Fscript Reference Manual

Release 3.5 of Forte™ 4GL

Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, CA 94303 U.S.A. 1-800-786-7638

Part No. 806-6670-01
October 2000, Revision A

Contents

2 Fscript Commands

Preface

The *Fscript Reference Manual* contains usage and reference information for the Forte Fscript utility.

The Fscript utility is a command-line interface that you can use to import and export TOOL code, interact with your development repository, and configure and make application distributions.

This manual assumes that you are familiar with writing TOOL code and interacting with the development repository. You should have the following manuals available when you use this book:

- *TOOL Reference Manual*, for information about writing TOOL code
- *A Guide to the Forte 4GL Workshops*, for information about tasks that you can perform when you interact with the repository, import or export TOOL code, and configure and make application distributions

Organization of This Manual

This manual describes how to start the Fscript utility and what commands you can use to perform a task. This manual also describes each Fscript command in detail.

The book is organized into the following chapters:

Chapter	Description
Chapter 1, "Using Fscript"	Explains how to start the Fscript utility, write Fscript scripts, and perform tasks using Fscript commands.
Chapter 2, "Fscript Commands"	Describes each Fscript command.
Appendix A, "Fscript Command Summary"	Provides the syntax and a brief description for each Fscript command.
Appendix B, "Memory and Logger Flags"	Describes the logger and memory options that you can specify with Fscript.
Appendix C, "Undocumented Fscript Commands"	Lists Forte Release 1 Fscript commands that have been replaced by other commands.

Conventions

This manual uses standard Forte documentation conventions in specifying command syntax and in documenting TOOL code.

Command Syntax Conventions

The specifications of command syntax in this manual use a “brackets and braces” format. The following table describes this format:

Format	Description
bold	Bold text is a reserved word; type the word exactly as shown.
italics	Italicized text is a generic term that represents a set of options or values. Substitute an appropriate clause or value where you see italic text.
UPPERCASE	Uppercase text represents a constant. Type uppercase text exactly as shown.
underline	Underlined text represents a default value.
vertical bars |	Vertical bars indicate a mutually exclusive choice between items. See braces and brackets, below.
braces { }	Braces indicate a required clause. When a list of items separated by vertical bars is enclosed in braces, you must enter one of the items from the list. Do not enter the braces or vertical bars.
brackets [ ]	Brackets indicate an optional clause. When a list of items separated by vertical bars is enclosed by brackets, you can either select one item from the list or ignore the entire clause. Do not enter the brackets or vertical bars.
ellipsis …	The item preceding an ellipsis may be repeated one or more times. When a clause in braces is followed by an ellipsis, you can use the clause one or more times. When a clause in brackets is followed by an ellipsis, you can use the clause zero or more times.

TOOL Code Conventions

Where this manual includes documentation or examples of TOOL code, the TOOL code conventions in the following table are used.

Format	Description
parentheses ( )	Parentheses are used in TOOL code to enclose a parameter list. Always include the parentheses with the parameter list.
comma ,	Commas are used in TOOL code to separate items in a parameter list. Always include the commas in the parameter list.
colon :	Colons are used in TOOL code to separate a name from a type, or to indicate a Forte name in a SQL statement. Always include the colon in the type declaration or statement.
semicolon ;	Semicolons are used in TOOL code to end a TOOL statement. Always type a semicolon at the end of a statement.

Command Line and Code Examples

Examples of installation script prompts, operating system command lines, and programming code are shown in a monospaced font set off in a shaded area. Here’s an example of a UNIX command line:

```
cd /usr/sbin/slibclean
```

Here’s an example of commands being entered at Forte’s Fscript prompt:

```
fscript> UsePortable
fscript> SetPath %{FORTE_EP_WKDIR}
fscript> Include dmathtm.fsc
```

The Forte Documentation Set

Forte produces a comprehensive documentation set describing the libraries, languages, workshops, and utilities of the Forte Application Environment. The complete Forte Release 3 documentation set consists of the following manuals in addition to comprehensive online Help.

