the Projects tab, double−click the file *guestbook.php*. The file opens in the Editor Pane; a tab at the top of the pane displays the filename.

# Analyzing the PHP Tutorial File

This section reviews the code in *guestbook.php*.

# Analyzing guestbook.php

## Introduction

In this step, you will analyze the PHP program on a line−by−line basis. Ensure that line numbers are enabled in Komodo (***View***/***View Line Numbers***) and that the file *guestbook.php* is displayed in the Komodo editor.

## HTML Header

### Lines 1 to 8 – HTML Header

- a standard HTML header is written to the program output

***Komodo Tip***: Notice that syntax elements are displayed in different colors. Adjust the display options for language elements in the Preferences
dialog box.

## PHP Declaration and Datafile

### Line 9 – PHP Declaration

- PHP programs are embedded in HTML
- the characters `<?php` indicate the start of the PHP program

### Lines 10 to 18 – Comments

- the `//` characters indicate a single−line comment in PHP programs; the # symbol can also be used. Multi−line comments are nested in `/*` and `*/` characters, as shown on lines 27 to 30

### Line 22 – Datafile

- the file *guestbook.dat* is created if it does not exist
- the `tmp` directory must exist beneath the root of the drive where the program resides (unless a different location is specified in the Debugging Options).
- PHP statements are terminated with semicolons

*Komodo Tip*: On line 23, type `$da`. Komodo displays a list of the variables declared above the cursor position that begin with the letters `da`. This is AutoComplete.

## GuestBook Class

### Lines 25 to 28 – Class Declaration

- a `class` is a collection of variables and functions
- `class GuestBook` contains the functions `GuestBook`, `_getData"`, `outputData`, etc
- the `var` statement declares variables as class members, thus making them portable across functions contained in the class

*Komodo Tip*: Click the mouse pointer at the end of line 25. Notice that the brace changes to a bold red font. The closing brace on line 144 is displayed the same way. In this case, the braces mark the beginning and end of a class. See Editing Files in the Komodo User Guide for more about matching braces.

## GuestBook Function

### Lines 34 to 37 – GuestBook Function

- a function is a discrete block of code
- the `$datafile` argument is passed to the function `GuestBook`; multiple arguments are separated by commas
- the contents of a function are enclosed in braces
- `$_SERVER` is a pre–defined PHP variable; it is passed to the script from the web server
    - in PHP, global variables must be declared to be global inside a function if they are going to be used in that function
- a local variable is defined for the current function by use of the term `$this`; notice that the same syntax is used to call another function
    - `gb_dat` variable is declared on line 27
    - `gb_dat` variable is assigned the value of `$datafile`
    - `$this->data` variable is cleared of any prior value
    - `$this->_getData` variable calls the `_getData` function that begins on line 53; when the `_getData` function is complete, processing returns to line 40

*Komodo Tip*: On line 38, type `function GuestBook(`. When you type the left parenthesis, Komodo displays a pop–up hint that describes parameters for the function `GuestBook`. This is a CallTip.

### Lines 40 to 44 – Check for Valid Form Entry

- if the `REQUEST_METHOD` contained in `$_SERVER` is equal to `POST`, processing passes to the `addGuestBookEntry` function on line 120
- if the `REQUEST_METHOD` is not equal to `POST`, a redirect message is displayed to the user
    - the `echo` command generates output
    - the characters `\"` are not included inside the double quotation marks that follow, so that the message can be displayed as output
    - the PHP variable `PHP_SELF` is the filename of the current script

♦ $_SERVER["PHP_SELF"] extracts the PHP_SELF variable from the $_SERVER variable

**Lines 45 to 46 – Check for Variable Value**

- the if ($this->data) statement tests if the variable $this->data has a value
    ♦ the program executes the outputData function and then the outputForm function

# _getData Function

**Lines 53 to 58 – _getData Function**

- the "file" statement parses the contents of the file stored in the gb_dat variable into the $lines array
    ♦ the @ symbol suppresses warnings; in this case, if the data file is empty, the program generates a non–fatal error
- the if ($lines) statement checks to see if the $lines variable has data
- the "join" statement converts the $lines array to a string and places it in the variable $this->data

*PHP Pointer*: Use the "@" operator with care; you could disable error messages for critical errors that terminate the execution of the script.

# outputData Function

**Lines 64 to 66 – outputData Function**

- the contents of the $this->data variable are written to the standard output using the echo statement

# _createEntryHTML Function

**Lines 72 to 77 – Retrieve Form Data**

- the PHP variable $_POST is used to provide data to the script via HTTP POST
- lines 74 to 77 extract the form data and place the items in variables

**Lines 80 to 83 – Validate Form Data**

- On line 80, the validity of the name and message variables is tested:
    ♦ in !$name and !$message, "!" is a "not" operator; it is *true* if either variable is *not true*
    ♦ The || symbol is an "or" operator

*PHP Pointer*: PHP has two "or" operators: the word "or", and the symbol ||. The || operator has precedence over the word "or", providing flexibility in logic tests.

**Line 86 – Current Date and Time**

- the variable $today contains the result of the PHP function date:
    - the date function returns a string
    - the "switches" are interpreted as follows:
        - ◊ F: text month
        - ◊ j: numeric day within month
        - ◊ y: four digit year
        - ◊ g: hour (12 hour format)
        - ◊ a: AM / PM

**Lines 89 to 94 – Interpolate Form Data with HTML**

- text and HTML tags are parsed with the $today variable and the form data
- the return statement supplies the result (true or false) of a function or the value of a variable to the routine from which it was called

# _writeDataFile Function

## Lines 100 to 106 – Open the Data File

- the fopen function opens the file stored in the $this->gb_dat variable
    - the w switch opens the file if it exists
    - If the file does not exist, fopen will attempt to create it
    - the file is opened for writing only, and the file pointer is positioned at the top of the file
- the if !$f statement checks to see if the $f variable contains a value

## Lines 108 to 110 – Write to the Data Files

- the fwrite function writes the contents of $this–>data to the file contained in the $f variable

## Lines 111 to 113 – Close the Data File

- the fclose function closes the file stored in the $f variable
- the value of the return statement is tested on line 112

*Komodo Tip*: Click on the minus symbol to the left of line 100. The entire _writeDataFile function collapses. This is Code Folding.

# addGuestBookEntry Function

## Lines 120 to 125 – Call Functions for Writing Data

- the $entry variable is local to the addGuestBookEntry function
- the $entry contains the contents of the $data variable, returned in the _createEntryHTML function

- on line 123, the contents of `$entry` are concatenated with the contents of `$this->data`, and stored in `$this->data`

## outputForm Function

### Lines 127 to 142 – The Function for HTML Form

- these lines generate a standard HTML form
- notice the PHP snippet on line 133 that provides the program name to the HTML output

## Closing Tags

### Lines 148 to 151 – Closing Tags

- the `$gb` variable creates a new instance of the `GuestBook` class using the file specified in the `$datafile` variable
- when the functions in the `GuestBook` class are complete, the PHP program is closed using the syntax `?>`
- closing HTML tags are written as output

# Running the Program

This section reviews how to run the *guestbook.php* program using the Komodo debugger.

1. ***Run the debugger***: Select ***Debug/Start*** (or 'F5').
2. ***Configure debugging options***: In the Debugging Options dialog box, configure the following options:
   - ♦ ***General tab:*** Select the ***Simulate CGI Environment*** check box.
   - ♦ ***CGI Input tab:***
     - ◊ Set the ***Request Method*** option button to ***Post***.
     - ◊ On the ***Post Type*** drop–down list, select ***URL encoded***.
     - ◊ On the ***Type*** drop–down list, select the variable type ***Post***.
     - ◊ Enter the following names in the ***Name*** text box, adding a meaningful value for each in the ***Value*** text box. For example, the value for "name" could be your own name. Click the ***Add*** button after each entry to add it to the ***Browser Arguments***.
       - · "name"
       - · "email"
       - · "company"
       - · "message"
3. ***Run the debugger***: Click ***OK*** to run the debugger with the selected options.
4. ***View the Command Output tab***: Notice the messages in the bottom left corner of the Komodo screen; these inform you of the status of the debugger.

5. *View the rendered HTML*: Click the *HTML* tab on the right side of the *Debug* tab. The HTML is displayed in the Bottom Pane; previous guestbook entries are displayed at the top of the output, and the HTML form is displayed at the bottom. Click the *Output* tab to return to the HTML code.
6. *Create New File*: To create a new HTML file that contains the HTML code on the *Output* tab, select *File/New/New File*. In the New File dialog box, select the *Common* Category, and the *HTML* template. Click *Open*.
7. *Save the Output*: Delete the contents of the new HTML file tab in the Editor Pane, then select the HTML code on the *Output* tab. Copy the contents to the new HTML file tab in the Editor Pane ('Ctrl'+'C', 'Ctrl'+'V' if the default key binding scheme is in effect). Select *File/Save As* to save the file with a unique name.

# Debugging the Program

In this step you will add breakpoints to the program and debug it. Adding breakpoints lets you run the program in chunks, making it possible to watch variables and view output as it is generated. Before beginning, ensure that line numbering is enabled in Komodo (*View/View Line Numbers*).

1. *Set breakpoints*: On the *guestbook.php* tab in the editor, click on the gray margin immediately to the left of the code in line 9 of the program. This sets a breakpoint, indicated by a red circle. Set a second breakpoint on line 148.
2. *Run the debugger*: Select *Go/Continue* (or 'F5', or use the Debug Toolbar). In the Debugging Options dialog box, click *OK* to accept the defaults (assuming that you created the CGI variables in the previous step, Running the Program).

*Komodo Tip*: Notice that the Debugger Options have been saved from the last time a PHP program was run or debugged.

*Komodo Tip*: Debugger commands can be accessed from the *Debug* menu, by shortcut keys, or from the Debug Toolbar. For a summary of debugger commands, see Debugger Command List.

3. *Watch the debug process*: A yellow arrow on the breakpoint indicates the position at which the debugger has halted.
4. *Line 9: Step In*: Select *Debug/Step In*. "Step In" is a debugger command that causes the debugger to execute the current line and then stop at the next processing line (line 19). The lines between line 9 and line 19 are comments, not processing statements, and are therefore ignored by the debugger.
5. *View Variables*: Expand the Bottom Pane (if necessary) by clicking and dragging the bottom margin of the Komodo workspace. Variables defined in the program are displayed on the *Locals* tab.

6. *Line 19*: Select *Go/Continue* (or 'F5', or use the Debug Toolbar). The debugger moves to line 148. The GuestBook class is called from line 148.
7. *Line 148: Step In*: The debugger is now processing the GuestBook function.

8. *View Variables*: The *Locals* tab displays all variables.

9. *Line 35: Step In*: Expand the `$this` variable on the *Locals* tab in the Bottom Pane. Notice that it now has a sub–variable `gb_dat`, which stores the value of the data file.
10. *Line 36: Step In*: Continue to step in until the debugger stops at the `_getData` function. Continue to select *Step In* to process each statement in the function. After line 57 has been processed and the debugger is stopped at line 58, the `$lines` variable can be expanded on the *Locals* tab.
11. *Line 58: Step Out*: On line 58, select *Step Out* to process the rest of the `_getData` function. The debugger will proceed to line 40, which follows the line where `_getData` was called.

*Komodo Tip*: What do the debugger commands do?

- **Step In**: Executes the current line of code and pauses at the following line.
- **Step Over**: Executes the current line of code. If the line of code calls a function or method, the function or method is executed in the background and the debugger pauses at the line that follows the original line.
- **Step Out**: Executes the code without stepping through the code line by line (when the debugger is within a function or method). The debugger stops on the line of code following the function or method call in the calling program.

14. *Line 40: Step In*: Continue to select *Step In* until the debugger is on line 121, in the `addGuestBookEntry` function. On line 121, the `addGuestBookEntry` function calls the `_createEntryHTML` function.
15. *Line 121: Step In*: In the `_createEntryHTML` function, the program assigns variables to the CGI input data configured in the Debugging Options.
16. *Line 74: Step Out*: The `_createEntryHTML` function completes, and processing returns to line 122.
17. *Line 122: Step In*: Use *Step In* to process each line of the `addGuestBookEntry` function, until processing moves to the `_writeDataFile` function on line 102.
18. *Line 102: Step In*: Process line 102.
19. *Open Watch Window*: On line 102, the program opened the datafile (by default, *\tmp\guestbook.dat*). To watch the activity in the datafile, select *Tools/Watch File*, then specify the datafile.
20. *Line 103: Step In*: Continue to select *Step In* until line 108 has been processed. After line 108 is processed, the contents of the `$this->data` variable are written to the datafile, as displayed in the *Watch* tab.
21. *Line 111: Step In*: *Step In* until processing returns to line 45 of the `GuestBook` function.
22. *Line 45: Step Over*: The *Step Over* debugger command executes the current line, including any functions called by the current line. When the debugger returns to line 46, notice that the contents of the `$this->data` variable have been written to the Bottom Pane.
23. *Line 46: Step Over*: The debugger executes the `outputForm` function, which writes the HTML form to the Bottom Pane.
24. *Continue*: Select *Debug/Continue* to run the debugger to the end of the program.

# More PHP Resources

## ASPN, the ActiveState Programmer Network

ASPN, the ActiveState Programmer Network, provides resources for PHP programmers:

- *Mailing lists* with many PHP topics.

## Tutorials and Reference Sites

There are many PHP tutorials and beginner PHP sites on the Internet, including:

- The PHP Resource Index, a collection of resources dealing with PHP.
- www.PHP.net, the home of all that is PHP–related.

# Python Tutorial

## Overview

### Before You Start

This tutorial assumes:

- [Python 2.3](#) or greater is installed on your system. ActivePython is a free distribution of the core Python language. See the Komodo [Installation Guide](#) for configuration instructions.
- You are interested in learning about Komodo functionality, including the debugger and the interactive shell.
- You are interested in Python and have some programming experience either in Python or another language.

### Python Tutorial Scenario

The Python Tutorial demonstrates how to use the Komodo debugger and interactive shell to explore a Python program. In particular, this tutorial examines a Python script that preprocesses files (similar to the C preprocessor). In this tutorial you will:

1. ***Open the Python Tutorial Project***.
2. ***Analyze preprocess.py*** the Python program included in the Tutorial Project.
3. ***Analyze contenttype.py*** the Python module included in the Tutorial Project.
4. ***Run the program*** and generate program output.
5. ***Debug the program*** using the Komodo debugger.
6. ***Explore Python*** using the Komodo interactive shell.

See [Interactive Shell](#) and [Debugging Programs](#) for more information on this Komodo functionality.

# Opening the Tutorial Project

From the *File* menu, click *Open/Project* and navigate to the *preprocess.kpf* project file on your file system (*install dir\Komodo x.x\samples\python_tutorials* in Windows and *install dir/Komodo x.x/samples/python_tutorials* in Unix). All files included in the tutorial project are displayed on the [Projects tab](#) in the Left Pane.

### Overview of the Tutorial Files

The following components are included in the *preprocess.kpf* project file:

- *preprocess.py*: The main program. This Python program parses input source files and produces output filtered on a set of rules and statements embedded in the original input source.
- *preprocess current file*: A run command for executing *preprocess.py* on the file currently open in Komodo.
- *contenttype.py*: A Python module used by the main program (*preprocess.py*) to identify the language of a given file.
- *content.types*: A support file used by the Python module *contenttype.py*.
- *helloworld.html* and *helloworld.py*: Sample files to process using *preprocess.py*.

## Open the Python Tutorial File

On the Projects tab, double−click the *preprocess.py* file. This file opens in the Editor Pane; a tab at the top of the pane displays the filename.

# Analyzing the Python Files

This section reviews the code in preprocess.py and contenttype.py.

## Analyzing preprocess.py

In this step, you will analyze the Python program *preprocess.py* in sections. This program is an advanced Python script that is best addressed by focusing on certain areas within the code. Be sure that line numbers are enabled in Komodo (*View/View Line Numbers*) and that *preprocess.py* is displayed in the Komodo Editor.

*About Preprocessors*: A preprocessor is a program that examines a file for specific statements called "directive statements". These directive statements are interpreted, and the resulting program output is conditional based on those statements. In languages like C/C++, preprocessing is a common step applied to source files before compilation. The Python *preprocessor.py* program mimics a C/C++ preprocessor using similar directive statements.

*About Directive Statements*: Preprocessor directive statements are dependent on the preprocessor program they are used within. In the *preprocessor.py* program, a directive is preceded with a pound sign (#), and is located alone on a line of code. Placing a directive on a unique line ensures the statement is included in a file without breaking file syntax rules. Valid *preprocessor.py* directives include:

```
#define <var>[=<value>]
#undef <var>
#if <expr>
#elif <expr>
#else
#endif
#error <error string>
```

## Setting Up the preprocess.py Program

*Komodo Tip*: Notice that syntax elements are displayed in different colors. You can adjust the display options for language elements in the Preferences dialog box.

### Lines 3 to 57 – Defining a Module Docstring

- help is defined in a module docstring
- docstrings are contained in triple–quoted strings (""")

*Komodo Tip*: See Explore Python with the Interactive Shell to examine these docstrings, and other Python elements, using the Komodo interactive shell.

*Komodo Tip*: Click on the minus symbol to the left of line 3. The entire section of nested help code is collapsed. This is called Code Folding.

### Lines 59 to 65 – Importing Standard Python Modules

- Imports the following six modules:
    - ♦ *os*: operating system dependant helper routines
    - ♦ *sys*: functions for interacting with the Python interpreter
    - ♦ *getopt*: parses command line options
    - ♦ *types*: defines names for all type symbols in the standard Python interpreter
    - ♦ *re*: evaluates regular expressions
    - ♦ *pprint*: supports pretty–print output
    - ♦ *logging*: writes errors to a log file

### Line 67 – Importing the contenttype Module

The custom `contenttype` module is used by the *preprocess.py* program and is not included in a standard Python installation.

- loads the `contenttype` module and imports the `getContentType` method

*Komodo Tip*: To interact directly with the `contenttype.py` module, see Explore Python with the Interactive Shell for more information.

## Defining an Exception Class

### Lines 72 to 88 – Declaring an Exception

- `PreprocessError` class inherits from the Python `Exception` class
- an instance of the `PreprocessError` class is thrown by the `preprocess` module when an error occurs

*Komodo Tip*: Click the mouse pointer on the closing parenthesis ")" on line 72. Notice that its color changes to a bold red. The opening brace is displayed the same way. This is called "Brace Matching". Related features in Komodo are ***Jump to Matching Brace*** and ***Select to Matching Brace***, available via

the *Code* menu.

## Initializing Global Objects

### Line 93 – Initializing log

- `log` is a global object used to log debug messages and error messages

*Komodo Tip*: On line 95, enter: `log = logging.`
When you type the period, Komodo displays a list of the members in the `log` package. This is called [AutoComplete](). Pressing 'Ctrl'+'J' (if the default [key binding]() scheme is in effect) also displays the AutoComplete list. Delete the contents of line 95.

### Lines 98 to 111 – Mapping Language Comments

- `_commentGroups` is a mapping of file type (as returned by `content.types`) to opening and closing comments delimiters
- mapping is private to the `preprocess.py` module (`_commentGroups` is prefixed with an underscore to indicate that it is private to the `preprocess.py` module). This is a common technique used in variable, function, and class naming in Python coding).

Note that preprocessor directives recognized by the `preprocess.py` module are hidden in programming language–specific comments.

*Komodo Tip*: Use the *Code* tab, located in the Left Pane, to browse the general program structure of all currently open files. For each file, the code browser shows a tree of classes, functions, methods and imported modules. Python instance attributes are also displayed.

## Defining a Private Method

### Lines 116 to 123 – Expression Evaluation

- `_evaluate method` is private to the `preprocess` module
- evaluates the given expression string with the given context

## Preprocessing a File

The `preprocess` method examines the directives in the sample source file and outputs the modified processed text.

### Lines 129 to 140 – The preprocess Method Interface

The `preprocess` method takes three parameters as input:

- first parameter is the filename, `infile`
- second parameter specifies the output file (defaults to `stdout`); `outfile=sys.stdout`
- third parameter is an optional list of definitions for the preprocessor; `defines={}`

### Lines 145 to 156 – Identifying the File Type

Examines how programming comments are delimited (started and ended) based on the type of file (for example, HTML, C++, Python).

- `getContentType` is called (imported earlier from the `contenttype.py` module) to determine the language type of the file
- file type is used to look up all comment delimiters (opening and closing language comment characters) in `_commentGroups`

### Lines 159 to 166 – Defining Patterns for Recognized Directives

This section defines advanced regular expressions for finding preprocessor directives in the input file.

***Komodo Tip***: Use the Komodo Rx Toolkit to build, edit, or test regular expressions. New to regular expressions? The Regular Expressions Primer is a tutorial for those wanting to learn more about regex syntax.

### Lines 178 to 303 – Scanning the File to Generate Output

This block of code implements a basic state machine. The input file is scanned line by line looking for preprocessor directives with the patterns defined above (`stmtRes`). This code determines whether each line should be skipped or written to the output file.

- source file is processed
- output is generated by a state machine implemented in Python

### Lines 311 to 349 – Interpreting Command Line Arguments

The `main` method takes the text entered at the command line and uses the `getopt` module to parse the data into arguments. These arguments are then passed into the "preprocess" method.

- runs when *preprocess.py* is executed as a program rather than loaded as a module
- parses the filename and any defines (`-D`) set as command line arguments
- passes all data to the `preprocess` method

### Lines 351 to 352 – Running the Main Method

- runs the `main` method when *preprocess.py* is executed as a program

## Analyzing contenttype.py

In this step, you will analyze the Python program *contenttype.py* in sections. This Python script is best addressed by focusing on certain areas within the code. Be sure that line numbers are enabled in Komodo (***View/View Line Numbers***) and that *contenttype.py* is displayed in the Komodo Editor Pane.

## Open contenttype.py

On the Projects tab, double−click the *contenttype.py* file. This file opens in the Editor Pane; a tab at the top of the pane displays the filename.

## Setting Up the contenttype.py Module

The `contenttype.py` module is used by the main program, *preprocess.py*, to identify what programming language a particular file is written in based on the file extension and several other tests.

### Lines 16 to 19 – Importing External Modules

- imports external modules used in this file (`re`, `os`, `sys`, `logging`)
- `logging` is not a standard module; it is new in Python 2.3

## Getting Data from content.types

### Lines 29 to 31 – Finding the Helper File (content.types)

This section outlines the usage of the private `_getContentTypesFile` method located in the `contenttype` module.

- returns the complete path to the `content.types` file
- assumes the file is in the same directory as *contenttype.py*
- `_getContentTypesFile` is a private method that cannot be accessed from outside of the `contenttype` module

### Lines 33 to 80 – Loading the Content Types from content.types

This section outlines the usage of the private `_getContentTypesRegistry` method located in the `contenttype` module.

- locates the `content.types` file and scans it to calculate three mappings to return, as follows:

```
file suffix -> content type (i.e. ".cpp", a C++ implementation file)
regex -> content type (i.e. ".*\.html?", an HTML file)
filename -> content type (i.e. "Makefile", a Makefile)
```

- `_getContentTypesRegistry` is a private method that cannot be accessed from outside of the `contenttype` module.
  - ♦ *Lines 44 to 45*: gets the `content.types` file; if none is specified in the parameter for the method, `_getContentTypesFile` is called to find the system default
  - ♦ *Lines 47 to 49*: lists the three mappings to return (empty mappings are created here)
  - ♦ *Lines 51 to 79*: opens and processes the `content.types` file on a line−by−line basis
    - ◊ scanning of the file stops when the last line is found, line 57
      - · *Lines 58 to 78*: each line is parsed to determine which of the three

mappings it contains

· an entry is made in the matching one

· commented lines (starts with #) are ignored

♦ *Lines 79 to 80*: closes the `content.types` file and returns the mappings

## Lines 85 to 118 – Determining a File's Content Type

This section outlines the usage of the public `getContentType` method located in the `contenttype` module.

- takes one parameter (the name of the file to determine the content)
- returns a string specifying the content type (for example, `getContentType("my_web_page.htm")` returns "HTML")
- `getContentType` is the only publicly accessible method in the module
    ♦ *Line 92*: `_getContentTypesRegistry` is called to load the `content.types` file and to load the mappings
    ♦ *Lines 96 to 99*: `filenameMap` is first checked to determine if the whole filename can be used to find a match
    ♦ *Lines 101 to 109*: if the filename has a suffix (contains a '.'), the suffix map is then used to find a match
    ♦ *Lines 111 to 117*: each regex in the regex map is then used to determine if it matches the filename
    ♦ *Line 118*: returns the content type for the file (returns an empty string if no match was found by the above three mappings)

# Running the Program

This section reviews how to run the *preprocess.py* program using both a run command and the Komodo debugger.

## Using a Run Command

To start, generate simple output by running the program with the *preprocess current file* run command, included in the *preprocess.kpf* project.

1. *Open the Source File*: On the *Projects* tab, double−click the *helloworld.html* file. The file opens in the Editor Pane.
2. *Open the Run Command*: On the *Projects* tab, double−click the *preprocess current file* run command. A Preprocess Current File dialog box appears.
3. *Preprocess Current File*: In the *Preprocessor Options* text area, enter:
   `−D SAY_BYE`
   Click *OK* to run the program.
4. *View Output*: The *helloworld.html* file output is displayed on the *Command Output* tab as follows:

```
  ['path_to_file\\python_tutorials\\helloworld.html']
<html>
<head> <title>Hello World</title> </head>
<body>
<p>Hello, World!</p>
</body>
</html>
```

*Python Tutorial Tip*: For more information about the -D SAY_BYE command, see Using the Debugger.

*Komodo Tip*: For more infomation on using run commands in Komodo, see the Run Command Tutorial.

## Using the Debugger

Generate output by running the program through the debugger without setting any breakpoints.

1. *Run the debugger*: Select the *preprocess.py* tab in the editor. From the menu, select *Debug/Go/Continue* (or press 'F5'). In the Debugging Options dialog box, click *OK* to accept the defaults.
2. *View the Debug Output Tab*: Notice the messages in the bottom left corner of the Komodo screen; these inform you of the status of the debugger. When the program has finished, program output is displayed in the Bottom Pane, on the right side. If necessary, click the *Debug Output* tab to display it.

*Troubleshooting*: "Why is this error message displayed?"

```
preprocess: error: incorrect number of arguments:
argv=['C:\\path_to_tutorial\\preprocess.py']
```

This error message is the expected output by the *preprocess.py* program when no source file or arguments are specified before it is run. The following instructions explain how to specify a file at the command line.

3. *Specify a File to Process*: On the *Projects* tab in the Left Pane, double−click the file *helloworld.html*. Note the preprocessor directives inside the comments (#) in this file. Select the *preprocess.py* tab in the editor. From the menu select *Debug/Go/Continue* (or press 'F5'). In the *Script Arguments* text box on the Debugging Options dialog box, enter helloworld.html. Click *OK*.

*Troubleshooting*: "Why is this error message displayed?"

```
<html>
<head> <title>Hello World</title> </head>
<body>
preprocess: error: helloworld.html:5: #error: "SAY_BYE is not
defined, use '-D' option"
```

This error message is the expected output by the *preprocess.py* program when no command–line arguments are specified with the source file *helloworld.html*. The following instructions explain how to specify a command–line argument with the source file to be processed.

4. ***Specify an Argument with the Source File***: Select ***Debug/Go/Continue*** (or press 'F5'). In the ***Script Arguments*** text box in the Debugging Options dialog box, enter the following source file and argument: `–D SAY_BYE helloworld.html`. Click ***OK***.

***Troubleshooting***: Specifying `–D SAY_BYE helloword.html` outputs the following:

```
<html>
<head> <title>Hello World</title> </head>
<body>
<p>Hello, World!</p>
</body>
</html>
```

In the *helloworld.html* file, if `SAY_BYE` is not defined, preprocessing generates an error. If `SAY_BYE` is defined, the preprocessor includes the line `<p>Hello, World!</p>` in the body of the output of the HTML. This demonstrates how a Python preprocessor can be used to conditionally include blocks of a source file being processed.

5. ***View the Debug Output Tab***: Notice the HTML output and compare the result to the actual file *helloworld.html*.
6. ***View Rendered HTML***: On the right side of the Bottom Pane, click the ***HTML*** tab. The rendered HTML for the *helloworld.html* file is displayed in the Bottom Pane. Click the ***Output*** tab to return to the HTML code.
7. ***Create New File***: To create a new HTML file that will later contain the HTML code in the Bottom Pane, select ***File/New/New File***. In the New File dialog box, select the HTML Category. Click ***Open***.
8. ***Save the Output***: Delete the contents of the new HTML file tab in the Editor Pane, and then select the contents of the ***Output*** tab on the Bottom Pane. Copy the contents to the new HTML file tab in the Editor Pane. Select ***File/Save As*** to save the file with a unique name.
9. ***Specify Another Source File***: Go through steps 3 to 5 using the file *helloworld.py* in place of *helloworld.html*. Notice how the output displayed is now in Python, (for example, `print "Hello, World!"`). This demonstrates how the *preprocess.py* program can be used to process files written in different language types.

# Debugging the Program

In this step you will add breakpoints to the program and "debug" it. Adding breakpoints lets you run the program in sections, making it easier to watch variables and view the output as it is generated.

1. ***Set a breakpoint***: On the ***preprocessor.py*** tab, click on the gray margin immediately to the left of the code on line 347 of the program. This sets a breakpoint, indicated by a red circle.

2. ***Run the debugger***: Select ***Debug/Go/Continue*** (or enter 'F5', or use the Debug Toolbar). In the ***Script Arguments*** text box on the Debugging Options dialog box, enter the following source file and argument (if not there from a recent run): `-D "SAY_BYE" helloworld.html`. Click ***OK***.

***Komodo Tip***: Debugger commands can be accessed from the ***Debug*** menu, by shortcut keys, or from the Debug Toolbar. For a summary of debugger commands, see the Debugger Command List.

3. ***Watch the debug process***: Notice that the line where the breakpoint is set (line 347) turns pink. Also, a yellow arrow appears on the breakpoint. This arrow indicates the position at which the debugger has halted.
4. ***View variables***: On the Debug tab, click the ***Locals*** tab. If necessary, resize the pane by clicking and dragging the upper margin. On the ***Locals*** tab, notice the declared variables are assigned values. Examine the `infile` variable. This variable contains the name of the file specified above (*helloworld.html*).

***Komodo Tip***: What do the debugger commands do?

- ***Step In***: Executes the current line of code and pauses at the following line.
- ***Step Over***: Executes the current line of code. If the line of code calls a function or method, the function or method is executed in the background and the debugger pauses at the line that follows the original line.
- ***Step Out***: When the debugger is within a function or method, Step Out executes the code without stepping through the code line–by–line. The debugger stops on the line of code following the function or method call in the calling program.

5. ***Step In***: Select ***Debug/Step In*** until the debugger stops at line 129, the `preprocess` method. "Step In" is a debugger command that causes the debugger to enter a function called from the current line.
6. ***Set another breakpoint***: Click on the gray margin immediately to the left of the code in line 145 to set another breakpoint. Line 145 is where `getContentType` is called.
7. ***Run the debugger***: Select ***Debug/Go/Continue*** (or enter 'F5', or use the Debug Toolbar).
8. ***Step Over***: When line 145 is processed, the variable `contentType` is assigned the source file's (*helloworld.html*) type (HTML). "Step Over" is a debugger command that executes the current line of code. If the line of code calls a function or method, the function or method is executed in the background and the debugger pauses at the line that follows the original line.
9. ***View variables***: On the ***Debug*** tab, click the ***Locals*** tab. Examine the `contentType` variable. This variable contains the type of the source file; the type is "HTML" for *helloworld.html*.
10. ***Set another breakpoint***: Click on the gray margin immediately to the left of the code in line 197 to set another breakpoint. Line 197 is inside of the loop where the source file *helloworld.html* is being processed.
11. ***Run the debugger***: Select ***Debug/Go/Continue*** (or enter 'F5', or use the Debug Toolbar).
12. ***Add Watches for Variables***: On the ***Debug*** tab, click the ***Watch*** tab. Click the ***New*** button in the lower–right corner of the ***Debug*** tab. An ***Add Variable*** dialog box appears. In the ***Add Variable*** dialog box, enter `lineNum` in the text box. Click ***OK***. Notice that the `lineNum` variable and its value are displayed in the ***Watch*** tab. The `lineNum` variable is the line number of the line

currently being processed in the source file *helloworld.html*. Follow the above steps again to enter a watch for the variable `line`. The `line` variable contains the actual text of the line currently being processed.

13. **Run the debugger**: Select *Debug|Go/Continue* (or enter 'F5', or use the Debug Toolbar). Notice how the variables in the *Watch* tab change every time the debugger stops at the breakpoint set at line 197. Also, notice the output in the right side of the *Debug* tab. This output changes as new lines are displayed.

14. **Disable and Delete a breakpoint**: Click on the red breakpoint at line 197. The red beakpoint is now white with a red outline. This breakpoint is now disabled. Click on the disabled white breakpoint. This removes the breakpoint, but does not stop the debugger.

15. **Stop the Debugger**: On the *Debug* menu, click *Stop* or 'Shift'+ 'F5' (if the default key binding scheme is in effect).

# Explore Python with the Interactive Shell

In this step you will use the interactive shell to explore the `contenttype` module. The Komodo interactive shell helps you test, debug, and examine your program. See Interactive Shell for more information.

If starting this section of the tutorial with currently open Python shells, please follow the steps below to ensure the Python shell's current directory is the Python Tutorial directory.

1. **Close any Current Python Shells**: Click the "X" button, located in the upper–right corner of the *Shell* tab, for each open Python shell.
2. **Make preprocess.kpf the Active Project**: On the *Project* menu, select *Make Active Project|preprocess*.

Start using the interactive shell with the Python Tutorial project files:

1. **Start the Interactive Shell**: On the *Tools* menu, select *Interactive Shell|Start New Python Shell*. A *Python Shell* tab is displayed in the Bottom Pane.
2. **Import a Module**: At the ">>>" Python prompt in the interactive shell, enter: `import contenttype`
Notice that another ">>>" Python prompt appears after the import statment. This indicates that the `contenttype` module imported successfully.
3. **Get Help for a Module**: At the prompt, enter: `help (contenttype)`
The help instructions embedded in the *contenttype.py* file are printed to the interactive shell screen. This is useful for easily accessing Python documentation without installing external help files.
4. **Get Help for a Method in a Module**: At the prompt, press the up arrow to redisplay previously entered commands. When `help (contenttype)` is redisplayed, enter `.getContentType` at the end of the command. The entire command is as follows:
`help (contenttype.getContentType)`
Press *Enter*. The help instructions for the `getContentType` method are printed to the shell

screen. The ability to instantly access help on specific Python functions is a powerful use for the interactive shell.

5. ***Run a Method***: At the prompt, enter:
   ```
   contenttype.getContentType("helloworld.html")
   ```
   Notice the output identifies the file type as `HTML`.

6. ***Run Another Method***: At the prompt, enter:
   ```
   contenttype.getContentType("helloworld.py")
   ```
   Notice the output identifies the file type as `Python`.

7. ***Run a Final Method***: At the prompt, enter:
   ```
   contenttype.getContentType("test.txt")
   ```
   Notice the output identifies the file type as `Text`. The `contenttype` module uses several tests to determine the data type used within a file. The test that determined that *test.txt* is a text file simply analyzed the file extension.

# More Python Resources

## ASPN, the ActiveState Programmer Network

ASPN, the ActiveState Programmer Network, a source of numerous resources for Python programmers, including:

- Free downloads of ***ActivePython***, ActiveState's Python distribution.
- Searchable ***Python documentation***.
- The ***Python Cookbook***, a collaborative library of regular expressions for Python.

## Tutorials and Reference Sites

There are many Python tutorials and beginner Python sites on the Internet, including:

- The Python Home Page, the home of all that is Python.
- Python DevCenter, run by O'Reilly Networks, which provides access to tips, articles and other Python related items.

## Preprocessor Reference

The *preprocess.py* program in this tutorial is a simplified version of another Python *preprocess.py* script available via http://starship.python.net/crew/tmick/#preprocess. The version available on *starship.python.net* is an advanced portable multi–language file preprocessor.

# Tcl Tutorial

## Tcl Tutorial Overview

### Before You Start

This tutorial assumes:

- [ActiveTcl 8.3](#) or greater is installed on your system. ActiveTcl is a free distribution of the Tcl language.
- You are interested in Tcl. Previous knowledge of Tcl is not required as this tutorial walks you through a simple program. Further Tcl resources are suggested later.
- Komodo Professional is required for the [Editing the GUI](#) section of this tutorial. Komodo Professional includes the GUI Builder component, which is required for editing Tcl GUIs. To upgrade to Komodo Professional, see [ActiveState Komodo](#) for more information.

### Tcl Tutorial Scenario

1. *[Open the Tcl Tutorial project file](#)*
2. *[Use Tcl Editing Features](#)*

3. *[Edit the GUI](#)*, by making changes to the *dlg_tcl_check.ui* file. Komodo Professional is required for this section of the tutorial.
4. *[Add callback code](#)* by running the program.
5. *[Debug the program](#)* using the Komodo debugger.

# Opening the Tcl Tutorial Project

On *File* menu, click *Open/Project* and navigate to the *tcl_tutorial.kpf* project file on your file system (*install dir\Komodo x.x\samples\tcl_tutorials* in Windows and *install dir/Komodo x.x/samples/tcl_tutorials* in Unix). All files included in the tutorial project are displayed on the [Projects tab](#) in the Left Pane.

## Overview of the Tutorial Files

The following components are included in the *tcl_tutorial.kpf* project file:

- *dlg_tcl_check.ui*: The GUI builder project. GUI builder projects are stored with Komodo project files (*.kpf*) and carry a *.ui* file extension. This project contains two associated files, *dlg_tcl_check.tcl* and *dlg_tcl_check_ui.tcl*.
  - *dlg_tcl_check.tcl*: A callback file. Only this file should be edited. Generated automatically when [a new GUI Builder project](#) is created.

♦ ***dlg_tcl_check_ui.tcl***: This file is overwritten each time modifications are made to a GUI Builder project. For this reason, any changes made to this file could be lost. Generated automatically when a new GUI Builder project is created.

## Opening the Tcl Project File

On the Projects tab, click the plus sign next to *dlg_tcl_check.ui* to display its contents. Double−click the file *dlg_tcl_check.tcl*. The *dlg_tcl_check.tcl* file opens in the Editor Pane; a tab at the top of the pane displays its name.

# Using Tcl Editing Features

All Komodo editing features are available when programming in Tcl. The examples and simple exercises below provide an introduction to these features.

## Syntax Coloring

Komodo detects keywords in a Tcl program and applies coloring that makes it easier to quickly identify specific elements. For example, in *dlg_tcl_check.tcl*, on line 36, notice that the `set` command is a different color from the strings `normal` and `disabled`. Use the ***Fonts and Colors Preferences*** dialog box (***Edit|Preferences|Fonts and Colors***) to specify custom coloring for these and many other Tcl elements.

## AutoComplete and CallTips

Komodo helps to make programming faster by displaying available methods for Tcl commands. Komodo also displays the list of arguments that can be passed to a called function.

1. On a blank, uncommented line in the file *dlg_tcl_check.tcl*, enter the following:

   ```
   str
   ```

   After typing the "r", Komodo lists all methods beginning with `str`. Use the mouse or the up and down arrow keys to scroll through this list.

2. Continue typing until your entry reads:

   ```
   string is
   ```

   Notice that the list reduces to the available methods only. Move through the list using the mouse or the up and down arrow keys.

3. From the drop−down list, select *string is alnum*, and then press the *Tab* key. Komodo completes the rest of the method name. This is AutoComplete.

4. Type a space after `string is alnum`. Komodo lists the arguments for calling the string. This is CallTips.

## Background Syntax Checking

Komodo checks for syntax errors while you are programming. Komodo identifies syntax errors by displaying red and green wavy lines below the code. Syntax errors are underlined with a red wavy line; syntax warnings are underlined with a green wavy line. Notice the text typed in the previous step, `string is alnum`, is underlined with a red wavy line. Position your cursor on the red line to display a warning message on the status bar. See Background Syntax Checking in the Editor documentation for more information.

Delete `string is alnum`.

## Code Folding

Collapse and expand blocks of code to view or analyze code structure.

1. On line 93, click the "−" sign to the left of the Komodo Editor Pane to collapse this block of code. Notice that the "−" sign becomes a "+" sign.
2. Click the "+" sign. The block of text expands again.

# Editing the GUI

Use Komodo GUI Builder features to add or remove widgets, or to modify the properties of existing widgets. Komodo Professional is required for this section of this tutorial. Komodo Professional includes the GUI Builder component, which is required for editing Tcl GUIs. To upgrade to Komodo Professional, see ActiveState Komodo for more information.

## Opening a GUI Builder Project

GUI builder projects are stored with Komodo Project files and carry a *.ui* file extension. The project file used in this tutorial is *dlg_tcl_check.ui*. To open the project:

1. Open the Tcl Tutorial Project, if it is not already open.
2. On the *Projects* tab, right−click *dlg_tcl_check.ui*, and select *Edit Dialog*. The GUI Builder application launches, and the dialog box is displayed in the GUI Builder workspace.

# Viewing Project Properties

When creating a new GUI Builder project, you are prompted to specify a target language and version. Different widgets sets are displayed on the Widget Palette, depending on the language and version selected. View properties of a project by selecting **Project Settings** from the GUI Builder's **File** menu.
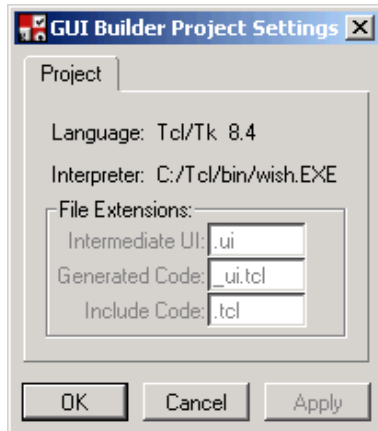


# Adding Widgets to a Dialog

Widgets are added to a dialog from the **Palette** tab or the **Menu** tab. In this step, you will add label widgets to the GUI.

1. On the **Palette** tab, click the **label** widget.
2. Click in the cell in the first column of the fifth row. The label widget is added.
3. On the **Palette** tab, click the **label** widget again.
4. Click in the cell in the third column of the fifth row. Another label widget is added.

## Resizing Widgets

Resizing widgets can involve spanning multiple columns, rows, or changing column or row width or height. To resize the label widgets:

1. Click the label widget in the first column.
2. Position the mouse pointer at the right of the label such that the pointer changes to an arrow pointing toward a line, and then drag to the right to expand the label widget across two cells.
3. Repeat for the label widget in the third column.



## Editing Widget Properties

Widget properties define the display characteristics of individual widgets. Common widget properties are accessible on the GUI Builder toolbar; the complete set of properties is accessible by double−clicking the widget.

1. Click the label widget that spans the first two columns. In the *Text* field on the toolbar, enter "Now Playing:", and click the "B" (Bold) button to the right.
2. Double−click the label widget that spans the last two columns. Remove the contents of the *−text* field. (This is the same as clearing the contents in the *Text* field on the toolbar.) Notice that the value of the *−text* field is, by default, the same as the name of the widget (as shown in the field at the top of the properties page). The widget name itself is used in the callback code you will add in the next step.
3. Click *Apply*.
4. Click *OK*.
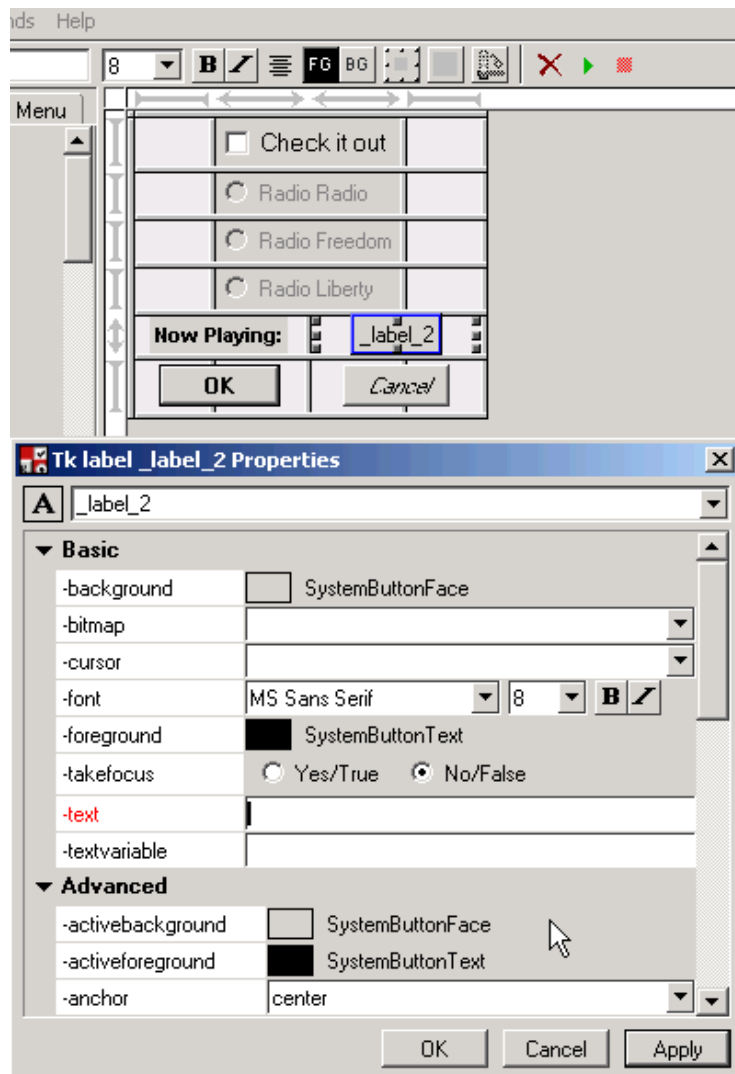
## Build the GUI

Use the GUI Builder test feature to generate output and preview the user interface.

To build the project:

1. Select **Commands/Start Test**, or click the green button on the GUI Builder toolbar.
2. A dialog box prompts you to save the project. Click **Save**.
3. The GUI is generated. Review the results.

# Adding Callback Code

Callback code assigns programmatic actions to the components configured in the GUI. In this step, you will add code that displays a different message in the second label depending on which radio button is selected.

Each dialog project in Komodo contains two files. One file (with a *_ui* suffix in the file name, in this case *dlg_tcl_check_ui.tcl*) is regenerated each time the GUI Builder is run. Therefore, user code should never be placed in this file. The other file (without the *_ui* suffix, *dlg_tcl_check.tcl*) is used to store the user code, which is preserved regardless of future edits to the dialog.

## Open the Program File

In the GUI Builder, select *Commands/View Code*. The *dlg_tcl_check.tcl* file opens in the Komodo Editor Pane.