Forte 4GL

- *A Guide to the Forte 4GL Workshops*
- *Accessing Databases*
- *Building International Applications*
- *Escript and System Agent Reference Manual*
- *Forte 4GL Java Interoperability Guide*
- *Forte 4GL Programming Guide*
- *Forte 4GL System Installation Guide*
- *Forte 4GL System Management Guide*
- *Fscript Reference Manual*
- *Getting Started With Forte 4GL*
- *Integrating with External Systems*
- *Programming with System Agents*
- *TOOL Reference Manual*
- *Using Forte 4GL for OS/390*

Forte Express

- *A Guide to Forte Express*
- *Customizing Forte Express Applications*
- *Forte Express Installation Guide*

Forte WebEnterprise and WebEnterprise Designer

- *A Guide to WebEnterprise*
- *Customizing WebEnterprise Designer Applications*
- *Getting Started with WebEnterprise Designer*
- *WebEnterprise Installation Guide*

Forte Example Programs

In this manual, we often include code fragments to illustrate the use of a feature that is being discussed. If a code fragment has been extracted from a Forte example program, the name of the example program is given after the code fragment. If a major topic is illustrated by a Forte example program, reference will be made to the example program in the text.

These Forte example programs come with the Forte product. They are located in subdirectories under $FORTE_ROOT/install/examples. The files containing the examples have a .pex suffix. You can search for TOOL commands or anything of special interest with operating system commands. The .pex files are text files, so it is safe to edit them, though you should only change private copies of the files.

Viewing and Searching PDF Files

You can view and search 4GL PDF files directly from the documentation CD-ROM, store them locally on your computer, or store them on a server for multiuser network access.

Note You need Acrobat Reader 4.0+ to view and print the files. Acrobat Reader with Search is recommended and is available as a free download from http://www.adobe.com. If you do not use Acrobat Reader with Search, you can only view and print files; you cannot search across the collection of files.

To copy the documentation to a client or server:

1. Copy the fortedoc directory and its contents from the CD-ROM to the client or server hard disk.

   You can specify any convenient location for the fortedoc directory; the location is not dependent on the Forte distribution.

2. Set up a directory structure that keeps the fortedoc.pdf and the 4gl directory in the same relative location.

   The directory structure must be preserved to use the Acrobat search feature.

Note To uninstall the documentation, delete the fortedoc directory.

To view and search the documentation:

1. Open the file fortedoc.pdf, located in the fortedoc directory.
2. Click the **Search** button at the bottom of the page or select **Edit > Search > Query**.
3. Enter the word or text string you are looking for in the Find Results Containing Text field of the Adobe Acrobat Search dialog box, and click **Search**.

   A Search Results window displays the documents that contain the desired text. If more than one document from the collection contains the desired text, they are ranked for relevancy.

Note For details on how to expand or limit a search query using wild-card characters and operators, see the Adobe Acrobat Help.

4. Click the document title with the highest relevance (usually the first one in the list or with a solid-filled icon) to display the document.

   All occurrences of the word or phrase on a page are highlighted.

5. Click the buttons on the Acrobat Reader toolbar or use shortcut keys to navigate through the search results, as shown in the following table:

Toolbar Button	Keyboard Command
Next Highlight	Ctrl+]
Previous Highlight	Ctrl+[
Next Document	Ctrl+Shift+]

6. To return to the fortedoc.pdf file, click the Homepage bookmark at the top of the bookmarks list.
7. To revisit the query results, click the **Results** button at the bottom of the fortedoc.pdf home page or select **Edit > Search > Results**.

Chapter 1

Using Fscript

This chapter explains how you can use the Fscript utility, including:

- starting and stopping Fscript
- working with parts of applications under development
- working with development repositories
- configuring and making application distributions

This chapter assumes that you are familiar with the standard Forte functions provided by the Forte Workshops.

About Fscript

The Fscript utility lets you use a command-line interface to perform the same tasks you can perform in the Forte workshops. Because of its simple interface, Fscript is available on all clients and servers that are running in the Forte installation.

Scripts

You can easily write *scripts*, files containing executable Fscript commands, that let you automate many Forte tasks, such as testing your application or backing up your repositories. Your scripts can be portable across the different platforms supported by Forte, and you can include any Fscript commands in your scripts.

Fscript provides a set of commands for performing the same tasks available in the Forte Workshops. In Fscript, you can:

- create new projects
- examine project components
- define and modify classes
- test or run application starting with the main projects
- partition and run distributed applications
- define and deploy libraries

This chapter contains general guidelines for working with Fscript commands.

Starting Fscript

You can start the Fscript utility on any node in your Forte environment.