## Adding Code to the Radio Buttons

In this section, you will add code that displays a different message in the label widget depending on radio button selection.

1. On line 49, insert a line after the opening brace and enter the following:

```
    variable BASE
    $BASE._label_2 configure -text "Free Bird"
}
```
2. On line 61, insert a line after the opening brace and enter the following:

```
    variable BASE
    $BASE._label_2 configure -text "The Liberty Sessions"
}
```
3. On line 73, insert a line after the opening brace and enter the following:

```
    variable BASE
    $BASE._label_2 configure -text "Radio Killed the Video Star"
```

```
}
```

Note that the text is associated with the widget by specifying the widget name in the
`$BASE._label_2` statement, where `_label_2` is the name of the widget. To determine the name of
a widget, double−click the widget in the GUI Builder and refer to the field at the top of the properties
page.

# Debugging the Program

This section reviews how to add breakpoints to the program and "debug" it. Adding breakpoints lets you
to run the program in parts, making it possible to watch variables and view output as they are generated.
Before you begin, be sure that line numbers are enabled (*View/View Line Numbers*). Open
*dlg_tcl_check.tcl* in the Komodo editor (if it is not already open).

1. *Set a breakpoint*: Click in the gray margin immediately to the left of the code on line 113 of the
   program. This sets a breakpoint, indicated by a red circle.
2. *Run the debugger*: Select *Debug/Start* (or <F5>, or use the Debug Toolbar). In the Debugging
   Options dialog box, click *OK* to accept the defaults. The debugger processes the program until it
   encounters the first breakpoint.

*Komodo Tip*: Debugger commands can be accessed from the *Debug* menu, by shortcut keys, or from the
*Debug Toolbar*. For a summary of debugger commands, see the Debugger Command List.

3. *Watch the debug process*: Notice that a yellow arrow appears on the breakpoint. A yellow arrow
   indicates the position at which the debugger has halted.
4. *Line 113: Step In*: Select *Debug/Step In*. "Step In" is a debugger command that causes the
   debugger to execute the current line and then stop at the next processing line. In this case, the
   debugger opens the GUI definition file (*dlg_tcl_check_ui.tcl*). This file initializes the graphical
   elements in the applications.
5. *Line 23: Step Out*: Select *Debug/Step Out* to run the rest of the code in *dlg_tcl_check_ui.tcl*. The
   debugger stops on line 118 in *dlg_tcl_check.tcl*.
6. *View variables*: In the Bottom Pane, click the *Debug* tab. On the left side of the Bottom Pane,
   click the *Global* tab. Right−click the `$example_radiobutton` variable, and then select *Add
   to Watch*.
7. *View watched variables*: On the *Watch* tab, notice the `$example_radiobutton` variable is
   listed.
8. *Line 118: Step In*: Click the *Global* tab. Select the `$example_radiobutton` variable. Select
   *Step In*. Line 118 launches the Tcl application. Note that the debugger appears to be inactive (for
   example, no yellow arrow indicates the current debugger position). The debugger is waiting for
   action from the application.
9. *Click "Check it out"*: Click the "Check it out" check box in the application. The debugger
   moves to line 35.
10. *Set a breakpoint*: Click the gray margin on line 39 to set a breakpoint.
11. *Line 35: Run* Select *Debug/Go/Continue*. The debugger stops on line 39.

12. *Line 39: Step In*: Select *Debug/Step In*. Focus returns to the application. Click the *Radio Liberty* radio button. The debugger moves to line 62. Notice that the `$example_radiobutton` variable on the *Watch* tab now has the value "liberty". This association is defined in the *dlg_tcl_check_ui.tcl* file.
13. *Line 62: Step Into*: Select *Debug/Step In* until line 63 is executed. This returns focus to the application, and writes "The Liberty Sessions" text to the label.
14. *Click OK*: On the application, click *OK*. This stops the debugger.

*Komodo Tip*: What do the debugger commands do?

- *Step In*: Executes the current line of code and pauses at the following line.
- *Step Over*: Executes the current line of code. If the line of code calls a function or method, the function or method is executed in the background and the debugger pauses at the line that follows the original line.
- *Step Out*: When the debugger is within a function or method, *Step Out* executes the code without stepping through the code line by line. The debugger stops on the line of code following the function or method call in the calling program.

# More Tcl Resources

## ASPN, the ActiveState Programmer Network

ASPN, the ActiveState Programmer Network, provides extensive resources for Tcl programmers:

- Free downloads of *ActiveTcl*, ActiveState's Tcl distribution
- Searchable *Tcl documentation*
- Trial versions of the *Tcl Dev Kit*

## Documentation

There is a wealth of documentation available for Tcl. The first source for language documentation is the ActiveTcl distribution installed on your system. The "doc" directory contains the ActiveTcl User Guide and Documentation Index. To view ActiveTcl documentation:

- Locate the ActiveTcl distribution on your system (by default, *C:\Tcl\doc* on Windows)
- Double−click to open the file *ActiveTclHelp.chm*.

## Tutorials and Reference Sites

There are many Tcl tutorials and beginner Tcl sites on the Internet, for example:

- Tcl Developer Xchange
- The Tcl'ers Wiki

# XSLT Tutorial

## XSLT Tutorial Overview

### Before You Start

This tutorial assumes:

- You are interested in XSLT. Previous knowledge of XSLT is not required for this tutorial. The XSLT Tutorial walks you through a simple program and later suggests various resources for further information.

### XSLT Tutorial Scenario

In the Perl Tutorial, a Perl program converts a text file containing exported email messages to an XML file. In this tutorial, XSLT converts the XML file to HTML. Note that you do not need to complete the Perl Tutorial before doing the XSLT Tutorial. Each tutorial can be completed independently. In this tutorial you will:

1. ***Open the XSLT Tutorial Project*** and associated files.
2. ***Analyze mailexport.xml*** the XSLT program included in the XSLT Tutorial Project.
3. ***Run the Program*** and generate HTML output through the transformation.
4. ***Debug the program*** using the Komodo debugger.

## Opening the Tutorial Project

On *File* menu, click *Open|Project* and navigate to the *xslt_tutorial.kpf* project file on your file system (*install dir\Komodo x.x\samples\xslt_tutorials* in Windows and *install dir/Komodo x.x/samples/xslt_tutorials* in Unix). All files included in the tutorial project are displayed on the Projects tab in the Left Pane.

### Opening the XSLT Tutorial Files

On the Projects tab, double−click the files *mailexport.html*, *mailexport.xml*, and *mailexport2html.xsl*. These files open in the Editor Pane; a tab at the top of the pane displays each of their names.

### Overview of the Tutorial Files

- *mailexport.xml*: An input file that contains email messages converted to XML format. (See how this was done in the Perl Tutorial.)
- *mailexport2html.xsl*: An XSLT program that generates an HTML file from the *mailexport.xml* input file.

- *mailexport.html*: A file that stores the HTML output from the XSLT transformation.

# Analyzing the Program

In this step, you will analyze the XSLT program on a line–by–line basis. Open the XSLT Tutorial Project and associated files as described in the previous step. Be sure Line Numbers are enabled in Komodo (*View/View Line Numbers*). Be sure the *mailexport2html.xsl* file is displayed in the Komodo Editor Pane.

## XSLT Header

### Lines 1 to 3 – XML and XSLT Declarations

- an XSLT program is an XML document – thus, the XML version and character set are declared on the first line
- the namespace declaration on the second line tells the "parser" (the XSLT interpreter) that XSLT elements are prefixed with `xsl:` to prevent confusion with user–defined element names and non–XSLT elements
- `xsl:output` controls the appearance of the generated output; for example, the presence of this line generates a `META` declaration in the head of the HTML output

*Komodo Tip*: Notice that different types of language elements are displayed in different colors. Adjust the display options for language elements in the Preferences dialog box.

*XSLT Pointer*: Processing routines in XSLT programs are enclosed in opening and closing tags similar to those in XML.

## HTML Header

### Line 6 – XSLT "template"

- `template` is the main processing element in an XSLT program
- the `match="/"` attribute specifies that the template is selected when the document element is processed

*XSLT Pointer*: XSLT commands have (up to) four components: namespace ("xsl"), element ("template"), attribute(s) ("match="), and attribute value(s) ("/").

*XSLT Pointer*: XSLT uses XPath expressions to select data from XML documents. On line 6, `match="/"` selects a "node" in the XML document's hierarchy, rather than a specific item of data.

**Lines 7 to 11 – HTML Tags**

- writes standard HTML tags to the output document

**Line 12 – XSLT apply–templates**

- processes each node of the XML document (that is, each sub–section contained beneath the current position in the XML document)
- for each node, the XSLT "engine" (the internal processor) checks the XSLT program for a matching template
- the first XML tag with a corresponding template is `<EMAIL>`

**Lines 13 to 15 – HTML Tags**

- after processing all the nodes in the XML document, processing returns to line 13, where the closing tags for the HTML page are written to the output
- line 15 closes the XSLT processing routine, completing the program

# Format Email Header

### Lines 18 to 21 – Select HEADER content

- when line 18 is processed, content in `<HEADER>` tags in the XML document are processed
- lines 19 and 21 write standard HTML formatting around the content generated in line 20
- on line 20, the `value-of` statement selects content contained in the `<SUBJECT>` tag and writes it to the output document

*Komodo Tip*: Click the minus symbol to the left of line 19. The entire section of nested code is collapsed. This is called Code Folding.

### Lines 22 to 29 – call–template

- after the `From:` text, the `call-template` routine causes the XSLT program to proceed to the template `formatEmail` on line 51; after completing the `formatEmail` routine, processing returns to line 23
- `with-param` indicates that the parameter `address` should be applied to the contents of the `<ORIGADDRESS>` XML tag
- the same selection and formatting routine is applied to the contents of the `<DESTADDRESS>` XML tag on lines 26 to 28

*XSLT Pointer*: Notice the `<BR/>` HTML tag on line 25. XML and XSLT treat all tags as container tags that have both opening and closing elements. However, some HTML tags (like `<BR>` and `<IMG>`) stand alone, and do not require a closing tag. They are represented with a closing slash. XSLT tags also use a closing slash if they are not a tag pair (as shown on line 23).

## Process Email

### Lines 33 to 34 – Process First Message

- when the `apply-templates` tag in line 12 is encountered, processing jumps to line 33
- on line 34, the `HEADER` node is selected and processing jumps to line 18

*XSLT Pointer*: Comments in XSLT programs are enclosed in the tags `<!--` and `-->`, the same as in HTML.

### Lines 36 to 39 – Process Email Body

- after processing the email header, the XSLT program proceeds to line 36
- the contents of the `BODY` tag are placed in the HTML tags

*Komodo Tip*: XSLT programs and XML input documents must be "well–formed" in order to perform transformations. Komodo's Background Syntax Checking makes it easy to identify and fix coding errors.

## Format Email Addresses

### Lines 45 to 52 – Format Email Addresses

- the routine that starts on line 47 is called from lines 22 and 26
- `address` parameter contents are determined on lines 23 and 27
- on line 49, the contents of the `address` parameter are converted to a variable and concatenated with the text that constitutes a valid email address reference in HTML

# Running the Program

To start, generate program output by running the program through the debugger without setting any breakpoints.

1. *Assign XML input file*: On the **Debug** menu, click **Go/Continue** (or 'F5'). In the Debugging Options dialog box, specify *mailexport.xml* as the XML input file. Use the **Browse** button to navigate to the directory containing the XSLT tutorial project files.
2. *Run the debugger*: Click **OK** to run the debugger.
3. *Stop the debugger*: From the **Debug** menu, select **Stop** to end the debugging process.
4. *View Debug Output*: Notice the messages displayed on the status bar in the bottom left corner of the screen; these indicate the debugger status. The results of the transformation are displayed on the **Debug** tab.
5. *View the Output as HTML*: On the right side of the Bottom Pane, click the **HTML** tab. The rendered HTML is displayed in the Bottom Pane. Click the **Output** tab to return to the HTML code.

6. ***Create New File***: To create a new HTML file that will later contain the HTML code in the Bottom Pane, select ***File/New/New File***. In the New File dialog box, select the HTML Category. Click ***Open***.
7. ***Save the Output***: Delete the contents of the new HTML file tab in the Editor Pane, and then select the contents of the ***Output*** tab on the Bottom Pane. Copy the contents to the new HTML file tab in the Editor Pane. Select ***File/Save As*** to save the file with a unique name.

# Debugging the Program

This section reviews how to add breakpoints to the program and "debug" it. Adding breakpoints lets you to run the program in parts, making it possible to watch variables and view output as they are generated. Before beginning, be sure that line numbering is enabled in Komodo (***View/View Line Numbers***).

1. ***Step In/Assign the XML input file***: If necessary, click on the *mailexport2html.xsl* tab in the editor. From the menu, select ***Debug/Step In*** (or <F11>). In the Debugging Options dialog box, specify *mailexport.xml* as the XML input file (unless the input file was assigned in the previous step). Assigning the XML input file to the XSLT program file selects the XML file as the default input file when running the transformation.
2. ***Start Debugging***: In the Debugging Options dialog box, click ***OK*** to start debugging.

***Komodo Tip***: Debugger commands can be accessed from the ***Debug*** menu, by shortcut keys, or from the ***Debug Toolbar***. For a summary of debugger commands, see the Debugger Command List.

3. ***Watch the debug process***: A yellow arrow on line 6 indicates the position in the XSLT file where the debugger has halted. Another yellow arrow on line 1 in the XML file indicates the processing point in the input file.
4. ***View Debug tab***: In the Bottom Pane, click the ***Debug*** tab. On the right side of the ***Debug*** tab, click the ***Call Stack*** tab. On the ***Call Stack*** tab, notice that the current call stack is the template in line 6 of the XSLT program.
5. ***Set a breakpoint***: On the ***mailexport2html.xsl*** tab in the Editor Pane, click the gray margin immediately to the left of the code on line 12. This sets a breakpoint, indicated by a red circle.

***Komodo Tip***: Breakpoints can be set at any time. An enabled breakpoint is a solid red circle. A disabled breakpoint is a white circle with a red outline. Click once in the gray margin to enable a breakpoint. Click an enabled breakpoint once to disable it.

6. ***Line 6: Step Out***: Select ***Debug/Step Out***. The debugger runs until it encounters the breakpoint on line 12; if no breakpoint had been set, the debugger runs until the end of the program.
7. ***Line 12: Step In***: Click ***Debug/Step In***. Notice the debugger jumps to line 33 of the XSLT program, and advances the pointer in the XML file to line 4. When the debugger processed line 12 (xsl:apply-templates), it looked for a template that matched the top node in the XML document (<EMAILCOMMENTS>). When no matching template was found, it proceeded to the next node in the XML document (<EMAIL>) and found a matching template on line 33.

8. *View the Debug tab*: The HTML tags on lines 7 to 11 of the XSLT program are written to the Bottom Pane.
9. *View the Debug Variables tab*: Notice that the *Call Stack* tab displays the current template; previous templates can be selected from the drop−down list.
10. *Line 33: Step In*: Use the *Step In* command until the current−line pointer in the XSLT file is on line 20.
11. *Line 20: Step In*: Watch the Bottom Pane as you Step In line 21. The `xsl:value-of` statement selects the contents of the `<SUBJECT>` field on line 9 of the XML file and places it within the HTML tags on lines 19 and 21.
12. *Line 21: Step In*: Line 22 calls the template `formatEmail` on line 45. Continue to step in until line 49 is processed. The `formatEmail` template is processed with the `address` parameter on line 46. This routine processes the contents of the `<ORIGADDRESS>` node in the XML document. In order to generate the hyperlink in the output HTML document, lines 48 and 49 concatenate the string `mailto:` with the contents of the `<ORIGADDRESS>` node.
13. *Stop the Debugger*: On *Debug* menu, click *Stop* to end debugger processing.

# More XSLT Resources

## ASPN, the ActiveState Programmer Network

ASPN, the ActiveState Programmer Network, hosts the *XSLT Cookbook*, a collaborative library of XSLT code.

## Documentation

The W3C (World Wide Web Consortium) specifications are available online:

- XSLT
- XPath
- XML

# Tutorials and Reference Sites

There are many XSLT tutorials and beginner XSLT sites on the Internet, including:

- xml.com's What is XSLT?
- free tutorials at W3Schools.com

# Run Command Tutorial

## Run Command Tutorial Overview

### Before You Start

This tutorial assumes:

- You are interested in running external commands from within Komodo.
- You are not running Windows 98 or ME. Running interactive commands (especially `command.com`) on Win98/ME through the Komodo Run Command feature can cause Komodo to hang. Therefore, the Run Command Tutorial is not supported on Windows 98 or ME.

### Run Command Tutorial Scenario

This tutorial introduces you to the Komodo Run Command feature. You will learn how to run simple and complex custom commands (such as `grep`, `make`, and `perl`); use these commands to process and analyze files; save commands to run with a single keystroke; and use commands to make Komodo a more powerful editor. In this tutorial you will:

1. *Run simple commands* using the Komodo Run Command feature.
2. *Use advanced command options* to control how and where a command is run.
3. *Save commands* in the Toolbox and assign keyboard shortcuts.
4. *Use command shortcuts* to customize commands for reuse.
5. *Use command query codes* to have your commands prompt you for information before running.
6. *Parse command output* into a list of results by specifying a regular expression.

### Opening the Tutorial Project

From the *File* menu, click *Open/Project* and navigate to the *runcmd_tutorial.kpf* project file on your file system (*<install dir>\Komodo x.x\samples\runcmd_tutorials* in Windows and *<install dir>/Komodo x.x/samples/runcmd_tutorials* in Unix). All files included in the tutorial project are displayed on the Projects tab in the Left Pane.


# Running Simple Commands

## Hello, World!

The Komodo Run Command feature offers another way to run commands that would otherwise be run on the system command line. This section starts with a simple `echo` command.

1. Select *Tools/Run Command* to open the Run Command dialog box.
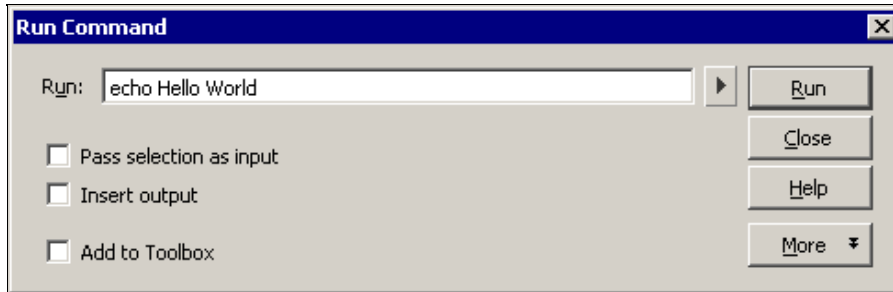
2. In the **Run** field, enter `echo Hello World`.
3. Click **Run**. The results are displayed on the **Command Output** tab.

## Command Output Tab

Output from commands is displayed on the **Command Output** tab.
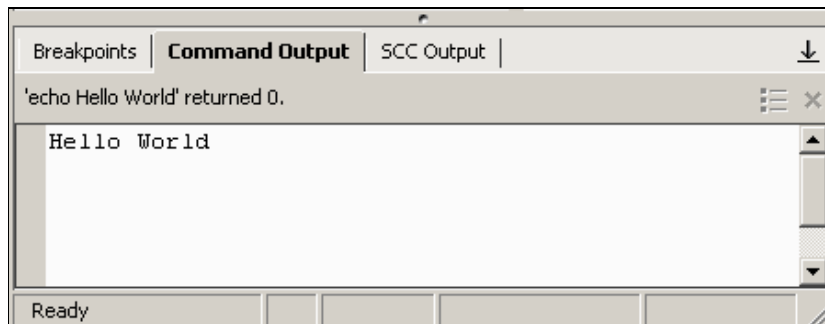


Use the **Command Output** tab to interact with commands; if the command accepts input, enter it directly into the command on the **Command Output** tab. The **Command Output** tab has the following features:

- Output written to `stderr` (standard error output) is displayed in red at the top of the **Command Output** tab.
- To terminate a running command, click the ✖ button in the upper right–hand corner of the tab.
- Many keyboard shortcuts available in the Komodo editor can also be executed on the **Command Output** tab. For example, 'Ctrl'+'Shift'+'8' displays white space and 'Ctrl'+'Shift'+'7' displays line endings (if the default key binding scheme is in effect).

The **Toggle Raw/Parsed Output View** button ⋮≡ is discussed in the Parsing Command Output section of this tutorial.

## Inserting Command Output

Insert command output into a document using the *Insert output* option.

1. On the Projects tab, double−click the file *play.txt*. The file opens in the Editor Pane; a tab at the top of the pane displays its name.
2. Select *Tools/Run Command*.
3. In the *Run* field, enter the command `dir` (on Windows) or `ls -al` (on Linux).
4. Select the *Insert output* check box, and then click *Run*. The contents of Komodo's current directory are inserted into *play.txt*.

## Filtering Parts of a Document

The *Pass selection as input* option passes selected text to the specified command. Use this option together with the *Insert output* option to filter selected regions of a document.

1. Open *play.txt* from the Run Command tutorial project (if it is not already open).
2. Select all six lines containing the word `frog`.
3. Select *Tools/Run Command*.
4. In the *Run* field, enter the command `sort` (on Windows) or `sort -n` (on Linux).

    Note that the *Pass selection* as input and *Insert output* options are selected automatically. If one or more lines are selected in a document, the Run Command expects to filter the selected lines.

5. Click *Run* to sort the list of frogs.

Use the `grep` command line utility to filter lines of text. Use `grep` to filter out all but the red frogs from the list.

This tutorial assumes the `grep` utility is installed on your system and is in your system's PATH. Grep is a Linux utility that searches for text and characters in files. Windows operating system users may not have a `grep` installation. There are a number of free versions available on the Web. Search using the keywords `grep for Windows`.

1. Open *play.txt* from the Run Command tutorial project (if it is not already open).
2. Select the "5 red frogs" and "6 green frogs" lines.
3. Select *Tools/Run Command*.
4. In the *Run* field, enter the command `grep red`.
5. Click *Run* to remove all but the red frogs.

# Using Advanced Options

Clicking the *More* button in the Run Command dialog box reveals a number of advanced options.
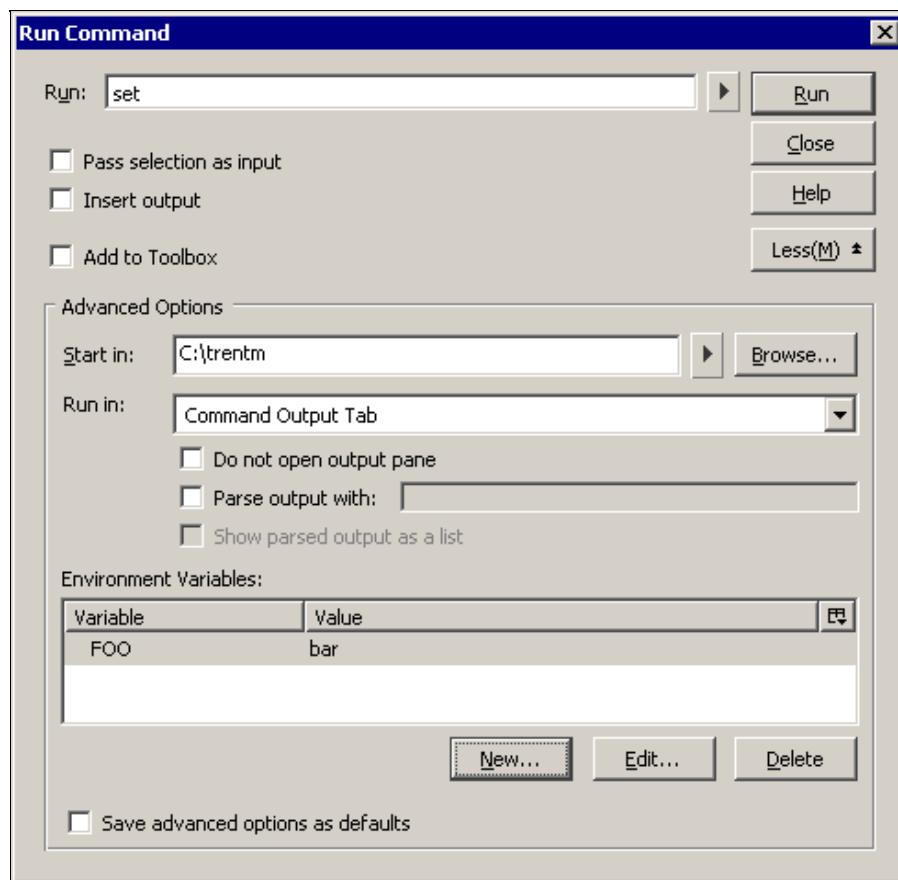
## Specifying a Command's Working Directory

To set the current working directory for a command:

1. Select *Tools/Run Command*. Click *More* to display Advanced Options.
2. In the *Run* field, enter the command: dir (on Windows), or ls -al (on Linux).
3. In the *Start in* field, enter *C:\* (on Windows), or */home* (on Linux).
4. Click *Run* to generate a *C:\* directory listing.

## Specifying Environment Variables

Specify which environment variables to set for a command. For example, use this feature for setting PERL5LIB or PYTHONPATH when running Perl or Python scripts.

1. Select *Tools/Run Command*.
2. In the *Run* field, enter the command: set.
3. Click *New...* to add a new environment variable. For the variable name, enter: PERL5LIB
4. Click *Add Path...* to choose a value for PERL5LIB (the actual value you choose does not matter for this example). Click *OK*.

5. Click *Run* to display all environment variables. Scroll through the results on the *Command Output* tab until the `PERL5LIB` setting is located.

## Running GUI Apps or Running Commands in a Console

Run GUI programs outside of the Command Output tab by changing the *Run in* option to *No Console*.

1. Select *Tools/Run Command*.
2. In the *Run* field, enter the command: `mozilla`
   If the Mozilla browser is not installed on your system, choose another GUI application to run.
   For example, on Windows, try running either the `iexplore` or `notepad` command.
3. From the *Run in* drop–down list, select *No Console (GUI Application)*.
4. Click *Run* to open the GUI application rather then the *Command Output* tab.

To run commands in a new console window:

1. Select *Tools/Run Command*.
2. In the *Run* field, enter the command: `dir`
3. From the *Run in* drop–down list, select *New Console*.
4. Click *Run* to execute the command and open a new console window.

# Saving and Rerunning Commands

Save frequently used commands for quick access and reuse.

## Rerunning Recent Commands

Select *Tools/Recent Commands* to rerun recently run commands.
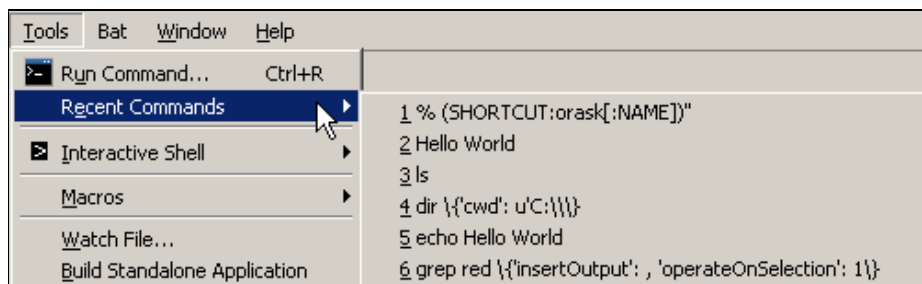


## Saving Commands in the Toolbox

The Run Command dialog box contains an option for saving commands in the Toolbox for reuse. A command saved in the Toolbox is indicated with the ■ icon.
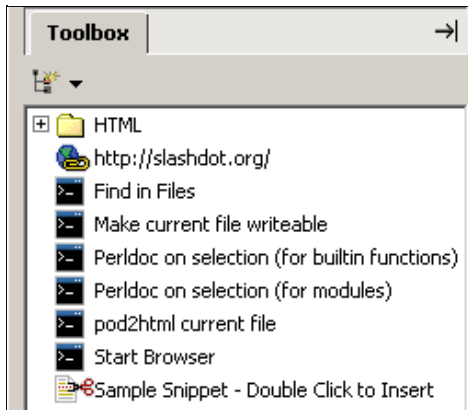
1. Select *Tools/Run Command*.
2. In the *Run* field, enter the command: `echo Hello World`
3. Select the *Add to Toolbox* check box.
4. Click *Run*. Notice that a command named `echo Hello World` is added to the Toolbox.
5. Double–click the  icon next to `echo Hello World` to rerun the command.

## Saving Commands in a Project

Commands can also be stored in a Komodo Project.
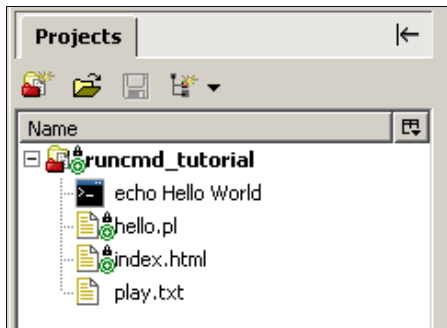


For example, the Run Command Tutorial project includes a Command to echo "Hello World".

Consider adding commands to your projects to run `make` or other command line tools. There are two ways to add commands to a project:

- Right click on a project and select *Add New Command...*.
- Drag and drop a command from the Toolbox tab or another open project onto the Projects tab.

## Editing Saved Command Properties

Edit command properties in the Command Properties dialog box.
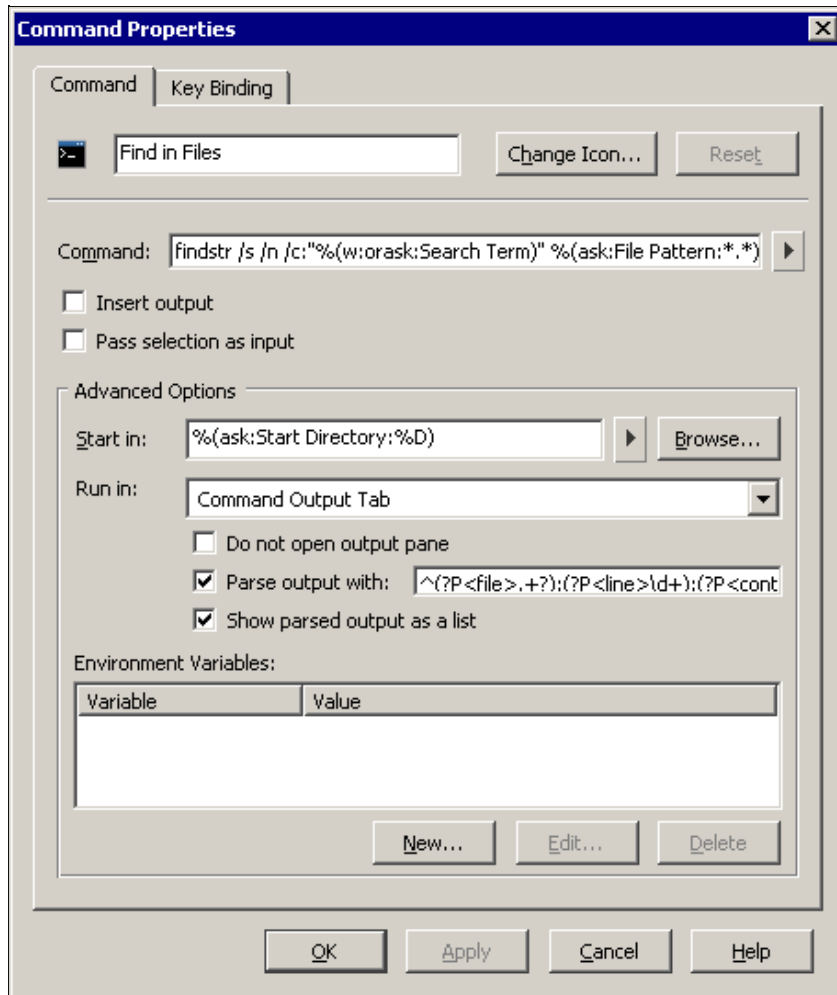
To open this dialog box, right click on any saved command and select *Properties*.

# Using Command Shortcuts

Run Command shortcuts insert interpolation codes for filenames, directory names, paths and other arguments into commands as variables. This creates commands that are more generic and useful. Enter command shortcuts in the Run and Start in fields, or select them from the drop–down lists to the right of the Run and Start in fields. Windows users should enclose all Komodo shortcuts (with the exception of `%(browser)`) in double quotation marks to ensure that spaces in filenames or file paths are interpreted correctly.

Click the arrow button to the right of the *Run* field to view a list of Run Command shortcuts.

```
%% : escaped percent sign
%f : file base name
%F : file path
%d : directory base name of file
%D : directory path of file
%P : path of the current file's project

%L : current line number
%s : selection
%S : URL-escaped selection
%w : selection or word under cursor
%W : URL-escaped selection or word under cursor

%(browser) : Configured browser
%(perl) : Configured Perl Interpreter
%(php) : Configured PHP Interpreter
%(python) : Configured Python Interpreter
%(tclsh) : Configured tclsh Interpreter
%(wish) : Configured wish Interpreter

%(ask) : Ask when command is run

Help on Shortcuts...
```

## Shortcuts for the Current File

The string %F in a command expands the full path of the current file.

1. On the *Projects* tab, double−click the file *play.txt*. The file opens in the Editor Pane; a tab at the top of the pane displays its name.
2. Select *Tools/Run Command*.
3. In the *Run* field, enter the command:
   `echo "%F"`
4. Click *Run*.

Change the current file status from "writable" to "read−only".

1. Open *play.txt* (if it is not already open).
2. Select *Tools/Run Command*.
3. In the *Run* field, enter the command:
   `attrib +R "%F"`
   on Windows, or:
   `chmod u+w "%F"`
   on Linux.
4. Click *Run*. The result is displayed at the top of the *Command Output* tab.

To open a current HTML file in a Web browser, combine %F with the %(browser) shortcut.

1. On the *Projects* tab, double−click the file *index.html*.
2. Select *Tools/Run Command*.

3. Click the arrow to the right the **Run** field to display the shortcuts drop−down list. Select **%browser**, press the space bar, and then select **%F**. Enclose the `%F` in double quotation marks.
4. From the **Run in** drop−down menu, select **No Console (GUI Application)**.
5. Click **Run**.

## Shortcuts for the Current Selection

The %s, %S, %w and %W codes insert current selections, or the current word under the cursor, into commands. This shortcut helps when running utilities like `grep`, or for searching the Web.

1. On the **Projects** tab, double−click the file *index.html*.
2. Position the cursor over the word "PHP" in *index.html*.
3. Select **Tools/Run Command**.
4. In the **Run** field, enter the command:
   `%(browser) http://www.google.com/search?q="%W".`
5. Select the **Add to Toolbox** check box to save this command.
6. Click **Run** to search for "PHP" with Google.

Now that you have searched for a word or selection in Google, try the following shortcut to search for PHP methods.

1. Open *index.html*.
2. Select the text `mysql_info methods` in the file.
3. Select **Tools/Run Command**.
4. In the **Run** field, enter the command "`%(browser)`
   `http://www.php.net/manual-lookup.php?pattern=%S`".
5. Select the **Add to Toolbox** check box to save this command.
6. Click **Run** to search `mysql_info methods` in PHP's online manual.

These two commands are built into Komodo. 'Ctrl'+'F1' (if the default key binding scheme is in effect) starts a Google search for the current selection. 'Shift'+'F1' in a Perl, Python or PHP file starts a help search appropriate for that language. Customize searches in the Preferences dialog box (**Edit/Preferences/Language Help**).

## Using Shortcuts for a Command's Directory

Run commands from the directory where the current file is stored, rather then the current directory. For example, use the command `%(perl) "%F"` to run the current file with a configured Perl interpreter.

1. On the **Projects** tab, double−click the file *hello.pl*.
2. Select **Tools/Run Command**.
3. In the **Run** field, enter the command:
   `%(perl) "%F"`
4. In the **Start in** field, enter: "`%D`"
5. Click **Run**.

This example assumes a perl interpreter is configured on your system. If a perl interpreter is not configured (the required file is perl.exe), an error message displays at the top of the *Command Output* tab. Alternatively, run the command `dir` (Windows) or `ls` (Linux) to display a list of files and folders beneath the current directory.

# Using Command Query Shortcuts

## Introduction

Run Command query shortcuts prompt for command input data via a dialog box. These queries can be configured with default values and/or prompt the user if no value could be determined automatically (e.g. a command to search Google for the current selection that prompts for a search term if nothing is selected).

The `%(ask)` shortcut always prompts the user for data. Other shortcuts can use the `orask` modifier to prompt the user if no valid value could be determined.

Windows users should enclose all Komodo shortcuts (with the exception of `%(browser)`) in double quotation marks. This is necessary to ensure that any spaces in filenames or file paths are interpreted correctly.

## Always Prompting with %(ask)

The full syntax of the `%(ask)` shortcut is:
`"%(ask[:NAME:[DEFAULT]])"`
where `NAME` is an optional name to use when prompting in the dialog box and `DEFAULT` is an optional default value to place in the dialog box.

For example:

1. Select *Tools/Run Command*.
2. In the *Run* field, enter the command: `echo Your favorite number is "%(ask)"`
3. Click the *Run* button to run the command. The Interpolation Query dialog box is displayed.
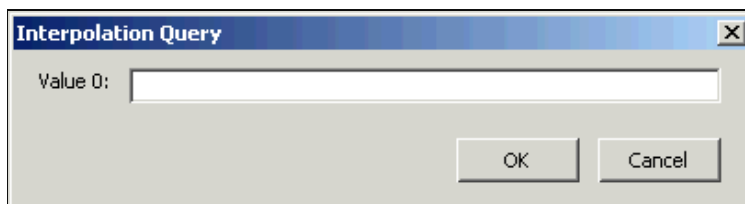


4. Enter your favorite number and click *OK* to finish running the command.

Refine this shortcut by adding a more meaningful name than "Value 0" and a more appropriate default value.

1. Select *Tools/Run Command*.
2. In the *Run* field, enter the command: `echo Your favorite number is "%(ask:Fav Number:42)"`
3. Click the *Run* button to run the command. The Interpolation Query dialog box will now look like this:
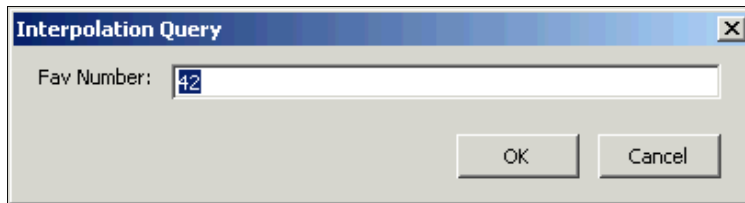


4. If your favorite number does not happen to be 42, enter a different number and click *OK* to finish running the command.

## Prompting When Necessary with %(...:orask)

Any Run Command shortcut can be modified to prompt the user for a value if one cannot be determined automatically. The full syntax of the modified shortcut is:
`"%(SHORTCUT:orask[:NAME])"`
where NAME is an optional name to use when prompting in the dialog box.

In the previous step we created a shortcut to search for the selected word on Google with the command:
`%(browser) http://www.google.com/search?q="%W"`
However, if nothing has been selected and there is no word under the cursor, the command fails. In this case, it would be better if the command prompted you for a search term.

1. Be sure your cursor is *not* positioned over a word.
2. Select *Tools/Run Command*.
3. In the *Run* field, enter the command:
   `%(browser) http://www.google.com/search?q="%(W:orask:Search for)"`
4. Click *Run*. The Interpolation Query dialog box prompts for a search term.

# Parsing Command Output

## Introduction

Use Run Commands to specify a regular expression to parse filename and line number information from lines of output. The parsed results are displayed in a table, to quickly identify the desired file. Explore this usage by creating a "Find in Files" command later in this section.

# Parsing with a Regular Expression

To parse command output, specify a regular expression. Each line of output from the command sent to the *Command Output* tab is also passed through the regular expression. If the regular expression matches, an entry is added to the list of parsed results.

Python's regular expression syntax is used to parse command output. Its regular expression syntax is largely identical to Perl's except for one relevant exception: named groups. Named groups are covered briefly in this tutorial because they are important for effectively parsing command output. For a more thorough reference, visit python.org.

***Komodo Tip***: Use the Komodo Rx Toolkit to build, edit, or test regular expressions. New to regular expressions? The Regular Expressions Primer is a tutorial for those wanting to learn more about regex syntax.

To parse the following line of output generated by running grep:
`hello.pl:5:print "Hello, frogs!\n";`
Output lines are of the form:
`<file>:<line>:<content>`
An appropriate regular expression to match this line could be:
`(.+?):(\d+):(.*)`

However, when parsing this line, Komodo requires you specify which group (i.e., which parenthesized section) is a filename, which is a line number, and which is content. This is where Python's named groups come in. Using Python's regular expression syntax, for example, `(?P<file>.+?)` instead of just `(.+?)` to assign a name to whatever matches `.+?`.

When parsing output from the Run Command, Komodo searches for the names *file*, *line*, *column* and *content* to determine which part of a matched output line to put into which field in its table of results. Note that because a *column* is not included in this particular line of output, Komodo does not assign a column name.

The regular expression can be extended to:
`(?P<file>.+?):(?P<line>\d+):(?P< content>.*)`
When parsing the line above, Komodo determines that `hello.pl` is the *file*, 5 is the *line* and `print "Hello, frogs!\n";` is the *content*. Click the link below to see the result highlighted in the parsed output.

## Using "Find in Files"

Create a "Find in Files" command using all information presented in this tutorial.

1. On the *Projects* tab, double−click the file *hello.pl*.
2. Position the cursor over the word *frogs*.
3. Select *Tools/Run Command*.
4. On Windows, enter the command:
   ```
   findstr /s /n /c:"%(w:orask:Search Term)" "%(ask:File
   Pattern:*.*)"
   ```
   Or on Linux enter the command:
   ```
   find . -name "%(ask:File Pattern:*)" | xargs -l grep -nH
   "%(w:orask:Search Term)"
   ```

   Note that *findstr* is a Windows command line utility that searches for strings in files.
5. Select the *Add to Toolbox* check box to save this command.
6. In the *Start in* field, enter:
   ```
   %(ask:Start Directory:%D)
   ```
   (When the command is run, Komodo should prompt for the "Start Directory" using the directory of the current file, or %D as the default value).
7. Select the *Parse output with* check box and enter:
   ```
   ^(?P<file>.+?):(?P<line>\d+):(?P<content>.*)$
   ```
   as the regular expression with which to parse.
8. Select the *Show parsed output as a list* check box.
9. Click *Run*. The Interpolation Query dialog box is displayed.



10. Click *OK* to run *findstr*. A list of all occurrences of "frogs" in the files of the Run Command

tutorial project is displayed on the Command Output tab.
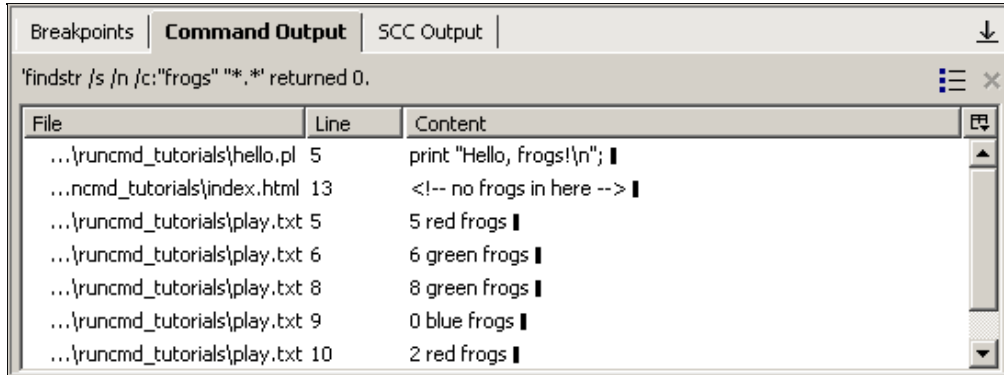


11. Double−click on a parsed result to jump to a specific file and line.
12. Click the ☰ button to toggle back to the Command Output tab.
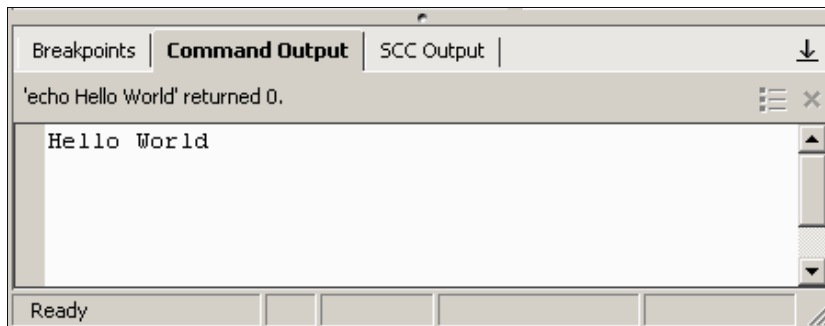


Alternatively, double−click on lines in the *raw* output view to jump to that file and line.

# Installing Komodo 3.0

# Windows

## Prerequisites

### Hardware Requirements

- Intel x86 processor, 200 MHz (or faster) with 128 MB RAM.
- Up to 230 MB in your TMP directory (as indicated by the value of your 'TMP' environment variable) during installation, even if you plan to install Komodo to another drive. If you do not have the required space on this drive, manually set the 'TMP' environment variable to a directory on a drive with sufficient space.

### Operating System Requirements

*Supported operating systems*:

The following platforms are officially supported. Current Critical Updates, Windows Updates, and Service Packs must be installed (see http://windowsupdate.microsoft.com).

- Windows XP
- Windows Server 2003

- Windows 2000

*Other operating systems*:

Komodo can also be run on the following platforms. This version of Komodo has not necessarily been tested on these platforms; platform–specific bugs may or may not be fixed.

- Windows NT4 (service pack 5)
- Windows Me
- Windows 98 (remote debugging unavailable)

## Software Prerequisites on Windows

Installation Prerequisites:

- ***Windows 98, Me and NT users***: Microsoft Windows Installer (MSI) version 2.0 or greater ( MSI 2.0 for 9x and Me, MSI 2.0 for NT)
- ***Windows 98/NT4 Users: Windows Scripting Host:*** Microsoft's Windows Scripting Host is required by the Microsoft Windows Installer. Older versions of Windows did not include the Windows Scripting Host. To check if your system has the Windows Scripting Host, select ***Run*** from the Windows Start menu, and enter `wscript`. If the Windows Script Host Setting dialog appears, WSH is installed on your system. If it doesn't, download the WSH from http://www.microsoft.com/msdownload/vbscript/scripting51.asp.