To start the Fscript utility on Windows NT platforms:

1 Double-click the Fscript Distributed or Fscript Standalone icon.

To start the Fscript utility on UNIX, OpenVMS, or Windows NT platforms:

1 Type the **fscript** command, as described below.

When the Fscript utility starts, it gives you an "fscript>" prompt.

fscript Command

To start Fscript, enter the following command:

Portable syntax

fscript [**-fs**] [**-fr** *repository_name*] [**-fw** *workspace_name*] [**-fcons**]
*[**-fns** name_server_address]* [**-fnd** *node_name*] [**-fmn** *model_node_name*]
[**-fm** *memory_flags*] [**-fl** *logger_flags*] [**-i** *input_file*]
[**-o** *output_file*]

OpenVMS syntax

VFORTE FSCRIPT
[/**STANDALONE**]
[/**REPOSITORY**=*repository_name*]
[/**WORKSPACE**=*workspace_name*]
*[/**NAMESERVER**=name_server_address]*
[/**FCONS**]
[/**NODE**=*node_name*]
[/**MODEL_NODE**=*model_node_name*]
[/**MEMORY**=*memory_flags*]
[/**LOGGER**=*logger_flags*]
[/**INPUT**=*input_file*]
[/**OUTPUT**=*output_file*]

The following table shows the command line flags for the **fscript** command.

Flag	Description
-fs /STANDALONE	Open Fscript as a stand-alone facility, without connecting to an Environment Manager. If you start Fscript with this flag, you cannot test distributed applications using RunDistrib and you cannot use a central repository as the current repository. To omit this flag, you must define a valid FORTE_NS_ADDRESS or **-fns** flag and be connected to the network.
-fr *repository_name* **/REPOSITORY**=*repository_name*	Specifies the repository to use for Fscript. See "Specifying a Repository" on page 18 for information about specifying the repository to use with Fscript.
-fw *workspace_name* **/WORKSPACE**=*workspace_name*	Specifies the workspace name to open for Fscript. See "Specifying a Workspace" on page 19 for information about specifying the workspace to use with Fscript.
-fns *name_server_address* **/NAMESERVER=** *name_server_address*	Specifies the name service address for the environment in which this application will run. This value overrides the value, if any, specified by the FORTE_NS_ADDRESS environment variable. If you want your application to be able to switch to a backup Environment Manager if the primary Environment Manager fails, you can also specify multiple name service addresses, as discussed in *Forte 4GL System Management Guide*.

Flag	Description
-fcons /FCONS	On UNIX and VMS, the **-fcons** flag on the fscript command ensures that Fscript will run even if the connection to X cannot be made. Without the **-fcons** flag, Fscript fails if the connection to X fails.
-fnd *node_name* **/NODE_NAME**=*node_name*	(Clients only) Specifies the node name that identifies the specific client node that is running Fscript. This is only valid on Windows.
-fmn *model_node_name* **/MODEL_NODE**=*model_name*	Specifies the name of the model node for the client that is running Fscript. See "Specifying Model Node Names" on page 19 for more information about model node name.
-fm *memory_flags* **/MEMORY**=*memory_flags*	Specifies the memory manager settings to use for this Fscript session. See Appendix B, "Memory and Logger Flags" for information.
-fl *logger_flags* **/LOGGER**=*logger_flags*	Specifies the starting log tracing flags for this Fscript session. See Appendix B, "Memory and Logger Flags" for information
-i *input_file* **/INPUT**=*input_file*	Specifies an alternate input file. The default is standard input.
-o *output_file* **/OUTPUT**=*output_file*	Specifies an alternate output file. The default is standard output.

The following examples demonstrate the use of commands to start Fscript:

- Start Fscript in standalone mode, with a local repository named Myrepos and a workspace named jimmy:

```
fscript -fs -fr bt:Myrepos -fw jimmy
```

- Start Fscript in distributed mode, with a workspace named Sammy, and memory flags setting minimum memory to 2000 and maximum memory to 4000.

```
fscript -fw Sammy -fm "(n:2000,x:4000)"
```

- Start Fscript in standalone mode with logger flags set to log all trace and Environment Manager messages to standard output:

```
fscript -fs -fl "%stdout(trc:* em:*)"
```

Specifying a Repository

-fr flag

If you do not specify the **-fr** flag, you must use the **SetRepos** command in **Fscript** before performing an **Open** command. If you specify neither the **-fr** flag or the **SetRepos** command, **Fscript** will connect to the central development shared repository.