Language and Debugging Prerequisites:

- ***Debugging***: if firewall software is installed on the system, it must be configured to allow Komodo to access the network during remote debugging.
- ***Perl***: ***Perl 5.6*** or greater is required to debug Perl programs. You can download the latest version of ActivePerl from the ActiveState website. Ensure that the directory location of the Perl interpreter (by default, `C:\perl`) is included in your system's `PATH` environment variable. Some advanced features, such as background syntax checking and remote debugging, require ActivePerl.
- ***Python***: ***Python 1.5.2*** or greater is required to debug Python programs. You can download the latest version of ActivePython from the ActiveState website. Ensure that the directory location of the Python interpreter (by default `C:\Pythonxx` (where "xx" is the Python version)) is included in your system's `PATH` environment variable. Some advanced features, such as background syntax checking and remote debugging, require ActivePython. ***Python 1.5.2*** or greater and a fully configured Tkinter installation are required to create Python dialogs with the GUI Builder.
- ***PHP***: ***PHP 4.0.5*** or greater is required for PHP syntax checking. ***PHP 4.3.1*** or greater is required to debug PHP programs. Debugging and syntax checking are also available for ***PHP 5***. Download PHP from http://www.php.net/downloads.php. Ensure that the directory location of the PHP interpreter (by default `C:\PHP`) is included in your system's `PATH` environment variable. For complete instructions for configuring Komodo and PHP, see Configuring the PHP Debugger. PHP debugging extensions are available on ASPN, the ActiveState Programmer Network.

- *Tcl*: *Tcl 7.6* or greater is required to debug Tcl programs. You can download the latest version of ActiveTcl from the ActiveState website.

GUI Builder Prerequisites:

- *Perl Dialogs created with the GUI Builder*: Require the Perl Tk module. This module has been included with ActivePerl since build 631. If necessary, manually install using the command `"ppm install Tk"`.
- *Python Dialogs created with the GUI Builder*: Require Python version 1.5.2 or greater.
- *Tcl Dialogs created with the GUI Builder*: Require Tcl version 8.3 or greater.

Source Code Control Integration Prerequisites:

- *CVS Source Control Integration*: Requires CVS, which is available from http://www.cvshome.org, or the latest stable version of CVSNT, which is available from http://www.cvsnt.org/wiki/.
- *CVS Source Control Integration using Putty*: Requires Putty version 0.52 or greater.
- *Perforce Source Control Integration*: Requires a connection to a Perforce server with version 99.1 or later.

Miscellaneous Prerequisites:

- *Komodo Documentation*: Komodo's documentation is displayed within the system's default browser. Supported browsers include Mozilla 1.0 or greater (and all browsers based on Mozilla 1.0 or greater), Opera 6 or greater, Internet Explorer 5 or greater and Konqueror version 3.0.3 or greater. JavaScript and cookies must be enabled.
- *The Visual Package Manager*: requires ActivePerl Build 633 or later.
- *Web Services*: To use Web services in Perl or Python programs, ActivePerl and / or ActivePython are required.
- *Perl Dev Kit*: In order to build executable programs, ActiveX controls and Windows services in Perl, you must have ActiveState's Perl Dev Kit version 3.1 or greater installed on your system.

# Upgrading from Previous Komodo Versions

To determine your current Komodo version, select *Help/About Komodo*.

## Uninstalling

Unless you are currently using a beta version of Komodo, it is not necessary to uninstall previous versions before installing Komodo 3.0 as long as the new version is installed in its own new directory. The Komodo 3.0 installation will import settings from previous Komodo versions.

If you want to replace an existing Komodo installation with the latest version, or if you are currently using a beta version, uninstall the existing version before installing the new one.

If you want to run two versions of Komodo, ensure they are installed in separate directories with unique names.

**Remote Debugging**

This version of Komodo contains updated remote debugging extensions. Refer to the Remote Debugging instructions for information about updating the extension.

- [Debugging Perl Remotely](#)
- [Debugging Python Remotely](#)
- [Configuring the PHP Debugger](#)
- [Using the Tcl Remote Debugger](#)

# Installing Komodo on Windows

Before you start:

- If you intend to run the installation from a shared network drive, your system must have SYSTEM rights (or greater) to the directory from which the installation is run. Alternatively, run the installation from a local drive.

To install Komodo on Windows:

1. Ensure you have the prerequisite hardware and software.
2. Download the Komodo installer file.
3. Double−click the installer file and follow the instructions.

When installation is complete, you will see an ActiveState Komodo icon on your desktop.

# Starting Komodo on Windows

To start Komodo on Windows, use one of the following methods:

- Double−click the desktop icon
- Select ***Start/Programs/ActiveState Komodo/Komodo***
- Add the Komodo install directory to your PATH environment variable, then from the command line prompt, enter komodo.

## Uninstalling Komodo on Windows

To uninstall Komodo, select *Start|Programs|ActiveState Komodo|Modify, Repair or Uninstall Komodo*.

Alternatively, use the *Add/Remove Programs* menu (accessible from the Windows Control Panel).

# Linux

## Prerequisites

### Hardware Requirements

- Intel x86 processor, 233 MHz (or faster) with 128 MB RAM (or greater)
- 100 MB hard disk space
- up to 200 MB of temporary hard disk space during installation

### Operating System Requirements

*Supported operating systems*:

The following platforms are officially supported.

- Red Hat 7.3, 8.x and 9.x
- Red Hat Enterprise Linux 3 (WS, ES and AS)
- SuSE 9.x

*Other operating systems*:

Komodo can also be run on the following platforms. This version of Komodo has not necessarily been tested on these platforms; platform–specific bugs may or may not be fixed.

- Fedora Core 2
- SuSE 8.2
- Debian
- FreeBSD (with Linux binary compatibility)
- Gentoo

### Software Prerequisites on Linux

Installation Prerequisites:

- *glibc 2.1 (or higher) and libjpeg.so.62 (or higher)*: These libraries are included in standard Linux distributions.

Two packages, based on different C++ libraries, are available for both Komodo Professional and Komodo Personal:

- The libcpp3 package: built for libstdc++ version 3 (Red Hat 7.x and 8.x)
- The libcpp5 package: built for libstdc++ version 5 (Red Hat 9.x, Red Hat Enterprise Linux 3, SuSE 8.x and 9.x, Fedora Core 2)

Choose the distribution that corresponds to your version of `libstdc++`. To check which version of `libstdc++` is installed on your machine, run the following command:

```
ls /usr/lib/libstdc++*
```

If `libstdc++.so.5` appears in the output, install the libcpp5 version. If not, install the libcpp3 version.

Language and Debugging Prerequisites:

- *Debugging*:
  - ♦ Python, XSLT and PHP require TCP/IP to be installed and properly configured, even if you are debugging scripts locally.
  - ♦ If firewall software is installed on the system, it must be configured to allow Komodo to access the network during remote debugging.
- *Perl: Perl 5.6* or greater is required to debug Perl programs. You can download the latest version of ActivePerl from the ActiveState website. Ensure that the directory location of the Perl interpreter is included in your system's PATH environment variable. See Adding Perl or Python to the PATH Environment Variable for instructions. Some advanced editing features, such as background syntax checking, also require that you have Perl installed.
- *Python: Python 1.5.2* or greater is required to debug Python programs. You can download the latest version of ActivePython from the ActiveState website. Ensure that the directory location of the Python interpreter is included in your system's PATH environment variable. See Adding Perl or Python to the PATH Environment Variable for instructions. Some advanced editing features, such as background syntax checking, also require that you have Python installed. *Python 1.5.2* or greater and a fully configured Tkinter installation are required to create Python dialogs with the GUI Builder.
- *PHP: PHP 4.0.5* or greater is required for PHP syntax checking. *PHP 4.3.1* or greater is required to debug PHP programs. Debugging and syntax checking are also available for *PHP 5*. Download PHP from http://www.php.net/downloads.php. For complete instructions for configuring Komodo and PHP, see Configuring the PHP Debugger. PHP debugging extensions are available on ASPN, the ActiveState Programmer Network. If you intend to debug PHP applications on a remote Linux Red Hat 9 server, you must install the Red Hat 7.3 backwards–compatibility package. Download the file `compat-libstdc++-7.3-2.96.118.i386.rpm` from the Red Hat FTP site (ftp://rpmfind.net/linux/redhat/9/en/os/i386/RedHat/RPMS).
- *Tcl*: *Tcl 7.6* or greater is required to debug Tcl programs. You can download the latest version of ActiveTcl from the ActiveState website.

GUI Builder Prerequisites:

- *Perl Dialogs created with the GUI Builder*: Require the Perl Tk module. This module has been included with ActivePerl since build 631. If necessary, manually install using the command `"ppm install Tk"`.
- *Python Dialogs created with the GUI Builder*: Require Python version 1.5.2 or greater.
- *Tcl Dialogs created with the GUI Builder*: Require Tcl version 8.3 or greater.

Source Code Control Integration Prerequisites:

- *CVS Source Control Integration*: Requires CVS, which is available from http://www.cvshome.org.
- *Perforce Source Control Integration*: Requires a connection to a Perforce server with version 99.1 or later.

Miscellaneous Prerequisites:

- *Komodo Documentation*: Komodo's documentation is displayed within the system's default browser. Supported browsers include Mozilla 1.0 or greater (and all browsers based on Mozilla 1.0 or greater), Opera 6 or greater, Internet Explorer 5 or greater and Konqueror version 3.0.3 or greater. JavaScript and cookies must be enabled.
- *The Visual Package Manager*: requires ActivePerl Build 633 or later.
- *Web Services* To use Web services in Perl or Python programs, ActivePerl and / or ActivePython are required.

## Adding Perl or Python to the `PATH` Environment Variable

To add Perl or Python to the PATH environment variable, do one of the following:

- Modify your PATH environment variable. For example, if you use the Bash shell, add the following line to your ~/.bashrc file:

  ```
  export PATH=<installdir>/bin:$PATH
  ```

  ...where *<installdir>* points to the directory where you installed ActivePerl or ActivePython.
- Create a symbolic link to the Perl or Python executable. For example, for ActivePerl, enter:

  ```
  ln -s <installdir>/bin/perl /usr/local/bin/perl
  ```

  For ActivePython, enter:

  ```
  ln -s <installdir>/bin/python /usr/local/bin/python
  ```

  ...where *<installdir>* points to the directory where you installed ActivePerl or ActivePython.

# Upgrading from Previous Komodo Versions

To determine your current Komodo version, select *Help|About Komodo*.

### Uninstalling

Unless you are currently using a beta version of Komodo, it is not necessary to uninstall previous versions before installing Komodo 3.0 as long as the new version is installed in its own new directory. The Komodo 3.0 installation imports settings from previous Komodo versions.

If you want to replace an existing Komodo installation with the latest version, or if you are currently using a beta version, uninstall the existing version before installing the new one.

If you want to run two versions of Komodo, ensure they are installed in separate directories with unique names.

### Remote Debugging

This version of Komodo contains updated remote debugging extensions. Refer to the Remote Debugging instructions for information about updating the extension.

- Debugging Perl Remotely
- Debugging Python Remotely
- Configuring the PHP Debugger
- Using the Tcl Remote Debugger

# Installing Komodo on Linux

This version of Komodo allows non−root installation on Linux. Note, however, that the user who executes the license file will be the user who is licensed to use the software.

To install Komodo on Linux:

1. Ensure you have the prerequisite hardware and software.
2. Create a temporary directory into which you will download the Komodo installer file. You will delete this directory after the installation procedure is complete.
3. Download the Komodo installer file (with the extension `tar.gz`) into the temporary directory.
4. Crack the tarball. When you enter the command below, it will unpack the file into a directory with the same name as the Komodo tar.gz file. From the command line, enter:

   ```
   tar -xvzf Komodo-<version>-<build>-linux-ix86.tar.gz
   ```

   ...where *<version>* is the version of Komodo and *<build>* is the Komodo build number.

5. Change to the new directory. From the command line, enter:

```
cd Komodo-<version>-<build>-linux-ix86
```
6. Run the install script ("install.sh"). From the command line, enter:

```
./install.sh
```
7. Answer the installer prompts:

♦ Specify where you want Komodo installed or press Enter to accept the default location (`/home/`*user*`/Komodo-x.x`).

If multiple users are sharing the system and will be using the same installation, install Komodo in a location every user can access (e.g. `/opt/Komodo-x.x/` or `/usr/local/Komodo-x.x/`).

*Note*: Each Komodo user requires an individual license key.

Do not install Komodo in a path that contains spaces or non−alphanumeric characters.

Be sure to install Komodo into a directory with a unique name. Do not install Komodo directly in a generic directory containing numerous shared files and directories (such as `/usr/local`) because Komodo does not divide its installed files into `bin`, `lib`, `include`, `share`, and `etc` subdirectories.

♦ Verify you have enough disk space.

*Note* – The installer will not install Komodo into a directory where Komodo is already installed. Be sure to install Komodo into a directory with a unique name.

8. Add the Komodo directory to your path. To do this, either:

♦ Modify your `PATH` environment variable. For example, if you use the Bash shell, put the following in your `~/.bashrc` file:

```
export PATH=<installdir>:$PATH
```

...where *<installdir>* points to the location where Komodo was installed.

♦ Create a symbolic link to the Komodo executable (not to the bin directory). For example:

```
ln -s <installdir>/komodo /usr/local/bin/komodo
```

...where *<installdir>* points to the directory where Komodo was installed.

When installation is complete, you can delete the temporary directory where the Komodo tarball was cracked.

The Komodo license will be sent to the email address you provided during registration. You must install this license. Otherwise, you will get an error if you select the "License" link from within Komodo.

1. To make the license executable, enter:

```
chmod +x <filename>
```

2. To run the license, enter:

```
./<filename>
```

## Starting Komodo on Linux

To start Komodo on Linux:

- If necessary, open the graphical shell by entering `startx`.
- Open the GUI terminal emulator, and enter `komodo`.

## Uninstalling Komodo on Linux

Use the procedure only if you want to fully remove Komodo from your system.

Note that you cannot relocate an existing Komodo installation to a new directory. You must uninstall Komodo from the existing location and reinstall it in the new location.

To uninstall Komodo on Linux:

1. Delete the directory that Komodo created during installation.
2. If you wish to delete your Komodo preferences, delete the `~/.komodo` directory. If you do not delete this directory, subsequent installations of Komodo will use the same preferences.

# Solaris

## Prerequisites

### Hardware Prerequisites

- Sun Sparc architecture
- UltraSparc IIe or faster CPU recommended
- 256MB RAM min, 512MB recommended
- 100 MB hard disk space
- up to 200 MB of temporary hard disk space during installation

### Operating System Requirements

- Solaris 2.8 or later.

*Note* – Komodo Personal Edition is not available for Solaris.

### Software Prerequisites

Installation Prerequisites:

- *GNU tar*: is required for unpacking the Komodo installation file. If you use the native Solaris tar version, you will get a checksum error when attempting to unpack the Komodo tar package, and any attempt to run the install script will fail.

Language and Debugging Prerequisites:

- *Debugging*:
    - ♦ Python, XSLT and PHP require TCP/IP to be installed and properly configured, even if you are debugging scripts locally.
    - ♦ If firewall software is installed on the system, it must be configured to allow Komodo to access the network during remote debugging.
- *Perl: Perl 5.6* or greater is required to debug Perl programs. You can download the latest version of ActivePerl from the ActiveState website. Ensure that the directory location of the Perl interpreter is included in your system's PATH environment variable. See Adding Perl or Python to the PATH Environment Variable for instructions. Some advanced editing features, such as background syntax checking, also require that you have Perl installed.
- *Python: Python 1.5.2* or greater is required to debug Python programs. You can download the latest version of ActivePython from the ActiveState website. Ensure that the directory location of the Python interpreter is included in your system's PATH environment variable. See Adding Perl or Python to the PATH Environment Variable for instructions. Some advanced editing features, such as background syntax checking, also require that you have Python installed.
- *PHP: PHP 4.0.5* or greater is required for PHP syntax checking. *PHP 4.3.1* or greater is required to debug PHP programs. Debugging and syntax checking are also available for *PHP 5*. Download PHP from http://www.php.net/downloads.php. For complete instructions for

configuring Komodo and PHP, see Configuring the PHP Debugger. PHP debugging extensions are available on ASPN, the ActiveState Programmer Network.

- *Tcl*: *Tcl 7.6* or greater is required to debug Tcl programs. You can download the latest version of ActiveTcl from the ActiveState website.

GUI Builder Prerequisites:

- *Perl Dialogs created with the GUI Builder*: Require the Perl Tk module. This module has been included with ActivePerl since build 631. If necessary, manually install using the command "ppm install Tk".
- *Python Dialogs created with the GUI Builder*: Require Python version 1.5.2 or greater.
- *Tcl Dialogs created with the GUI Builder*: Require Tcl version 8.3 or greater.

Source Code Control Integration Prerequisites:

- *CVS Source Control Integration*: Requires CVS, which is available from http://www.cvshome.org.
- *Perforce Source Control Integration*: Requires a connection to a Perforce server with version 99.1 or later.

Miscellaneous Prerequisites:

- *Komodo Documentation*: Komodo's documentation is displayed within the system's default browser. Supported browsers include Mozilla 1.0 or greater (and all browsers based on Mozilla 1.0 or greater), Opera 6 or greater, Internet Explorer 5 or greater and Konqueror version 3.0.3 or greater. JavaScript and cookies must be enabled.
- *The Visual Package Manager*: requires ActivePerl Build 633 or later.
- *Web Services* To use Web services in Perl or Python programs, ActivePerl and / or ActivePython are required.

# Installing Komodo on Solaris

This version of Komodo allows non–root installation on Solaris. Use this procedure for installing Komodo.

To install Komodo on Solaris:

1. Ensure you have the prerequisite hardware and software.
2. Download the Komodo installer file (with the extension `tar.gz` into a temporary directory. You need a temporary directory to crack the tarball. You will not install Komodo in this temporary directory.
3. Using GNU tar, crack the tarball into the temporary directory. This unpacks the file into a directory with the same name as the Komodo tar.gz file. From the command line, enter:

```
tar -xvzf Komodo-<version>-<build>.tar.gz
```

...where *<version>* is the version of Komodo and *<build>* is the Komodo build number.

4. Change to the new directory. From the command line, enter:

```
cd Komodo-<version>-<build>
```

5. Run the install script ("install.sh"). From the command line, enter:

```
./install.sh
```

6. Answer the installer prompts:

   ♦ Specify where you want Komodo installed or press Enter to accept the default location
   (`/home/`*user*`/Komodo-x.x`).

   If multiple users are sharing the system and will be using the same installation, install
   Komodo in a location every user can access (e.g. `/opt/Komodo-x.x/` or
   `/usr/local/Komodo-x.x/`).

   *Note*: Each Komodo user requires an individual license key.

   Do not install Komodo in a path that contains spaces or non−alphanumeric characters.

   Be sure to install Komodo into a directory with a unique name. Do not install Komodo
   directly in a generic directory containing numerous shared files and directories (such as
   `/usr/local`) because Komodo does not divide its installed files into `bin`, `lib`,
   `include`, `share`, and `etc` subdirectories.

   Do not install Komodo in a path that contains spaces or non−alphanumeric characters.

7. Verify you have enough disk space.

   *Note* – The installer will not install Komodo into a directory where Komodo is already installed.
   Be sure to install Komodo into a directory with a unique name.

8. Add the Komodo directory to your path. To do this, either:

   ♦ Modify your `PATH` environment variable. For example, if you use the Bash shell, put the
   following in your `~/.bashrc` file:

   ```
   export PATH=<installdir>:$PATH
   ```

   ...where *<installdir>* points to the location where Komodo was installed.
   ♦ Create a symbolic link to the Komodo executable (not to the bin directory). For example:

   ```
   ln -s <installdir>/komodo /usr/local/bin/komodo
   ```

   ...where *<installdir>* points to the directory where Komodo was installed.

When installation is complete, you can delete the temporary directory where the Komodo tarball was
cracked.

The Komodo license will be sent to the email address you provided during registration. You must install

this license. Otherwise, you will get an error if you select the "License" link from within Komodo.

## Starting Komodo on Solaris

To start Komodo on Solaris:

- If necessary, open the graphical shell by entering `startx`.
- Open the GUI terminal emulator, and enter `komodo`.

## Uninstalling Komodo on Solaris

Use the procedure only if you want to fully remove Komodo from your system.

Note that you cannot relocate an existing Komodo installation to a new directory. You must uninstall Komodo from the existing location and reinstall it in the new location.

To uninstall Komodo on Solaris:

1. Delete the directory that Komodo created during installation.
2. If you wish to delete your Komodo preferences, delete the `~/.komodo` directory. If you do not delete this directory, subsequent installations of Komodo will use the same preferences.

# Release Notes

Welcome to Komodo, ActiveState's Integrated Development Environment (IDE). This document accompanies Komodo version 3.0.

- New in Komodo 3.0
- Release History
- Known Issues

## Komodo 3.0: July 2004

This release includes the following new features and bug fixes:

### Code Intelligence

Komodo's Code Intelligence system is a set of tools that supports multiple languages. The Code Intelligence tools include the Code Browser, Object Browser and Python AutoComplete and CallTips. All Code Intelligence tools require a Code Intelligence database to operate fully.

The Object Browser is a graphical browser that searches the Code Intelligence database for specified code symbols and modules. Use the Object Browser's preview pane to view code snippets containing the search criteria.

The Code Browser displays on the *Code* tab next to the *Projects* tab in the Left Pane. The Code Browser displays a hierarchical tree view of all code constructs (for example, variables, methods, imports) in all open files. For Python, instance attributes are all displayed. The code tree, which can be navigated using either the keyboard or mouse, includes the following features:

- Double−click a node to jump to that point in the code.
- When the cursor is positioned in the Editor Pane, clicking the Locate current scope button on the Code Toolbar or pressing 'Ctrl'+'K', 'Ctrl'+'L' causes Komodo to jump to the node in the code tree that most closely matches the current point in the program. Additionally, a Scope Indicator, located in the status bar, displays the current scope of a selected code construct. Double−click the Scope Indicator to open the tree hierarchy to the code construct.
- When the focus is in the Code Browser tree, pressing the 'Tab' key causes Komodo to shift to the Filter text box. Typing any string in this text box causes the Code Browser to display all of the nodes matching that string. Pressing 'Tab' again shifts the focus back to the tree. Clearing the *Filter* text box returns the tree to its original state. Note that all matching nodes are shown, even those that are invisible because their parents were not "expanded".
- The Sort menu on the Code Toolbar lets you toggle whether the nodes at each level of the tree are sorted in alphabetical order or according to the order they occur in the file.
- The Code Description pane, located in the lower part of the Code Browser, displays additional documentation (when available) on various program components.

Python AutoComplete and CallTips are enhanced with the Code Intelligence database.

# Interactive Shell

The interactive shell is an implementation of Perl, Python and Tcl's interactive shell within Komodo. The interactive shell supports, AutoComplete, CallTips, debugging functions, customization and history recall.

# Debugging

- A debugger proxy is available that enables multiple users to debug with Komodo on the same machine.
- The *Breakpoints* tab in the Bottom Pane of the Komodo workspace is designed for convenient management of breakpoints and Tcl spawnpoints. This tab displays enabled and disabled breakpoints and spawnpoints. Double−clicking an enabled or disabled breakpoint on the *Breakpoints* tab opens the related file in the Editor Pane and shifts focus to the line in the program associated with that breakpoint.
- Komodo now handles multi−session debugging. Komodo can debug multiple programs, regardless of the supported languages used in the applications being concurrently debugged.
- The Bottom Pane of the Komodo workspace (previously the Output Pane) has been reorganized and enhanced. The new Debug tab is displayed whenever a Komodo debugging session is launched.
- A Debug Toolbar is available on the *Debug* tab in the Bottom Pane of the Komodo workspace. It includes additional controls for detaching from a new debugging session and forcing a break.
- New remote debugging features include the ability to listen for remote debugger sessions continuously, set remote debugger preferences, and check the listener status of the current configuration.
- The new debugger properties dialog box supports multiple 'configurations' to be saved per file.
- The new Debugger dialog box (displayed when a debug session is invoked) supports configurable interpreter arguments.
- The *Disable Output Buffering* and *Enable Implicit Flush* options for PHP debugging have no effect when Simulate CGI Environment is selected. To disable output buffering in CGI emulation mode, manually comment out the output_buffering setting in *php.ini* with a ";" character, or set it to "off".

# Rx Toolkit

The Rx Toolkit has been completely overhauled. New features include:

- Match Type buttons that set the match mode for the regular expression. The buttons represent functions that let you *Match*, *Match All*, *Split*, *Replace* and *Replace All*.
- For Python regular expressions, there are two new modifiers: Unicode and Locale.

# Multi−User Features

- A Shared Toolbox with functionality similar to the standard Toolbox makes it possible for multiple users to share items such as run commands and code snippets. The Shared Toolbox is enabled via a check box on the Shared Support page of Komodo Preferences.

- Users can accept a default Common Data Directory or specify a custom location via the new Shared Support page in Komodo Preferences.
- Multiple users can share .tip, .pcx and .pdx files and preferences.

## Enhanced Search Functionality

- The new Open/Find Toolbar makes it easier to open files and search for strings in currently open files.
- A Find in Files dialog box is used to search for files that are not currently open.

## Macro Enhancements

- A Macro Toolbar provides a quick and easy way to record, pause, play, stop and save macros.
- Macros are no longer stored as separate files (as they were in 2.x). Because macro content is now included in a macro item, it is easier to share macros between multiple users.
- The Properties dialog box for macros contains a window in which macros can be coded in either JavaScript or Python.
- Triggers can be set so that macros execute as a result of specific Komodo events.
- The documentation includes a macro API.

## Custom Toolbars, Menus and Icons

Create Custom Toolbars and Menus for frequently used components. Custom Toolbars are displayed beneath the existing toolbars; custom menus are added to Komodo's top–level menu, between the *Tools* and *Windows* menus. Custom icons can be assigned to components like run commands, snippets, etc.

## Editing Enhancements

- Reflow Paragraph
- Join Lines
- Enter Next Character as Raw Literal
- Repeat Next Keystroke N Times
- Emacs–style "marks"
- Clean Line Endings on save

## Miscellaneous

- PHP 5 is supported.
- Support for Web services has been deprecated. Note that this will not prevent the use of Web services in applications developed in Komodo.
- Users familiar with the Emacs editor can configure Komodo to use Emacs key bindings and set a File Associations preference so that Komodo checks for an embedded Emacs mode specification.
- Support for Tcl has been improved to include more detailed logging and additional syntax checking options.
- Two new interpolation shortcuts (%L and %P) can now be used in the Run Command dialog

box.
- Users now have the option to assign custom icons to Komodo components (snippets, run commands, etc).
- Printing features for specifying line wrapping and font scaling have been added to the Printing page in Komodo Preferences. This page is now also accessible by clicking **Print Settings** on the **File** menu.
- Components from Projects or the Toolbox can be exported to a self–contained archive for distribution to other users. The import and export wizards are invoked via the right–click context menu in the Project or Toolbox tab, or from the Toolbox menu.
- Perl, Python and Tcl preferences include the ability to specify import directories.
- On the **Editor/Key Bindings** preference page, the keybindings list can be filtered.

## Documentation

- The new Feature Showcases are quick demos of Komodo features.
- Komodo Help now displays in a selected web brower.
- Improved search capabilities, including phrase search and result ranking, make it easier to locate specific information.
- The documentation includes an index.
- A PDF file of the complete documentation set makes it possible to quickly create a print version of Komodo's documentation. Click the PDF link to the right of the **Search** tab to view and/or print the PDF file.
- Context–sensitive Help buttons have been added throughout the Komodo workspace.
- The documentation includes a macro API.

To view the complete list of bugs fixed in this release, see http://bugs.activestate.com.

# Release History

## Komodo 3.0 Beta 4: June 2004

## Komodo 3.0 Beta 3: May 2004

## Komodo 3.0 Beta 2: May 2004

## Komodo 3.0 Beta 1: May 2004

## Komodo 2.5.2: January 2004

This release contains bug fixes for Komodo 2.5 and 2.5.1, including:

- Intermittent crashes were occurring on Linux, caused by the autosave feature.
- On Linux, the Internationalization dialog in Komodo's Preferences was generating an error.

- Perl's Visual Package Manager was not always reporting a startup failure to the user.
- Autocomplete for Perl and Python did not work.
- Intermittently, the variable type indicator (i.e., $, @, %) was not being selected when double−clicking on a variable in the editor pane.
- Umlaut characters were causing file contents to be lost on save.
- CVS source code control integration functions were not available on directories or projects.
- The Tcl statement `source file` was not finding the specified file.
- There were errors in Tcl syntax checking.
- CVS source code control integration was generating errors on Solaris.
- There were debugging errors when working with Tcl scripts on Solaris.
- When adding template items to a project or the Komodo Toolbox, it was not possible to cancel the file selection operation.
- The CVS source code control integration now supports use of a local repository.
- The update function in Komodo's source code control integration was not functioning under certain circumstances.
- On Solaris, it was not possible to create a file on an NFS−mounted disk.
- Regardless of the setting for line endings, new files were created with DOS/Windows line terminators.
- Perl brace matching could cause Komodo to lock up under certain circumstances.
- XML autocomplete could cause Komodo to lock up.
- During a workspace restoration, if multiple files contained breakpoints, the breakpoints were not being written to Komodo's breakpoint manager and would have to be manually re−set.
- After debugging, debugger arrows were still active on files that had been stepped into.
- Files names that included spaces could not be added to projects.
- Under certain circumstances, Komodo would not start on Red Hat version 8.
- Autosave functionality is now more robust.
- Under certain circumstances, opening files resulted in a blank page in the editor pane.
- On Linux, the debug session would occasionally hang due to problems with the remote debug listener.

## Komodo 2.5.1: October 2003

This release contained bug fixes for Komodo 2.5. The only user−visible changes were:

- When clickable links are displayed in the Output Pane, you must double−click (rather than single click) to follow the link. This allows for portions of the link to be selected via single−clicks. The link is no longer underlined.
- The default Perl remote debugging port has changed from 9010 to 9011 (due to a conflict with the XIM server on UNIX). To use the new port, alter the PERLDB_OPTS setting on the remote machine. If you are invoking the Komodo debugger via the Perl Dev Kit's −debug switch, ensure that the specified port number matches the Komodo's port number.
- The menu link for the Start Page has been moved from the Help menu to the Windows menu.
- The *Close All* menu option no longer closes the Start Page.
- The *Internationalization* preference page for language encodings has changed. Programming−language specific encodings are now used for existing files, as well as new files.

The following major bugs were fixed:

- The highlighting of selected text was not released if the mouse pointer moved outside of the editor pane.
- Komodo Personal failed to start up properly if the *Show Button Text* option was enabled.
- Background syntax checking would cease to function.
- Saving of remote files failed for long editing sessions.
- Remote debugging didn't honor Perl breakpoints.
- The Rx window could hang.
- Hotspots in the Output pane were generated for spurious lines and the dialog was hard to dismiss.
- Installing web services for Perl 5.8 is fixed.
- The *Build Standalone Perl Application* dialog didn't deal well with scripts with UNC filenames (\\machine\drive\...).
- Printing collapsed whitespace in the middle of lines.
- The ability to show diffs in an editor tab wasn't working.

## Komodo 2.5 for Windows, Linux: September 2003

*General*

- Komodo's speed has been boosted through re−engineering, resulting in quicker times for startup, tab−switching, loading of dialogs, as well as overall performance gains.
- Menus have been re−organized, and new items have been added. Every Komodo feature can now be accessed through the top−level menus.
- The Preferences pages and a number of existing dialogs have been improved and reorganized for ease−of−use. Changed dialogs include those associated with current file settings, templates, source code control, remote files, and the Perl Dev Kit.
- Templates can now use the same interpolation shortcuts as commands and snippets. Prior to Komodo Version 2.5, a set of variables could be used to embed the current date and time in files created from custom templates. The old codes have been deprecated and will be removed in the next Komodo release.
- The Show Unsaved Changes command, previously exclusive to the Editor context menu, has been added to the *File* menu.
- A new *Compare Files* feature on the *Tools* menu provides a quick and easy way to locate and "diff" two files.
- A *Word Wrap* command has been added to the *View* menu.
- A Tcl tutorial has been added to Komodo's set of language tutorials. Another new tutorial, the Komodo Tour, provides an introduction to the program's key features.
- ASPN Cookbook integration is no longer supported.

*Workspace*

- Changes have been made to make more efficient use of the Komodo workspace:
    - A *Full Screen* command has been added to the *View* menu.
    - The Toolbox and Project Manager, which previously appeared in separate panes, are

now contained in single pane. You can select ***Projects/Toolbox Pane*** from the ***View*** menu to show or hide this pane.

   ♦ Toolbar button text is not displayed by default. Hiding the button text makes it possible for the toolbars to share a single row.

- The functionality of Komodo's ancillary windows has been enhanced. The Debug Output, Command Output, and SCC Output tabs in the Output Pane offer context menus with access to a subset of the top–level View menu. There are key bindings for each of the context menu items. The same context menu is available in the "diff" window, which is invoked by either the Show Unsaved Changes command or the ***Compare Files*** command. The ancillary windows also support "hot–spotting". If Komodo detects a file reference, it will create a hot spot that links to the file itself.
- Interpreter errors displayed in the Output Pane are now hyperlinks to the relevant line.
- ***Ctrl+Tab*** functionality is now supported in all of the panes in the Komodo workspace. Earlier versions of Komodo supported tab–switching in the Editor Pane only.
- The ***Output*** tab and the ***Variables*** tab are now known as the Debug Output tab and Debug Variables tab, respectively.
- The "grippies" used to show and hide the panes in the Komodo workspace have been abandoned in favor of "x" close buttons.

### Project Manager and Toolbox

- The new ***Toolbox menu*** provides functionality for importing and exporting items, along with other commands that were previously exclusive to the Toolbox context menu. The Projects/Toolbox Pane contains the Projects tab and the Toolbox tab. You can show or hide the ***Projects*** and ***Toolbox*** tabs by pressing ***Ctrl+Shift+P*** and ***Ctrl+Shift+T***, respectively.

### Editor

- Additions to the ***Window*** menu offer new display options for the Editor Pane. Selecting ***Split View*** divides the Editor Pane in two, so that you can preview files (e.g., HTML or XML) or compare two files. Select ***Rotate Tab Groups*** to toggle vertical and horizontal variations of the split view.
- The View As Language item on the View menu now has a submenu item called ***Reset to best guess***. Select this item to ignore the user preference, and analyze the file in an attempt to determine its language.
- Change the language of an open file by right–clicking the name of the language on the status bar and selecting a language from the pop–up menu.
- A new ***Smart Editing*** preference gives you the choice of configuring the ***Tab*** key instead of ***Ctrl+Space*** to trigger Komodo's word–completion feature.
- The ***Go To Line*** dialog box now supports entries that are relative to the current line. For example, instead of entering a specific line number, you can enter "+10" or "−10" to move ten lines forward or back.
- There are some changes to the default key bindings. For a complete listing of default key bindings, select ***Help/List Key Bindings***.
- The ***Ignore Hidden Text*** option has been removed from the ***Find*** dialog box.
- Komodo now supports editing with non–English characters, based on the locale specified in the regional settings for Windows. See Customizing International Encodings for more information.

*Web and Browser*

- A Preview in Browser command is now available through the *View* menu and the main toolbar. Depending on how the Web and Browser preferences are configured, this command will display a preview in one of three ways:
    - ♦ in the Editor Pane
    - ♦ side−by−side with the source code in a split view of the Editor Pane
    - ♦ in a separate window

*GUI Builder*

- GUI Builder functionality has been extended to include language−specific widget palettes that are displayed according to the initial language selection. In addition, a greater variety of widgets is available.
- The new Dialog tab displays an interactive organizational map that makes it easier to plan and manage the structure of dialogs.
- The new Menu tab is used to create drop−down menus. The *Menu tab* features an additional widget set used to create standard menu commands, as well as commands with check box or radiobutton behavior.

*Printing*

- The new Page Setup options used to set orientation, margins, and headers and footers can now be accessed from the *File* menu.
- A new Print Preview command has also been added to the *File* menu. In addition to navigation buttons for viewing multi−page print jobs, the preview window offers access to *Page Setup* options. As a result, printing preferences available in Komodo 2.3, including *Print Line Wrap Markers*, *Characters per line* and *Font Size Adjustment*, have been removed.
- Selecting the *Print to HTML* command now opens the generated file.

*Fonts and Colors*

- The Fonts and Colors preferences have been streamlined to make managing these settings easier. You can create new font and color schemes based on the default scheme. In addition to Language−Specific Coloring, you can now set Common Syntax Coloring to specify the settings for elements used in multiple languages.

*Debugging*

- A *Run Without Debugging* item has been added to the *Debug* menu, providing a faster way to run scripts in Komodo.
- A new debugging preference, *Try to find files on the local system when remote debugging*, has been added. When this preference is turned on, Komodo will search for the debugger file on the local system instead of automatically opening a read−only version of the file retrieved from the debug engine.
- Komodo now offers full support for Tcl debugging. In addition, runtime Tcl syntax checking no longer requires the Tcl Dev Kit.

*Source Code Control*

- The *Automatically check out before save* option has been moved from "General Source Code Control Preferences" to "Perforce" Preferences, where it appears as *Automatically open files for edit before save*.

# Komodo 2.5 Technology Preview 1 for Solaris: August 2003

# Komodo 2.5 Beta 1 for Windows, Linux: August 2003

# Komodo 2.5 Alpha 2: July, 2003

# Komodo 2.3: February, 2003

*General*

- Compatibility issues with ActivePerl 5.8 (VPM), PHP 4.3 (syntax checking) and Python 2.2.2+ (debugging) have been resolved.
- The XSLT engine has been upgraded to Xerces 2.1 and Xalan 1.4, and includes built−in extensions for that version. (Refer to http://xml.apache.org/xalan−c/extensionslib.html for information about extension support.)
- The new Windows Integration page in the Komodo Preferences is used to configure system−wide file associations.
- The Projects and Workspace page in Komodo can be used to have Komodo prompt you to open recent files and projects when you start the program.
- The configuration of the Komodo workspace is restored after closing and re−opening Komodo.

*PHP Configuration Wizard*

- The PHP Configuration Wizard simplifies the previously cumbersome process of configuring PHP debugging extensions in Komodo.

*Run Command*

- The Run Command function has been enhanced, and a new Run Command Tutorial has been added.

*Toolbox*

- Snippets now support shortcut codes. To view the supported shortcuts, right−click a snippet, select *Properties*, and click the arrow button in the Properties dialog.
- The new *Add "Open..." Shortcut...* item on the Toolbox menu is used to create shortcut links in the Toolbox to directories. Key bindings can be assigned to these links.
- Items can be exported from the Toolbox. Select the desired items, then right−click and select *Export as Project File...*.

- Exported Toolbox items can be imported using the ***Import into Toolbox...*** menu item, accessible from the Toolbox menu.
- The new ***Compare File With...*** item on the file context menus in the Toolbox and Project Manager is used to compare the selected file with another file selected via a file picker dialog.

### *Project Manager*

- Projects load much faster than in previous versions.
- Snippets now support shortcut codes. To view the supported shortcuts, right–click a snippet, select ***Properties***, and click the arrow button in the Properties dialog.
- The new ***Add "Open..." Shortcut...*** item on the Project Manager menu is used to create shortcut links in the Project Manager to directories and URLs. Key bindings can be assigned to these links.
- The new ***Compare File With...*** item on the Project Manager menu is used to compare the selected file with another file selected via a file picker dialog.
- When a project has been changed, it can be reverted to its last saved state using the ***Revert Project*** menu item.
- Many settings now persist in Komodo's projects, including debugging options and Import from File System settings.
- The ***Import from Filesystem*** function prompts if you want to install all the files in a filesystem, and allows you to deselect desired files.
- After using the ***Import from Filesystem*** function, if you attempt to import the same filesystem location into the same project, only files that are new since the last import will be imported.

### *SCC Integration*

- If you are using Komodo's Perforce integration, you can now specify an external "diff" tool.
- The "diff" window displays the original content and the new content in different colors.
- The "diff" window supports function keys to navigate to changes in the file.

### *Editor*

- Select ***Show Unsaved Changes*** from the Editor context menu to view the differences between the version of the file currently displayed in the Editor Pane and the last saved version of the file.
- Breakpoints and code folding information is stored with the file, and therefore does not need to be reconfigured when the file is closed and re–opened.
- There is a new automatic indentation style ("Indent to first non–empty column") that can be configured using Komodo's Indentation preferences.
- The symbols used to indicate folded sections of code (and sections of code that can be folded) can be configured on the ***Editor|Indentation*** page of the Komodo Preferences.
- You can change Read Only status of a file in the file's preferences. Access the file preferences by right–clicking the file in the Editor Pane and selecting ***Properties and Settings***.
- If you change and save a file with a Read Only status, you will be given the option to "force" the write. This will remove the Read Only flag, save the changes, then reset the Read Only flag.
- You can specify the language association for an open file in the file's preferences. Access the preferences for the current file by right–clicking the file in the Editor Pane and selecting ***Properties and Settings***. Use the ***Reset*** button to reassert the language associated with the file

type.

- Various editor bugs have been fixed, including intermittent failure of the paste function, periodic hanging when large projects were being opened, Perl syntax checking problems and lost code folding after tab switching.

### *GUI Builder*

- In Komodo's Preferences, you can specify the preferred port that the GUI Builder will use.
- There are now no constraints for user code added to Python files generated by the GUI Builder.

### *PDK Support*

- Komodo 2.3 supports version 5 of the Perl Dev Kit, including the new PerlNET and PerlTray components.
- Support for all the Perl Dev Kit tools has been enhanced.
- When the ***Build Standalone Perl Application*** function is used, settings in the dialog are stored with the file.

## Komodo 2.3 beta 2: February, 2003

## Komodo 2.3 beta 1: January, 2003

## Komodo 2.0.1 for Linux: November, 2002

- See [Komodo 2.0 for Windows](#) for a list of 2.0 features included in version 2.0.1 for Linux.
- International Encoding support now works with European and Cyrillic character sets. Configure encoding support on the Internationalization page of Komodo Preferences.
- a new Web Preferences page, used to specify the default browser
- additional Run Command Shortcuts

## Komodo 2.0.1 for Windows: October, 2002

## Komodo 2.0 beta 3 for Linux: October, 2002

## Komodo 2.0 beta 2 for Linux: September, 2002

## Komodo 2.0 for Windows: September, 2002

- Project Manager
  - ◆ add Run commands to projects
  - ◆ add GUI dialogs to projects (*Komodo Pro*)
  - ◆ add Web services to projects
  - ◆ add macros to projects
  - ◆ add code snippets to projects
  - ◆ organize objects into folders

- ♦ import groups of files from the filesystem
- ♦ organize the order of objects in projects
- ♦ drag items to and from the Project Manager to the Toolbox and Editor
- Toolbox
  - ♦ add Run commands to the Toolbox
  - ♦ add GUI dialogs to the Toolbox (*Komodo Pro*)
  - ♦ add Web services to the Toolbox
  - ♦ add macros to the Toolbox
  - ♦ add code snippets to the Toolbox
  - ♦ add code snippets to the Toolbox
  - ♦ organize objects into folders
  - ♦ drag items to and from the Toolbox to the Project Manager and Editor
- Customizable Key Bindings for standard Komodo functions, and for macros and Run commands
- Source Code Control integration for CVS and Perforce (*Komodo Pro*)
- Visual Package Manager integration (*Komodo Pro*)
- GUI Builder for building graphical applications (*Komodo Pro*)
- Macros for recording and repeating common keystroke sequences
- Create code snippets by selecting code in the Editor Pane and dropping it into the Project Manager or Toolbox
- Enhanced "Run" Command
  - ♦ save run command to the Toolbox
  - ♦ new shortcuts, including escaped % symbol, URL−escaped selection, etc
  - ♦ specify starting directory for command
  - ♦ specify run location (output window, new console, no console)
  - ♦ specify environment variables
  - ♦ save advanced options (such as environment variables) as Run command defaults
- Function Search to quickly find Perl packages and subs, Python or PHP classes and functions, and Tcl procs
- Incremental Search
- HTML Preview Tab
- Customizable Language Help
- Customizable Cursor, Current Line, Selection and Background Colors
- "Mark All" option in Find dialog
- "Remove All Bookmarks" command
- Printing enhancements
- Background Syntax Checking
  - ♦ hover mouse over icon for total number of errors and warnings
  - ♦ double−click icon to move the editing cursor to the next error or warning
- Tip of the Day
- Ctrl+Tab switches between the two most recently opened or edited files
- Ability to re−order open files by dragging−and−dropping file tabs in the Editor Pane
- File tab context menu with common file tasks, such as saving and accessing source code control commands
- Ctrl+X and Ctrl+C take entire lines if no text is selected, and will continue to accumulate lines
- Ctrl+Shift+V pastes and selects the pasted text
- Improved word wrapping
- Partial word movement

- Overtype / insert toggle support
- Search through the document for the word under the cursor
- Web service proxies in Perl programs
- Brace matching
- Transposing characters
- Literal characters

## Komodo 2.0 beta 2 for Windows: September, 2002

## Komodo 2.0 beta 1 for Linux: September, 2002

## Komodo 2.0 beta 1 for Windows: August, 2002

## Komodo 1.2.9: July, 2002

The following issues were addressed in Komodo 1.2.x releases:

*Editor*

- (Windows only) Syntax checking now works properly for files located on a network.
- (Tcl, Perl and PHP) Double−clicking variables now selects the variable prefix.
- When dragging and dropping text into the Editor Pane, Komodo no longer attempts to open a new file.
- When dragging and dropping text from the Editor Pane to other parts of Komodo, Komodo was crashing. This has been fixed.
- When calltips are displayed, tip highlighting now progresses as statement components are entered in the editor.
- When opening large HTML files in the editor, Komodo no longer displays a syntax error warning.
- Search and replace strings containing backslashes are now handled properly.
- The error regarding invalid regular expressions during search and replace has been fixed.
- Templates in language directories are now correctly included in the language categories.

*General Debugging*

- Watched variables can be manipulated regardless of whether the debugger is running.
- When the debugger is running, use Ctrl+F10 to run the debugger to the current cursor position.
- When the debugger is running, the current line is now highlighted. The highlighting is dimmed when a stack other than the current stack is selected.
- The debugger now positions the current line in the middle of the Editor Pane.

*Perl Editing and Debugging*

- Socket errors in Perl debugger connections have been fixed by the addition of a new configuration item. Select Edit|Preferences|Debugger, and enter the socket port in the Perl Listener Port field. The "CallKomodo" configuration of the PERLDB_OPTS is no longer used,

and can be removed. See Debugging Perl Remotely for more information.

- Support for the overload.pm module has been added.
- Evals in included files no longer fail when stepping into the included file.
- "Here" doc styling is now supported.
- When debugging Perl, the Komodo debugger does not step over "require" statements.