You can either specify the name of a central repository server, a shadow repository, or a local repository.

To use a central repository, specify the name of the central repository server, as shown in the following example:

```
fscript -fr CentralRepository -fw MyWorkspace
```

To use a shadow repository or a local repository, you need to specify the letters "bt:" before the path and name of the repository, as shown in the following example:

```
fscript -fr bt:c:\repos\LocalRepository -fw MyWorkspace
```

Specifying a Workspace

-fw flag

If you do not specify the **-fw** flag, you must use the **Fscript SetWorkspace** command as your first command. If you are using a private repository, you can use the workspace name FirstWorkspace.

After you start Fscript, you must use the **Open** command to actually open your workspace.

Specifying Model Node Names

-fmn flags

On Windows, you can specify **-fmn** flags when you start an Fscript session. If the **-fmn** flag is specified (or if the FORTE_MODELNODE environment variable is set in the Windows forte.ini file), the client is assumed to be a model node, and the model node name will be looked up in the environment repository to identify information about the node for use in partitioning and running applications. If the **fscript** command does not specify the **-fmn** flag, then the node information associated with the node is read from the environment repository.

Running A Script File

-i flag

You can run an Fscript script file as soon as you start Fscript by specifying the name of the script on the **-i** flag of the **fscript** command.

For example, you might want to define an Fscript script in a file called Backup.scr that backs up a repository, then create an icon for it on your Windows desktop. You could define an icon to invoke the following command:

```
fscript -i Backup.scr
```

If you want Fscript to start, run your script, then automatically exit Fscript, you must include a **Quit** or **Exit** command as the last command in the script.

Exiting Fscript

To leave Fscript, enter the **Quit** or **Exit** command in the Fscript window. For information about the **Quit** command, see “Quit” on page 74. For information about the **Exit** command, see “Exit” on page 49.

Working with Objects in Fscript

Current objects

In Fscript, different commands work with different objects. A *current* object is the last object that was set or found using an Fscript **Set…** or **Find…** command for a certain type of object. Many Fscript commands operate a current object. For example, the **ShowPlan** command works with the current plan, and the **SetPartRepCount** command works with the current partition. You use the various **Set…** and **Find…** commands to designate the current setting for each object type, and use the various **Show…** commands to list the properties of those objects.

The following table lists the objects that you can use as current objects for Fscript commands:

Object	Description	Command to Set Current Object
application	An application is the *current application* when the current project is this application's *main project*. The main project is the project that contains the code that starts the application.	FindPlan
application component	An *application component* can be a partition or a library project.	FindAppComp
configuration	The *current configuration* is either an application configuration or a library configuration. A *library configuration* is the assignment of projects that will be installed as libraries on nodes of the current environment. An *application configuration* is the assignment of partitions for the current application on nodes of the current environment.	FindPlan FindEnv or FindActEnv Partition
environment	An *environment* is the definition for the distributed system on which you run a distributed Forte application.	FindEnv or FindActEnv
library project	A *library project* is a project that is assigned to a node in a library configuration. This project will be installed as a library.	FindAppComp
partition	A *partition* is a portion of the application that runs on a client or a server, which is assigned to a node in the environment.	FindAppComp
plan	A *plan* can be a project, a business model, or an application model.	FindPlan
project	A *project* is the definition for an application or shared library.	FindPlan
repository	A *repository* is a database that stores the plans you create with Forte.	SetRepos
workspace	A *workspace* is your view of the repository, where you can work on code stored in the repository.	SetWorkspace

Using File Names in Fscript Commands

A number of Fscript commands use file names as arguments. Fscript has several features that are useful for setting these file names.

Working with Directories

Fscript provides two commands, **SetPath** and **AddPath,** which let you set a path of directories in which to search for referenced files. These commands let you keep sets of commonly used **Fscript** command scripts in convenient directory structures so you do not have to constantly re-specify path names. The **ShowPath** command lets you show the current search path, while the **WhichFile** command searches through the directories in the current directory search path to locate the first directory in which the specified file exists.

File Naming Conventions

You can use either local file naming or Forte's portable name format to specify directory and file names.