### *PHP Editing and Debugging*

- Komodo now fails gracefully if the php.ini file does not exist in the specified location.
- Autocompletion is now available inside strings.
- The icon on the Komodo start page now correctly shows when PHP has been enabled.
- Autocompletion no longer fails inside certain string elements.
- Syntax checking now functions on large PHP files.
- PHP configuration is correctly reloaded if it is changed.
- Autocompletion now picks up changes in the php.ini file.
- The PHP 4.1 / 4.2 debugger extension is included with the Komodo distribution. Note that the 4.1 debugger extension functions with PHP version 4.2.

### *Tcl Editing and Debugging*

- Breakpoints can now be set in included files.
- For Tcl Dev Kit users, Prowrap was being incorrectly called, rather than Procheck.

### *Other*

- PDK version 4.0 is now supported.
- The Rx Toolkit has better font styling.
- To speed initial startup, sample Web services are no longer added to the Komodo Preferences.

### *Web Services*

- When parsing WSDLs, there is now schema support for complex types (however, not all schema is supported).
- The WSDL parser now parses all XMethods.com WSDL files.

## Komodo 1.2.7 RC1 for Windows and Linux: March, 2002

## Komodo 1.2 for Windows and Linux: December, 2001

### *Web Service Consumption*

Komodo provides Web service support, bookmarks; quickly add Web services to Perl, Python or PHP files (including automatic language support like CallTips and AutoCompletion for Web service objects); browse your Web service bookmark library in the Komodo User Guide.

### *Share Recipes with the ASPN Cookbook*

Komodo provides a fast, easy way to share recipes. Submit your favorite Perl regular expression or Python, Tcl or XSLT code snippet to the ASPN Cookbook.

### Enhanced Editing

The Komodo editor detects files that have changed on disk, and gives you the option to reload. Komodo also remembers the "state" of each file in the Most Recently Used list, including cursor position, bookmarks, fold points, language association, and whitespace preference. Undo changes to a file using the Revert File function. Quickly select blocks of code when performing editing functions; configure text to auto−wrap in the editor and to auto−save at the interval you prefer.

### Keyboard Shortcuts

New keyboard shortcuts include Ctrl+Insert (copy), Shift+Delete (cut), Shift+Insert (paste) and Ctrl+<debug command> (suppress Debug Options dialog). Note that you can also use the Ctrl key in conjunction with the Debugging Toolbar buttons to suppress display of the Debugging Options dialog.

### Find and Replace

Search and / or Replace a word or phrase in all documents open in the Komodo editor. Use the new tab on the Output Pane to view all results from Find or Replace operations.

### Templates and Macros

Code faster with language−specific templates. Create custom templates, and embed variables for date and time stamps.

### Perl Dev Kit Support

Use Komodo in conjunction with the Perl Dev Kit to build Windows executables, services and controls written in Perl.

### Tcl Support

With a membership to ASPN Tcl, use Komodo's syntax checking and debugging with Tcl.

### FTP Support

Edit remote files on FTP servers; add remote files to projects; save files to remote FTP servers. Use Komodo's Preferences to configure connection information for servers you use frequently.

### Komodo User Guide

Search the User Guide or find a word or phrase in the current page; navigate using the Table of Contents; change the font size. Take a visual tour of the Komodo Workspace, the Komodo Debugger, the Komodo Editor or the Rx Toolkit.

*Komodo Tutorials*

Learn about new languages and language–specific Komodo features in the Tutorials.

*PHP*

Now supports AutoCompletion.

*Fonts and Colors*

Customize the display of fonts and colors for elements of every language supported by Komodo.

*CGI Debugging Emulation*

Emulate a CGI environment while debugging on your local machine by configuring server environment variables and form input.

*Enhanced Debugging*

Expanded Debugging Options remember your debug settings from one session to another; view HTML output on the Output tab; configure the Remote Debugging Listener Port; enter program input on the Output tab.

*Run Commands*

Interact with the command line using the Run Commands function.

*Internationalization*

Set the default encoding for files in Komodo's Preferences. While Komodo does not yet support editing outside of the English character set, non–English characters in existing files will be preserved.

## Komodo 1.2 beta 1 for Linux: November 2001

## Komodo 1.2 beta 2 for Windows: November, 2001

## Komodo 1.2 beta 1 for Windows: October, 2001

## Komodo 1.1: June, 2001

*PHP Debugging* Enjoy PHP syntax checking and debugging with Komodo. You can configure Komodo to debug PHP files locally or within a Web server environment. For more information, see Debugging Programs in the Online Help.

*XSLT Debugging* – This release supports XSLT debugging. You can view your XSLT file, your XML input file, and your output simultaneously. You can even open an XML file on your Web server and

transform the file while you debug your XSLT program. We've also improved the sample XSLT file in our Sample Project. For more information on XSLT debugging, see Debugging Programs in the Online Help.

*Tcl support* – You can now use Komodo to edit your Tcl files, including syntax coloring, code folding, autocomplete and calltips. We added a Tcl file to our Sample Project. If you have Tcl Dev Kit, you can enjoy Tcl syntax checking within Komodo. We plan to include stand–alone syntax checking that does not require Tcl Dev Kit in an upcoming release. Komodo does not yet support Tcl debugging.

*Improved Performance* – This release is based on the latest Mozilla tree, which includes an optimized code base. Performance improvements include faster installation and startup and faster response while you type or change files. Komodo responds much faster during debugging, when opening files and switching between files, and when loading the File Associations pane in the Preferences dialog.

*Support for more languages* – You can now use Komodo to edit Ruby and other languages, such as VB and SQL.

*Improvements for Komodo on Linux* – You can now resize the Komodo workspace, you can use fixed–width fonts, and you can debug Perl scripts if you have Perl 5.6 installed and configured.

*XML Autocomplete* – Komodo now features autocomplete for your XML files. When you type an open angle–bracket "<", Komodo lists the elements in your file. Komodo XML autocomplete also lists tag attributes and suggests attribute values for certain values. XML Autocomplete also helps you close your tags.

*Expanded Find and Replace dialog* – You can now use regular expressions in your find and replace strings, you can search through folded text, and more.

*Partial Unicode support* – Komodo supports ASCII, Latin–1 and Unicode (UTF–8, UCS–2 or UCS–4) encoding. If you use Komodo to edit a file that has a different character encoding than those mentioned above, non–English characters may be removed when you save the file.

## Komodo 1.0: April, 2001

## Komodo .1: November, 2000

# Known Issues

To view the status of outstanding Komodo issues, including those that have been fixed in this release, or to add comments or additional issues, please visit the Komodo Bug Database.

## Installation Issues

- Windows NT users may need to manually move their license file from `\WINNT\Profiles\[username]\ActiveState` to `\WINNT\Profiles\[username]\Application Data\ActiveState`.
- If you upgraded your system from Win9x/WinME to WinNT/2K, ensure that your `ComSpec` environment variable is configured to `%SystemRoot%\system32\cmd.exe` (for example, `C:\system32\cmd.exe`). There was a bug in the Microsoft Windows installer that did not update the variable from its original value of `command.com`.
- The Komodo installer requires up to 230 MB in your TMP directory (as indicated by the value of your 'TMP' environment variable) during installation, even if you plan to install Komodo to another drive. If you do not have the required space on this drive, manually set the 'TMP' environment variable to another directory with sufficient space. Ensure that you have enough space, temporarily, for installation.
- If you try to install Komodo on Windows and the MSI install fails with error 2355, your MSI file is corrupt. Please download Komodo again and re-run the MSI install.
- There are known issues regarding the installation of PHP on Windows Millennium systems; please refer to the PHP site for installation information.

## Startup Issues

- The first time Komodo is run after installation, it must register a number of components with the operating system. This causes the first launch to be considerably slower than subsequent launches.
- Certain programs may cause problems launching Komodo. For example, server programs such as SQL Server are known to be problematic. Also, Norton Anti-Virus (NAV), or more specifically, the File System Realtime Protection feature is problematic. If Komodo fails to load, examine the applications you are running, and try stopping those that might be conflicting with Komodo.

## Editing Issues

- The macro recorder will record events that it cannot handle, such as the opening of dialogs. The only dialog that can be opened via a macro is the Find dialog; other dialogs will cause the macro to stop.
- Application Data redirection on Windows: When you install Komodo, if you redirect the files that Komodo expects to have permissions to install in your Application Data folder, Komodo and Mozilla can close unexpectedly.
- Languages that are read right-to-left and Asian languages are not supported. All Latin and Cyrillic languages are fully supported.
- Cheyenne Antivirus Realtime Monitor and Komodo on Windows 9x – When RealMon is set to monitor outgoing files (or both incoming and outgoing files) Komodo's syntax checking doesn't function.

- On slow networks, users may notice performance degradation when editing files on network machines. Performance can be improved by disabling the Komodo function that checks if files on disk have changed. Use the Editor Preferences to disable this feature.
- Interpolation shortcuts in snippets are not executed when the snippet is inserted in the Editor Pane via dragging and dropping.

## Debugging Issues

- If the debug listener (***Debug/Listen for Remote Debugger***) is off, multithreaded applications may not run or debug as expected. Only the main thread operates through the debugger. To debug multithreaded applications, turn on debug listening prior to debugging. (Debug listening is turned on by default.)
- PHP configurations that use Zend Extensions (such as PHP Accelerator) are not compatible with the Komodo PHP debugger.
- Due to the way the core Perl interpreter works, it is not possible to step over "require" statements.
- You cannot use the 'freestanding' option when debugging Perl applications created with the PDK in Komodo. Instead, build a 'dependant' executable, which requires a local install of ActivePerl.
- The "Delete temp files after each run" option generates an error when debugging an application with the Perl Dev Kit interface.
- The variable watcher does not work when debugging `\\machine\d$\path\to\perl_script.pl`. It does work when opening the same file via a UNC path that does not include a '$' character.
- You cannot debug in a separate console on Windows Me/9x machines.
- If a script has syntax errors, the debugger can fail without warning.
- When debugging remote applications, Komodo fails if the remote process does not have valid stdout and stderr handles. GUI applications, such as those started with "wperl.exe" or "pythonw.exe", or those using the Pythonwin or wxPython frameworks, or those with certain embedded applications, can have invalid stdout and stderr handles. Until we resolve this issue, try to run and debug your remote program under perl.exe or python.exe.
- Python, XSLT and PHP debugging require TCP/IP to be installed and properly configured, even if you are debugging scripts locally. While TCP/IP is configured by default on most systems, early versions of Windows may require manual TCP/IP configuration.
- When debugging a GUI script in Komodo, adding a "watched variable" when *not* stopped at a breakpoint can cause Komodo to hang. You must manually terminate the script being debugged to stop Komodo from hanging. The problem occurs because the GUI script, while in its message loop, does not respond to Komodo's request for the variable value.
- Local debugging with the PHP debugger will fail if the debugger proxy is used.
- If the Komodo debugger is configured to use a specific port, when Komodo is shut down, the port is sometimes not immediately released. If Komodo is restarted before the port is released by the operating system, a message is displayed advising that the system is unable to bind that port. As a workaround, we suggest configuring port 0 as the Komodo debugging port and using the debugger proxy for remote debugging.

## Other Issues

- Komodo inherits a Mozilla bug whereby certain video drivers on Windows cause Komodo to crash. If you experience this behavior, upgrade your video driver to the latest version. If the problem persists, reduce the color definition assigned to the driver (Control Panel|Display|Settings).
- On Windows XP, the Windows task bar may show the old Komodo icon from a previous installation of Komodo. To fix this issue, delete the icon cache, located by default in C:\Documents and Settings\<username>\Local Settings\Application Data\IconCache.db.
- Komodo inherits a Mozilla bug regarding display on dual–monitor systems where the secondary monitor is to the left of the primary monitor (causing negative coordinate results). The Komodo display occasionally fails to refresh; Komodo must be stopped and restarted to fix the display.
- When using Komodo's Preview in Browser feature, users running Mozilla on Windows XP Pro, Windows 2000 Pro and Windows 98 may find that they cannot exit Komodo if Mozilla is still open. If this should occur, close all open browser windows before exiting Komodo.
- The Palm Desktop for Windows software makes exclusive use of the *Ctrl – Shift – T* key combination, thus making this combination unavailable in Komodo.
- When adding a GUI Builder project file (.ui) to a Komodo project file (.kpf), the files associated with the dialog are not included when the .ui file is added to a project. You must edit the dialog and save it before the files will be included under the dialog in the project.
- In the GUI Builder, the –image option for labels and buttons does not work.
- In the GUI Builder, there is no documentation for the `userinit` and `run` functions.
- A bug in CVS will cause WinCVS and TortoiseCVS to detect file changes when a Komodo project has merely been opened. The problem is likely a bug in the cvshome.org executable or in the cvsnt.org executable, which are used by both WinCVS and TortoiseCVS.
- When using the PDK 'Build standalone application' feature in Komodo with Perl 5.8.0 on a Linux installation where the environment is set to use UTF–8, you must add a module 'utf8' on the modules tab. This is the equivalent of 'perlapp ––add utf8'. This does not affect Perl 5.6.x or future versions of Perl 5.8.1 or higher.
- Komodo's integration with the Perforce commit/submit command cannot commit files that are not in the default changelist. These files must be submitted via an external interface (e.g. p4, P4Win). Note that any files checked out inside Komodo will be in the default changelist, so this limitation should only apply to users who already use an external interface to their Perforce repository.
- On Windows NT, some interactive commands may not work properly when run from the Command Output tab of the Output Pane. You must run these commands in a separate shell.
- Running interactive commands (especially "command.com") on Win98/ME through Komodo's "Run Command" feature can cause Komodo to hang. It is recommended that Win98/ME Komodo users run only simple commands using the Run Command feature.
- In file picker dialogs that display a list of files, when "All files" is specified, files that begin with a period are not displayed.
- Perforce client version 2001.1 and previous for Windows is known to hang when used for Komodo's Perforce integration. Upgrading to the most recent version of Perforce is known to fix the problem.

- If the Perforce connection cannot be established, checking the status of files in a Perforce repository will hang Komodo.
- The Output tab cuts off lines at 250 characters.
- Macros will not record certain commands, including (but possibly not limited to) Ctrl+Shift+b|r|e|d (toggle toolbars or button text), Ctrl+Shift+n (new default file), and View as Language menu items.
- If you are using CVS Source Code Control, note that the very first time you log in to a repository, cvs.exe fails to create the .cvspass file correctly and will return an error. Repeat the command to login a second time and correctly generate the file. This is a CVS bug.
- If you are using CVS Source Code Control on Windows 98 or Me, the environment variables HOMEDRIVE and HOMEPATH must be configured on your system. Typically, HOMEDRIVE is set to "c:", and HOMEPATH is set to "\".
- If you are using the Pop−Up Stopper ad−blocking program, it will close the Rx Toolkit window immediately after it is opened.
- When you schedule a new file to be added using CVS, CVS will not permit you to remove the file from the repository using the "revert changes" command.
- Users of the Japanese version of Windows XP may experience difficulties in starting Komodo.
- The *Open* field in the Open/Find Toolbar does not automatically display a drop−down list of directories when an UNC path is typed. Currently, the list is only displayed when a the path includes a subdirectory.
- Komodo cannot currently handle directory names and filenames that include the "%" (percent) character.

## Linux / Solaris Issues

Unless specified otherwise, these issues apply to both Linux and Solaris.

- On Solaris, GNU tar must be used to unpack the Komodo installation file. If you use the native Solaris tar version, you will get a checksum error when attempting to unpack the Komodo tar package, and any attempt to run the install script will fail.
- Installing Komodo on an NFS filesystem on Linux is not currently supported. As a workaround, you can set the KOMODO_USERDATADIR environment variable to a non−NFS directory in which your Komodo preferences and startup files are created and stored.
- Support for sub−pixel rendering ("antialiasing"), while available in GTK+ 2.0, is not supported in Komodo due to the performance impact.
- On Solaris, interactions between Komodo and Sun's CDE Window Manager may cause a modal child window (such as the Preferences page, or the Open | File dialog) to get hidden behind Komodo's main window. The main Komodo window will be hung waiting for the (now unreachable) modal window to be closed. The problem is likely related to a specific CDE configuration "focus follows mouse" or similar.
- The *Fonts and Colors* page in the Preferences displays the same list of fonts in both the Fixed and Proportional lists. There is no programmatic way to identify whether a font is proportional or not on GTK; therefore, users must know the properties of the individual fonts when modifying these values.

- Install Komodo into a directory path that only includes alphanumeric characters. Komodo is known to have trouble with paths that include spaces and some non−alphanumeric characters.
- Filenames or paths containing non−ASCII characters cannot be opened remotely.
- Key bindings defined in the window manager (such as KDE) take precedence over Komodo key bindings. In the case of conflicts, you must either change the Komodo key bindings or the window manager key bindings.
- You cannot relocate an existing Komodo installation to a new directory. You must uninstall Komodo from the existing location and reinstall it in the new location.
- When using the PHP Configuration Wizard, you must have write access to any directories you specify in the wizard.
- Red Hat Linux 9.0 is known to have threading library bugs in its glibc that may cause Komodo to hang in certain situations. The recommended solution is to upgrade to the latest glibc for Red Hat Linux 9.0.

# Komodo FAQ

- [Komodo doesn't start](#)
- [I can't see my Left or Right Pane](#)
- [I can't see my Bottom Pane](#)
- [I want to maximize my Editor Pane](#)
- [How do I know if I'm debugging?](#)
- [How do I know if I'm editing?](#)
- [How can I add command−line arguments to my program for debugging?](#)
- [Komodo crashes. What can I do?](#)
- [Why is Komodo so big?](#)
- [I already have Mozilla. Why do I need to have two versions?](#)
- [I'm having trouble debugging PHP. What do I do?](#)
- [How do I emulate sessions in PHP debugging?](#)
- [How do I configure Virtual Hosting on an Apache Web server?](#)
- [I moved my Komodo installation on Linux, and am now getting Perl debugging errors.](#)
- [How do I prevent the dialog from displaying every time I start the debugger?](#)
- [Why do I get a CGI security alert when debugging PHP?](#)
- [I'm using Windows 98. When I start Komodo, I get the error "Page fault in MSVCRT.DLL".](#)
- [The Check Configuration window reports that a language installed on my system is not available. Why?](#)
- [My screen goes black for a second or two whenever I open files for which Komodo performs background syntax checking. Why?](#)
- [Why does VPM display a "Failure to Connect To Web Server" message?](#)
- [Why can't I find the module that I want using the Visual Package Manager (VPM)?](#)
- [How can I run additonal CVS commands from within Komodo?](#)

# Komodo doesn't start

If Komodo doesn't start, there could be one of several issues.

- Do you have a license installed?

Komodo needs a license to become functional. If you have a Beta release, we include a trial license with the package. If you have a final release (non−Beta), you can download a license from [ActiveState](#).

- Does your username have non−ASCII characters?

Komodo keeps your user preferences in an Application Data directory on Windows machines. For example, <installdir>\Documents and Settings\<username>\Application Data\ActiveState\Komodo.

In Komodo 1.0 and earlier, Komodo did not understand non−ASCII characters in your username, so when Komodo looked for your preferences file during startup, Komodo got confused. This issue was fixed in Komodo 1.1 Beta 1.

- Do you have Norton Anti–Virus (NAV) installed, or more specifically, the File System Realtime Protection feature enabled?

The problematic relationship between Komodo and Norton Anti–Virus' File System Realtime Protection feature is a known issue, which we are working to remedy. In the meantime, you can disable `NAV Corp 7.5 FSRP` before running Komodo, and then re–enable it after Komodo starts.

# I can't see my Left or Right Pane

One or more panes may be hidden.

To view the Left Pane, click the ***Show/Hide Left Pane*** button on the toolbar, use the options on the ***View*** menu, or use the associated key binding.

# I can't see my Bottom Pane

The Bottom Pane appears below the Editor Pane during debugging. If you can't see your Bottom Pane, it may be hidden.

To view the Bottom Pane, click the ***Show/Hide Bottom Pane*** button on the toolbar, use the options on the ***View*** menu, or use the associated key binding.

For more information, see Debugging Programs

# I want to maximize the Editor Pane

I like to see the Left and Right Panes and the Bottom Pane, but right now I want to maximize my Editor Pane to get some coding done. How can I maximize my Editor Pane?

To maximize your Editor Pane, hide the other panes in the Komodo workspace:

1. Click the close arrow button that appears in the top right corner of each of these panes.

# How do I know if I'm debugging?

When Komodo is debugging, the title of the Komodo workspace includes an indication of the state of the debugger. If the debugger is running, the title looks similar to ***[pathname\filename] – ActiveState***

*Komodo – Debugger is running*. If the debugger has hit a breakpoint, the title looks similar to *[pathname\filename] – ActiveState Komodo – Debugger is in Break Mode*.

# How do I know if I'm editing?

You are editing any time you're not formally debugging. When Komodo is editing, the title of the Komodo workspace is *[pathname\filename] – ActiveState Komodo*.

# How can I add command–line arguments to my program for debugging?

If you want to send add command–line arguments to your program for debugging, you can add these using the Debugger Launch Options dialog:

1. Go to the *Debug* menu and select *Start* or press F5.
2. In the Debugger Launch Options dialog, select the directory you want to begin debugging your program in. Click *Browse* and navigate to the appropriate directory.
3. In the same Debugger Launch Options dialog, enter your command–line arguments. These are sent to the script and not to the interpreter. Separate the arguments with spaces.
4. Click *OK*.

# Komodo crashes. What can I do?

If Komodo crashes, please create the following error log files, find the files, verify their contents and send them to us so we can determine what happened:

- stderr.tmp
- stdout.tmp

## Step 1: Creating the error log files

To create the error logs on *Windows*:

1. Go to the *Start* menu and select *Run*.
2. Enter `cmd`
3. In the command box, change to the Komodo directory. The default is `C:\Program Files\Komodo-x.x`, where "x.x" is the Komodo version.
4. In the Komodo directory, enter `Komodo -v` and press Enter.

This starts Komodo.

5. Perform the task that caused the original Komodo error.
6. Close Komodo, or let it crash.

This creates the error log files and puts them in a directory as indicated in the table in Step 2: Locating the error log files.

*Note* – If Komodo "cancelled" itself, open the Task Manager and close the Mozilla process.

To create the error logs on *Linux*:

1. In the command box, change to the Komodo directory. The default is ~/.komodo.
2. In the Komodo directory, enter Komodo -v and press Enter.
   This starts Komodo.
3. Perform the task that Komodo didn't like.
4. Close Komodo, or let it crash.

This creates the error log files and puts them in your ~/.komodo directory, as indicated in the table in Step 2: Locating the error log files.

*Note* – If Komodo cancelled itself, close the Mozilla process.

To close the Mozilla process on Linux:

1. Run ps ux to identify the Mozilla process id.
2. Run kill <process id> to close that process.

## Step 2: Locating the error log files

Komodo stores the error log files in a directory beneath the Application Data directory. The location of this directory varies according to the version of Windows, and whether or not User Profiles are enabled on your system. Use the following table as a guide for the file location. In the examples below, "x.x" refers to the version of Komodo that you are using.

| *System* | *Location* | *or, if User Profiles is enabled,* |
|---|---|---|
| Windows XP | C:\Windows\Application Data\ActiveState\Komodo\x.x | C:\Documents and Settings\<username>\Application Data\ActiveState\Komodo\x.x |
| Windows 2000 | C:\Windows\Application Data\ActiveState\Komodo\x.x | C:\Documents and Settings\<username>\Application Data\ActiveState\Komodo\x.x |
| Windows NT | C:\Windows\Application Data\ActiveState\Komodo\x.x | C:\WINNT\Profiles\<username>\Application Data\ActiveState\Komodo\x.x |

| Windows Me | C:\Windows\Application Data\ActiveState\Komodo\x.x | C:\Windows\Profiles\<username>\Application Data\ActiveState\Komodo\x.x |
|---|---|---|
| Windows 98 | C:\Windows\Application Data\ActiveState\Komodo\x.x | C:\Windows\Profiles\<username>\Application Data\ActiveState\Komodo\x.x |
| Windows 95 | C:\WINDOWS\Application Data\ActiveState\Komodo\x.x | C:\WINDOWS\Profiles\<username>\Application Data\ActiveState\Komodo\x.x |
| Linux | ~/.komodo | n/a |

## Step 3: Verifying and sending the files to ActiveState

To send the error log files to ActiveState:

1. Locate the files.
2. Verify that the files are not blank by viewing them with a text editor.
3. Create a bug describing what happened just before the crash in the ActiveState bug database. (If you do not already have an ASPN or ActiveState bug database account, you can open one by selecting "*join*".)
4. Once the bug has been created, add the error log files by selecting **Create** in the **Attachments and Dependencies** section of the bug report.