Local file naming

Local file naming means that you can specify file names using the format native to the system on which you invoke an Fscript command. Local file naming is the default, but you can explicitly select it using the **UseLocal** command. The following example shows how you could specify a file on different platforms:

Operating System	Specifying the working.pex file
UNIX	/dev/mammoth/working.pex
VMS	dev:[mammoth]working.pex
Windows	c:\dev\mammoth\working.pex

Portable file naming

You can also have Fscript interpret all file names as *portable* name format by invoking the **UsePortable** command. The portable format uses a UNIX-style syntax to specify directory paths. Also, you should keep file name lengths down to an 8 character header, an optional dot (.), and a maximum of a 3 character trailer to ensure portability across all systems. You should not use case as a means of differentiating file names.

Referencing Directory Paths using Environment Variables

You can use environment variables to specify directory paths in Fscript commands.

Local file naming

If you define your environment variables using local file naming, you can use the $ expansion option to reference directory paths in your Fscript commands. Any time you need to reference a directory path contained in a Forte environment variable, you can type the name of the environment variable, surrounded by braces, and preceded by a dollar sign.

For example, suppose you have defined PEXDIR as an environment variable containing a directory path using the appropriate local file naming syntax shown above. You can invoke the **ImportPlan** command to import the contents of the working.pex file using the following syntax for Windows:

```
fscript> ImportPlan ${PEXDIR}\working.pex
```

Portable file names

When using portable file names, you can use the special character % in place of the $ for environment variable expansion. This expands the environment variable name that was specified in the braces. And it also takes the directory or file name that has been specified in local operating system format and converts it to a valid portable name.

For example, although the PEXDIR environment variable is defined in local operating system format, you can still invoke the following commands on any system:

```
fscript> UsePortable
fscript> ImportPlan %{PEXDIR}/working.pex
```

Fscript Commands Listed By Task

This section contains lists of Fscript commands, organized by tasks you can perform using Fscript. These tables contain a list of Fscript commands associated with each task; not all commands listed with the task are required to complete the task.

Working with Repositories

The following table contains commands you can use to work with a Forte repository.

	Fscript commands	Description	See...
Setting the current repository and workspace	SetRepos	Set the name of the current repository for the next **Open** command.	page 90
	SetWorkspace	Set the name of the current workspace for the next **Open** command.	page 91
Opening the current workspace in the current repository	Open	Open the workspace specified on the last **SetWorkspace** command.	page 72
Getting information about the current repository	ShowReposInfo	Show physical information about the current repository.	page 96
	ListWorkspaces	List the workspaces in the current repository.	page 65
	ShowCompHistory	Show the integration history of a component, or set of matching components in the current repository.	page 93
	ShowIntegrations	Show information about integrations of workspaces in the current repository.	page 94
Working with detached shadow repositories	DetachFromCentral	Detach a shadow repository from the current central repository, reserving the current workspace.	page 45
	AttachToCentral	Attach the current shadow repository to the central repository.	page 37
Backing up the repository	BackupRepos	Backup the current repository to another directory.	page 38
Managing workspaces in this repository	NewWorkspace	Create a new workspace in the current repository.	page 71
	RemoveWorkspace	Remove a workspace from the current repository.	page 77
	ShowLockedWorkspaces	Show a list of workspaces that have locks on them in the current repository, and show the type of lock.	page 95
	UnlockWorkspace	Unlock a workspace in the current repository.	page 101
	ForceWorkspaceUnreserved	Forces a workspace to become unreserved.	page 54
	ExportWorkspace	Writes the definition of all plans in the current workspace to a text file.	page 52
	ShowIntegrations	Show information about the integrations of workspaces in the current repository.	page 94
Working with publicly-available plans	ListPublicPlans	List the publicly-available plans in the current repository.	page 63
	RemovePublicPlan	Remove a publicly-available plan completely from the current repository.	page 76

Working with the Workspaces

The following table contains commands you can use to work with a Forte workspace.

	Fscript commands	Description	See...
Setting the current workspace	SetRepos	Set the name of the current repository for the next **Open** command.	page 90
	SetWorkspace	Set the name of the current workspace for the next **Open** command.	page 91
Opening the current workspace	Open	Open the workspace specified on the last **SetWorkspace** command.	page 72
Setting the workspace password	SetPassword	Set the password for the current workspace.	page 86
Getting information about the current workspace	ShowWorkspace	Show the name of the current workspace and repository.	page 96
Listing the contents of the current workspace	ListEnvs	List the environments in the current workspace.	page 61
	ListPlans	List the plans in the current workspace.	page 62
Saving changes made to the current workspace	Save	Save changes in the current workspace to the shadow repository.	page 80
	Commit	Commit changes in the current workspace to the central repository.	page 42
Compiling all plans in a workspace	CompileWorkspace	Compiles all plans in the current workspace.	page 44
Importing a workspace	ImportWorkspace	Import one or more plans from a file and merge the changes into the plans in a workspace.	page 57
Integrating changes into the system baseline	UpdateWorkspace	Update the current workspace with any changes in the system baseline since the last update.	page 102
	IntegrateWorkspace	Integrate the changes in the current workspace into the shared repository.	page 59
Closing the current workspace	Close	Close the current open workspace.	page 41
Defining a new workspace	NewWorkspace	Create a new workspace in the current repository.	page 71
Managing workspaces in the current repository	ListWorkspaces	List the workspaces in the current repository.	page 65
	RemoveWorkspace	Remove a workspace from the current repository.	page 77
	ShowLockedWorkspaces	Show a list of workspaces that have locks on them in the current repository, and show the type of lock.	page 95
	UnlockWorkspace	Unlock a workspace in the current repository.	page 101
	ForceWorkspaceUnreserved	Forces a workspace to become unreserved.	page 54
	ExportWorkspace	Writes the definition of all plans in the current workspace to a text file.	page 52
	ShowIntegrations	Show information about the integrations of workspaces in the current repository.	page 94
Adding a plan from the repository	ListPublicPlans	List the publicly-available plans in the current repository.	page 63
	IncludePublicPlan	Include a publicly-available plan in the current workspace.	page 58
Removing a plan from the current workspace	ExcludePlan	Exclude a plan from the current workspace, which removes it completely if it has never been integrated.	page 48

Working with Any Plans

The following table contains commands you can use to work with any kind of Forte plan, including a project, an application model, or a business model.

	Fscript command	Description	See...
Getting a list of plans in the current workspace	ListPlans	List the plans in the current workspace.	page 62
Working with plans	NewPlan	Create a new plan in the current workspace.	page 70
	FindPlan	Set the current plan.	page 53
	ShowPlan	Display information about the current plan.	page 95
	ExportPlan	Export all components in a plan to a file.	page 50
	ImportPlan	Import the plan from a file and merge the changes into a plan.	page 56
	CompilePlan	Compile all out-of-date components in a plan.	page 43
	AddSupplierPlan	Include a plan as a supplier plan to the current plan.	page 36
	RemoveSupplierPlan	Remove a supplier plan from the current plan.	page 76
	ShowPlanHistory	Print information about past integrations of the current plan.	page 96

Working with Projects and Project Components

To define and modify projects in **Fscript**, you provide project definition information in one or more text files, then import each text file. Importing the file compiles the project definition information as a project in the repository.

For the full syntax of the statements that you can include in project definition files, see the *TOOL Reference Manual*. For information about compiling the project definition information using the Fscript **CompilePlan** command, see "CompilePlan" on page 43.

The following table contains commands you can use to work with Forte projects, and project components.

	Fscript command	Description	See...
Working with projects	NewProj	Create a new project in the current workspace.	page 70
	NewPlan	Create a new plan in the current workspace.	page 70
	AddSupplierPlan	Include a project as a supplier project to the current project.	page 36
	RemoveSupplierPlan	Remove a supplier project from the current project.	page 76
	IncreaseCompatLevel	Increase the compatibility level for a project.	page 59
	RevertProj	Undo all changes made to a project since last integration.	page 78

	Fscript command	Description	See...
Working with classes, constants, and attributes of the current project	ListComps	List components in the workspace with branch and checkout status.	page 61
	BranchComp	Branch a read-only or checked out component in the workspace.	page 38
	BranchAllComps	Branch all components in the current project to the workspace.	page 38
	UndoBranchComp	Undo the branch for a component, and revert to previous state.	page 99
	CheckoutComp	Check out a read-only or branched component in the workspace.	page 40
	CheckoutAllComps	Check out all components in the current project to the workspace.	page 39
	UndoCheckoutComp	Undo the checkout for a component, and revert to previous state.	page 100
	Compile	Import and compile a file containing project component definitions.	page 42
	ImportClass	Import and compile a file containing project component definitions.	page 55
	RemoveComp	Remove the named component from the current project.	page 75
	UndoRemoveComp	Undo the effects of a **RemoveComp** on a component and revert to previous state.	page 101
	RenameComp	Rename a component in the current project.	page 77
	ExportClass	Export the named class to a text file.	page 50
	ExportWindowClass	Export a UserWindow subclass to two files, one for the class and one for the window definition.	page 51
	ShowCompHistory	Show the integration history of a component, or set of matching components.	page 93

Working with Express Components

The following table contains commands you can use to work with Forte application models and business models. Other commands that can be used with application models and business models are listed in "Working with Any Plans" on page 25.