# Why is Komodo so big?

Because Komodo is built on the Mozilla framework, it is necessary for us to include the Mozilla build that exactly matches the development version of Komodo. For that reason, even if you have Mozilla on your system, Komodo installs the Mozilla version that it requires.

Another sizeable component of Komodo is language support. Komodo is so tightly integrated with Perl, Python and PHP that it is necessary to include components of those languages, at specific version levels, for debugger and editor support.

# I already have Mozilla. Why do I need to have two versions?

When ActiveState develops a Komodo release, the work is based upon a specific version of Mozilla. During the development process, we upgrade the level of Mozilla used by Komodo, but this process requires considerable testing to ensure that no functionality is lost. Additionally, we add some custom components to the Mozilla tree that are used by Komodo. For these reasons, we recommend that you do not replace the Mozilla version included with Komodo with a later Mozilla version.

# I'm having trouble debugging PHP. What do I do?

If you receive an error message when attempting to debug a PHP program or if the debugging process does not proceed as expected, verify that you have installed PHP and the Xdebug extension as per the instructions in the Debugging PHP documentation, then check the following:

## Confirm PHP Configuration

1. *xdebug:* in the command or shell window, enter `php -m`. "xdebug" should be listed as both a regular module and a zend extension. If this is not the case, your configuration is incorrect. See "Common PHP Configuration Problems" below.
2. *Syntax Checking:* in Komodo, select *Edit/Preferences*. Click on *Smart Editing*, and ensure that "Enable background syntax checking" is checked. Open a PHP file and enter something that is syntactically incorrect, such as:

   ```
   <?
   asdf
   echo test;
   ?>
   ```

   Komodo should display a red squiggly line under `echo test;`. If it does not, it indicates that Komodo is not able to communicate with the PHP interpreter.
3. *Debug:* if steps one and two were successful, ensure that the debugger is functioning by opening a PHP program and debugging it. Ensure that the correct Preferences are configured for PHP.

If any of the steps above were unsuccessful, proceed to the next section.

## Common PHP Configuration Problems

- *Multiple PHP executables on one machine:* in Komodo's Preferences, explicitly specify the PHP interpreter configured in your php.ini file. The location of the php.ini file can also be explicitly set.
- *Verify the PHP version:* PHP 4.0.5 or greater is required for PHP syntax checking. PHP 4.3.1 or greater is required to debug PHP programs.
- *Verify Xdebug library specification:* The location of *xdebug.dll* (Windows) or *xdebug.so* (Linux and Solaris) must be defined the *php.ini* file, for example:

    ♦ *Windows:* zend_extension_ts=C:\php−4.3.7\extensions\php_xdebug.dll
    ♦ *Linux:* zend_extension=/php−4.3.7/extensions/php_xdebug.dll

- Ensure that the Xdebug extension is configured correctly in the *php.ini* file as per the Remote PHP Debugging instructions.

### Windows–Specific Configuration Issues

- *Windows 2000 upgrade:* if you upgraded to Windows 2000 from an earlier version of Windows, check the value of the "COMSPEC" variable in the "System variables" (as described above). It should point to C:\WINNT\system32\cmd.exe, and not command.com. If you must change the variable, reboot your system.
- There are known issues regarding the installation of PHP on Windows Millennium systems; please refer to the PHP site for installation information.

### Version Error Messages

If you receive a dialog with the following text:

```
Warning
xdebug: Unable to initialize module
Module compiled with debug=0, thread-safety=1 module API=20001222
PHP compiled with debug=0, thread-safety=1 module API=20001222
These options need to match
```

... download an updated version of xdebug.dll (Windows) or xdebug.so (Linux) from the Xdebug.org site.

# How do I emulate sessions in PHP debugging?

Though it is possible to emulate sessions in local debugging mode, this requires pre–knowledge of session keys, and how those session keys are communicated to PHP.

It is easier to debug sessions using remote debugging. Run the script under a web server and start the debugging session from a web browser. Komodo intercepts the session and debugs it. All session data is available and modifiable through the Variable tabs.

# How do I configure Virtual Hosting on an Apache Web server?

Virtual Hosting is an Apache feature for maintaining multiple servers on the same machine, differentiating them by their apparent hostname. For example, a single machine could contain two servers, "www.yourdomain.com" and "debug.yourdomain.com".

If you have configured your Apache installation to use Virtual Hosting (see httpd.apache.org/docs/vhosts/), you can add directives to your VirtualHost sections to specify how Komodo's PHP debugger extension operates for those hosts. Use the "php_admin_value" to set specific

debugger settings for that virtual host. Here is an example:

```
NameVirtualHost *
<VirtualHost *>
php_admin_value xdebug.enabled 0
DocumentRoot "/Apache/htdocs/"
ErrorLog logs/www.error.log
Servername www.yourdomain.com
</VirtualHost>

<VirtualHost *>
php_admin_value xdebug.enabled 1
DocumentRoot "/Apache/htdocs/"
ErrorLog logs/debug.error.log
Servername debug.yourdomain.com
</VirtualHost>
```

This will enable debugging under debug.yourdomain.com, but not under www.yourdomain.com. You can additionally configure the Virtual Host to use a specific machine for remote debugging:

```
<VirtualHost *>
php_admin_value xdebug.enabled 1
php_admin_value xdebug.host komodo.yourdomain.com
DocumentRoot "/Apache/htdocs/"
ErrorLog logs/debug.error.log
Servername debug.yourdomain.com
</VirtualHost>
```

For more information on configuring Virtual Hosting under Apache, see the Apache documentation at httpd.apache.org/docs/.

# I moved my Komodo installation on Linux, and am now getting Perl debugging errors.

On Linux, you cannot relocate an existing Komodo installation to a new directory. You must uninstall Komodo from the existing location and reinstall it in the new location. See Uninstalling Komodo on Linux for instructions.

# How do I prevent the dialog from displaying every time I start the debugger?

To prevent the debugger dialog from appearing each time you start the debugger, hold down the 'Ctrl' key when you start the debugger. (For example, press 'Ctrl'+'F5' rather than 'F5' to start debugging.)

# Why do I get a CGI security alert when debugging PHP?

The CGI security alert only occurs when you compile PHP with −−enable−cgi−force−redirect. That compilation directive forces PHP to check if it is being run as a CGI by looking at environment variables commonly available only under a CGI environment. If they exist, it looks for another environment variable that is reliably available ONLY under Apache, REDIRECT_STATUS (or HTTP_REDIRECT_STATUS under Netscape/iPlanet). If that environment variable does not exist, the security alert is generated.

To run your compilation of PHP under Komodo with CGI emulation, you have to add a CGI environment variable called REDIRECT_STATUS with any value.

# I'm using Windows 98. When I start Komodo, I get the error "Page fault in MSVCRT.DLL".

If your system generates the above error when starting Komodo, you should install the latest "critical" fixes for Windows 98. Use the *Windows Update* utility to download and install the upgrade packages, or refer to the Microsoft Support Web site.

# When I click Check Configuration on the Start Page, Komodo reports that a language that is installed on my system is not available. Why?

In order for Komodo to detect the presence of a language installed on your system, the location of the language interpreter must be specified in your system's PATH environment variable. If the Komodo Start Page states that a language is "Not Functional", or if the Komodo Preferences say that the language interpreter is not found on your system, check that the interpreter is specified in your PATH.

# My screen goes black for a second or two whenever I open files for which Komodo performs background syntax checking. Why?

Komodo launches a process as part of the background syntax checking that can cause a full screen command prompt to momentarily appear on some Windows systems. You can make the process invisible by editing the properties for the command prompt window. On the Windows *Start* menu, right−click the *Command Prompt* item, and select *Properties*. Select the *Options* tab, and change the Display options to *Window*.

## Why does VPM display a "Failure to Connect To Web Server" message?

The Visual Package Manager (VPM) runs as a local HTTP server on a system−assigned port.

If you are running firewall software that blocks connectivity to localhost, your browser may not be able to connect to the VPM server. Since the port used by VPM is assigned by the system and changes each time VPM is run, you cannot set a specific port number in your access rules. If possible, configure the firewall to allow TCP connections from `localhost` and `127.0.0.1` on any port. Consult the firewall documentation for information on changing access rules.

If your browser is configured to use a proxy, ensure that the proxy is bypassed for local addresses (e.g. "Bypass Proxy Server for local addresses" in Internet Options on Windows), or that `localhost` and `127.0.0.1` are listed in the proxy exception list.

## Why can't I find the module that I want using the Visual Package Manager (VPM)?

Modules stored in Perl Package Manager (PPM) repositories are built by ActiveState for use with the Visual Package Manager (VPM). Not all Perl packages have PPM and VPM equivalents for every platform and Perl version. To check the build status of a package, see ActiveState's ASPN website.

If a Perl module is not available from the PPM repository, it can be manually installed and built, for example from the CPAN module repository. For instructions on using modules from CPAN, see the ActivePerl documentation.

## How can I run additonal CVS commands from within Komodo?

Komodo can be used to check out, add, remove, compare, submit and revert files in a CVS repository. CVS offers additional commands such as import, checkout, history, annotate, rdiff and watch which can

be put into Run Commands and saved to a project or the Toolbox. For example, the following `cvs import` command prompts for the User, Host, Module, Project and Version to import:

```
cvs -d :ext:%(ask:User)@%(ask:Host):%(ask:Path) import %(ask:Module:)
%(ask:Project:) %(ask:Version:)
```

Alternatively, the `%(ask:...)` interpolation shortcut could be populated with defaults or replaced with static values:

```
cvs -d :ext:%(ask:User:jdoe)@myhost:/var/cvsroot import %(ask:Module:)
%(ask:Project:MyProject)
```

CVS reqires a real terminal for adding change descriptions. Be sure to set **Run in: New Console** in the command's properties.

# License and Copyrights

Copyright ©2004 ActiveState Corporation. ActiveState Corp. is a division of Sophos Plc. All Rights Reserved.

*ActiveState* is a registered trademark of ActiveState Corp. *Komodo*, the *Perl Dev Kit*, the *Tcl Dev Kit*, *ActivePerl*, *ActivePython*, *ActiveTcl* and *ASPN* are trademarks of ActiveState Corporation.

All other products mentioned are trademarks or registered trademarks of their respective companies.

The table of contents in the Komodo documentation is based on an open–source project written by by Dieter Bungers, GMD, German National Research Center for Information Technology, Department for Innovative Consulting and Development (IBE). See http://www.d.umn.edu/ece/publications/handbook/Generate/OrigDoc.htm for more information.

ActiveState Komodo licenses are issued on a per–user basis. A Komodo license may be installed on more than one system or platform, as long as the licensee is the sole user of the software.

# Komodo License

ACTIVESTATE KOMODO LICENSE AGREEMENT

Please read carefully: THIS IS A LICENSE AND NOT AN AGREEMENT FOR SALE. By using and installing ActiveState Corporation's Software or, where applicable, choosing the "I ACCEPT..." option at the end of the License you indicate that you have read, understood, and accepted the terms and conditions of the License. IF YOU DO NOT AGREE WITH THE TERMS AND CONDITIONS, YOU SHOULD NOT ATTEMPT TO INSTALL THE SOFTWARE. If the Software is already downloaded or installed, you should promptly cease using the Software in any manner and destroy all copies of the Software in your possession. You, the user, assume all responsibility for the selection of the Software to achieve your intended results and for the installation, use and results obtained from the Software. If you have any questions concerning this, you may contact ActiveState via email at Sales@ActiveState.com.

This ActiveState License ("License") is made between ActiveState Corporation ("ActiveState") as licensor, and you, as licensee, as of the date of your use of the Software (the Software is in use on a computer when it is loaded into the RAM or installed into the permanent memory (e.g., hard disk or other storage device) of that computer.).

This License reflects ActiveState's intent to retain full ownership of and control of the use and distribution of ActiveState Komodo, the License Key (as hereinafter defined) and other applicable software (collectively the "Software").

1. License Grant. Subject to the terms and conditions of this License, ActiveState grants to you a personal, non-exclusive, non-transferable, and limited license to use the Software solely to create, compile, test and deploy, in source or object code form, your own application programs ("Works"). You may create redistributable applications based on your unique

development work using the Software and you may sublicense to end users of such Works ("End Users") the personal, non-exclusive, non-transferable right to install and execute the files, and/or libraries that are necessary to use the Works created using the Software. Use of such files and/or libraries by such End Users is limited to runtime purposes only. You may not provide any End User with access to the development or interactive capabilities of the Software libraries or technology, nor may you expose the base programming language(s) as a scripting language within the Works to any such End User. When you pay the license fee established by ActiveState, you will receive a license key (the "License Key") from ActiveState that authorizes you to use the Software only in the following specific contexts:

(i) Commercial Use. If your License Key authorizes Commercial Use, you may use the software on more than one computer or on a network so long as you are the sole user of the Software. (A "network" is any combination of two or more computers that are electronically linked and capable of sharing the use of a single software program.) You will obtain a separate license for each additional user of the Software (whether or not such users are connected on a network). You may make only one copy of the Software for archival or backup purposes. You are not permitted to sell, lease, distribute, transfer, sublicense, or otherwise dispose of the Software, in whole or in part, for any form of actual or potential commercial gain or consideration.

(ii) Evaluation (Trial) Use. If your License Key authorizes Evaluation (Trial) Use you may use the software only for evaluation purposes without payment of the License Fee for a period of no more than twenty-one (21) days from the date of download.

(iii) Non-commercial Use. If your License Key authorizes Non-commercial Use, you may use the software in a teaching or learning environment only. You are not permitted to sell, lease, distribute, transfer, sublicense, or otherwise dispose of the Software, in whole or in part, for any form of actual or potential commercial gain or consideration.

(iv) Other Use. Any use of the software other than the uses specified above requires a separate license from ActiveState and the payment of additional license fees as determined by ActiveState. Such additional redistribution options are available, at ActiveState's sole discretion, on a case-by-case basis. Contact Sales@ActiveState to discuss any redistribution options not covered by this license agreement. ActiveState reserves all rights not expressly granted to you herein.

2. Termination. This License Agreement is effective until terminated. ActiveState may terminate this License immediately and without prior notice if you breach any term of this License or for any other commercially reasonable ground. In the event of any termination or expiration, you agree to immediately destroy and/or erase the original and all copies of the Software, any accompanying documentation and License Keys and to discontinue their use and you will not retain or store the Software or any copies thereof, in any form or medium.

3. Proprietary Rights. The Software is licensed, not sold, to you. ActiveState reserves all rights not expressly granted to you. Ownership of the Software and its associated proprietary rights, including but not limited to patent and patent applications, are retained by ActiveState. The Software is protected by the copyright laws of Canada and the United States and by international treaties. Therefore, you must comply with such laws and

treaties in your use of the Software. You agree not to remove any of
ActiveState's copyright, trademarks and other proprietary notices from the
Software.

4. Distribution. Except as may be expressly allowed in Section 1, or as
otherwise agreed to in a written agreement signed by both you and
ActiveState, you will not distribute the Software, either in whole or in
part, in any form or medium.

5. Transfer and Use Restrictions. You may not sell, license, sub-license,
lend, lease, rent, share, assign, transmit, telecommunicate, export,
distribute or otherwise transfer the Software to others, except as expressly
permitted in this License Agreement or in another agreement with
ActiveState. In order to use the Software you will be required to obtain a
License Key and agree to this License Agreement for the use of the Software.
You will not disclose or provide access to your License Key to any other
person or entity. You must comply with all applicable Canadian and other
export control laws in your use of the Software. Except as may be expressly
permitted above, you may not modify, reverse engineer, decompile, decrypt,
extract or otherwise disassemble the Software.

6. NO WARRANTY. ACTIVESTATE MAKES NO WARRANTIES WHATSOEVER REGARDING THE
SOFTWARE AND IN PARTICULAR, DOES NOT WARRANT THAT THE SOFTWARE WILL FUNCTION
IN ACCORDANCE WITH THE ACCOMPANYING DOCUMENTATION IN EVERY COMBINATION OF
HARDWARE PLATFORM OR SOFTWARE ENVIRONMENT OR CONFIGURATION, OR BE COMPATIBLE
WITH EVERY COMPUTER SYSTEM. IF THE SOFTWARE IS DEFECTIVE FOR ANY REASON, YOU
WILL ASSUME THE ENTIRE COST OF ALL NECESSARY REPAIRS OR REPLACEMENTS.

7. DISCLAIMER. ACTIVESTATE DOES NOT WARRANT THAT THE SOFTWARE IS FREE FROM
BUGS, DEFECTS, ERRORS OR OMISSIONS. THE SOFTWARE IS PROVIDED ON AN "AS IS"
BASIS AND ACTIVESTATE MAKES NO OTHER WARRANTIES OR CONDITIONS, EXPRESS OR
IMPLIED, WITH RESPECT TO THE SOFTWARE OR ANY ACCOMPANYING ITEMS INCLUDING
WITHOUT LIMITATION THE IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY
OR FITNESS FOR A PARTICULAR PURPOSE. SOME JURISDICTIONS DO NOT ALLOW THE
EXCLUSION OF IMPLIED WARRANTIES, IN WHICH CASE THE ABOVE EXCLUSIONS MAY NOT
APPLY TO YOU.

8. LIMITATION OF LIABILITY. ACTIVESTATE WILL HAVE NO LIABILITY OR OBLIGATION
FOR ANY DAMAGES OR REMEDIES, INCLUDING, WITHOUT LIMITATION, THE COST OF
SUBSTITUTE GOODS, LOST DATA, LOST PROFITS, LOST REVENUES OR ANY OTHER
DIRECT, INDIRECT, INCIDENTAL, SPECIAL, GENERAL, PUNITIVE OR CONSEQUENTIAL
DAMAGES, ARISING OUT OF THIS LICENSE OR THE USE OR INABILITY TO USE THE
SOFTWARE. IN NO EVENT WILL ACTIVESTATE'S TOTAL AGGREGATE LIABILITY (WHETHER
IN CONTRACT (INCLUDING FUNDAMENTAL BREACH), WARRANTY, TORT (INCLUDING
NEGLIGENCE), PRODUCT LIABILITY, INTELLECTUAL PROPERTY INFRINGEMENT OR OTHER
LEGAL THEORY) WITH REGARD TO THE SOFTWARE AND/OR THIS LICENSE EXCEED THE
LICENSE FEE PAID BY YOU TO ACTIVESTATE. FURTHER, ACTIVESTATE WILL NOT BE
LIABLE FOR ANY DELAY OR FAILURE TO PERFORM ITS OBLIGATIONS UNDER THIS
LICENSE AS A RESULT OF ANY CAUSES OR CONDITIONS BEYOND ACTIVESTATE'S
REASONABLE CONTROL.

9. Intellectual Property Claims. Subject to the limitation of liability set
out in Section 8 above, ActiveState will indemnify and hold you harmless
against any damages finally awarded against you pursuant to a judicial
proceeding, to the extent such proceeding is based upon an infringement of
any valid U.S. or Canadian patent issued as at the date of your acceptance
of this License Agreement, or of any valid U.S. or Canadian copyright, by

the Software provided that you:

(i) give written notice of the claim promptly to ActiveState;

(ii) give ActiveState sole control of the defense and settlement of the claim;

(iii) provide to ActiveState all available information and assistance; and

(iv) have not compromised or settled such claim.

ActiveState will have no obligation under this Section for any claims which result through no fault of ActiveState, including: (i) the use of the Software in combination with any non-ActiveState approved software; or (ii) modification of the Software by you or anyone other than ActiveState. This Section sets forth the entire liability of ActiveState and your exclusive remedies for claims of infringement involving the Software.

10. U.S. Government Restricted Rights Legend. If the Software is acquired by any agency or other part of the U.S. government in a transaction subject to the Federal Acquisition Regulations or the Defense Federal Acquisition Regulations, the Software is furnished with Restricted Rights. Use, duplication, or disclosure of the Software by the U.S. government is subject to all applicable restrictions set forth in such Regulations, as amended from time to time, including subparagraphs (c)(1) and (2) of the Commercial Computer Software – Restricted Rights clause at Section 48 C.F.R. 52.227-19.

11. Entire Agreement. This License and the Usage License Agreement constitutes the entire agreement between you and ActiveState regarding the Software and all accompanying documentation. If any provision is found to be invalid by a court of competent jurisdiction, the balance of the provisions will remain in full force and effect. This License will be governed by and construed in accordance with the laws of the Province of British Columbia, Canada, excluding its conflict of laws rules. The parties hereby attorn to the jurisdiction of the courts of the Province of British Columbia in the event of any dispute hereunder. The provisions of the U.N. Convention on Contracts For The International Sale of Goods (1980) and any successor Convention, will not apply to this License.

12. Inurement. The rights, restrictions, limitations, disclaimers and remedies granted to, retained by, or for the benefit of ActiveState will inure to the benefit of and will be enforceable by ActiveState and its licensors, successors and assigns. You may not assign your rights, obligations and interest in and to this License without the prior written consent of ActiveState. The obligations, covenants and rights which apply to you will inure to your benefit and will be binding on you and your permitted successors and assigns.

BY INSTALLING THE SOFTWARE, OR, WHERE APPLICABLE, CHOOSING THE "I ACCEPT..." OPTION, YOU INDICATE THAT YOU HAVE READ, UNDERSTOOD AND ACCEPT THE TERMS AND CONDITIONS OF THE LICENSE. IF YOU DO NOT AGREE WITH THE TERMS AND CONDITIONS, YOU SHOULD NOT ATTEMPT TO INSTALL THE SOFTWARE.

# Sending Feedback

We welcome your feedback, suggestions, feature requests and bug reports.

## Comments and Feature Requests

If you tried Komodo and you would like to see a certain feature, or you would like to tell us how Komodo met or didn't meet your coding needs, send us an email.

Please send suggestions and feature requests to **Komodo−feedback@ActiveState.com**

## Reporting Bugs

If you have found a bug, we'd like to know about it. First, view our on−line bug database and see if your bug has already been reported. If you can't find your issue, you can submit a bug report.

To view our bug database and submit a bug report:

1. Go to our on−line bug database at http://bugs.ActiveState.com/Komodo
   or
   From the Komodo Workspace, go to the *Help* menu and select *Komodo Bug Database*.
2. Click *View Bugs*. You can apply filters and then sort the result by column.
3. If necessary, click *Search* and enter a keyword for your bug.
4. If you can't see your bug, click *Submit a bug*.
5. Complete the on−line form with as much detail as possible. This helps us identify the problem.
6. Click *Submit Bug Report*.
   If you make a mistake, click *Reset*.

Your bug will be assigned an ID and you can monitor the progress of your bug on this Web site. We will email you when your bug is resolved.

---