For general information about working with Express components, see *A Guide to Forte Express*.

	Fscript command	Description	See...
Working with application and business models	BranchPlan	Branch the current business model or application model.	page 39
	UndoBranchPlan	Undo the branch for the current business model or application model, and revert to its previous state.	page 99
	CheckoutPlan	Check out the current business model or application model.	page 40
	UndoCheckoutPlan	Undo the checkout for the current business model or application model, and revert to its previous state.	page 100
Generating code from a business or application model	CompilePlan	Generate application code from the business or application model that is the current plan	page 43
Checking model for errors	ValidatePlan	Check the business model or application model for errors without generating application code.	page 104
Export an Express template file	ExportTemplate	Export the current project as a Forte Express template file.	page 51

Partitioning, Testing, and Distributing Applications

The following table contains commands you can use to partition, test, and distribute an application.

	Fscript commands	Description	See...
Setting the main project of the application as the current project	FindPlan	Set the current plan.	page 53
Setting the current environment	FindEnv	Set the current environment.	page 53
	FindActEnv	Set the current environment to the active environment.	page 52
Getting information about the current environment	ShowEnv	Show information about the current environment.	page 93
Partitioning the application	SetPrefNode	Set the preferred server node for subsequent partitioning of an application in an environment.	page 88
	SetProjType	Set project as user application (1), server only (2), or library (3).	page 89
	SetProjRestricted	Set the restricted property for a project.	page 88
	SetServiceEOSInfo	Define what external object service a service object is available to and its export name.	page 91
	Partition	Partition using the current project and the current environment.	page 73
Setting the current configuration	FindPlan	Set the current plan.	page 53
	FindActEnv	Set the current environment to the active environment.	page 52
	FindEnv	Set the current environment.	page 53
Getting configuration information for applications	ShowApp	Show information about the current application configuration.	page 92
	FindPlan	Set the current plan.	page 53
	ShowPlan	Display information about the current plan.	page 95
Deleting an application configuration	RemoveConf	Remove a configuration for the current project.	page 75
Modifying a configuration	NewPart	Create a new partition, with the specified service object.	page 69
	FindAppComp	Set the current component for the current configuration.	page 53
	AssignAppComp	Assign a partition or a library to a specific node.	page 36
	UnassignAppComp	Remove the assignment of a partition or library on a node.	page 99
	MoveServiceToPart	Move a service object in the current application configuration to another partition.	page 68
	SetPartRepCount	Set the autostart replication count for a load balanced or failover server on a node.	page 85
	SetAppletFlag	Defines the client application associated with a logical client partition as an applet.	page 82
Setting up reference partitions	ListServiceApps	List installed applications containing the same referenced service object.	page 63
	UseServiceFromApp	Reference the named service object from the named application.	page 104
Setting up a server partition to autostart	EnableAppComp	Enable autostart for a partition on a specific node.	page 47
	SetPartArgs	Set the startup arguments for a partition on a specific node.	page 84
	DisableAppComp	Disable autostart for a partition on a specific node.	page 46
Defining a partition as compiled	SetAppCompCompiled	Define a partition or library as compiled on a specific node.	page 82

	Fscript commands	Description	See...
Testing an application	SetProjStart	Set the starting class and method for the current project.	page 88
	Run	Run the application in test mode from the starting class and method.	page 78
	RunDistrib	Run the application in distributed mode for the current configuration.	page 79
	RunFile	Run the fragment of TOOL code that is in the file.	page 80
	StopRemoteParts	Stop remote partitions which have been launched by the **RunDistrib** command.	page 97
Testing client partitions	ListTestApps	List applications which are currently being partitioned for use in shared client testing.	page 64
	TestApp	Start a wait for testing the client portion of a remotely started application.	page 98
Making an application distribution	MakeAppDistrib	Create a distribution for the current configuration.	page 65

Configuration Locks

In Fscript, when you partition an application or library project or modify a configuration, Forte registers the configuration lock for the environment, which indicates that someone is configuring an application or library project in this environment. This lock does two things:

- It allows concurrent users of the **TestClient** utility or the **TestApp** command in Fscript to participate in concurrent testing of the application under configuration.
- It prevents any concurrent users of the Environment Console or Escript from making changes to the environment.

Registering a lock

The following commands register configuration locks:

Fscript command	See:	Fscript command	See:
AssignAppComp	page 36	RunDistrib	page 79
DisableAppComp	page 46	SetPartArgs	page 84
EnableAppComp	page 47	SetPartRepCount	page 85
FindAppComp	page 53	SetPrefNode	page 88
MakeAppDistrib	page 65	ShowApp	page 92
MoveServiceToPart	page 68	UnassignAppComp	page 99
NewPart	page 69	UseServiceFromApp	page 104
Partition	page 73		

The configuration lock for the current environment and application remains in place until you execute a command that indicates you are no longer interested in this configuration.

Releasing a lock

The commands that release the configuration lock are:

Fscript command	See:	Fscript command	See:
Close	page 41	FindPlan	page 53
Exit	page 49	Quit	page 74
FindActEnv	page 52	StopRemoteParts	page 97
FindEnv	page 53		

Building Libraries

The following table contains commands you can use to build libraries.

	Fscript commands	Description	See...
Setting the library project as current project	FindPlan	Set the current plan.	page 53
Setting the current environment	FindEnv	Set the current environment.	page 53
	FindActEnv	Set the current environment to the active environment.	page 52
Getting information about the current environment	ShowEnv	Show information about the current environment.	page 93
Creating a new library	SetPrefNode	Set the preferred server node for subsequent partitioning of a library in an environment.	page 88
	SetProjType	Set the project as user application (1), server only (2), or library (3).	page 89
	SetProjRestricted	Set the restricted property for a project.	page 88
	Partition	Partition using the current project and the current environment.	page 73
Modifying libraries	AddProjToLib	Add a project to the current library configuration.	page 35
	RemoveProjFromLib	Remove a project from the current library configuration.	page 76
	FindAppComp	Set the current component for the current configuration.	page 53
	AssignAppComp	Assign a partition or a library to a specific node.	page 36
	UnassignAppComp	Remove the assignment of a partition or library on a node.	page 99
Deleting a library configuration	RemoveConf	Remove a configuration for the current project.	page 75
Defining a library as compiled	SetAppCompCompiled	Define a partition or library as compiled on a specific node.	page 82
Making a library distribution	MakeAppDistrib	Create a distribution for the current configuration.	page 65

Writing Scripts

You can easily create and maintain scripts of Fscript commands for background or batch execution. Also, because of its simple interface, Fscript is available on all clients and servers that are running in the Forte installation, so your scripts can be portable across the different platforms supported by Forte. You can include any Fscript commands in your scripts.

Comments

To include comments in your scripts, start the line containing the comment with the # character, as shown in the following example:

```
# Find the current active environment.
FindActEnv
```

The following table contains commands you can use within scripts, in addition to the other Fscript commands.

	Fscript commands	Description	See...
Recording manually entered commands	Script	Capture Fscript commands into a file.	page 81
Invoking OS commands	ExecCmd	Execute an operating system command.	page 48
	Shell	Starts a session in which operating system commands can be entered.	page 92
Running a script as part of another	Include	Execute a set of Fscript commands from another file.	page 58
Controlling command execution	Repeat	Repeats the next command the specified number of times.	page 78
	Step	Step through commands in a file started with the **Include** command.	page 97
	Delay	Delay the current task for the specified number of milliseconds.	page 45
Leaving Fscript if no environment manager is running	ExitIfNoEnvMgr	Leave Fscript as soon as no environment manager is running.	page 49
Controlling error printing	CommentOff	Turn off recording of comments to standard output.	page 42
	CommentOn	Turn on recording of comments to standard output.	page 42
	SilentOff	Turn off the printing of exceptions to standard output.	page 97
	SilentOn	Turn on the printing of exceptions to standard input.	page 97
	ShowExpansions	Enable or disable the printing of alias expansions to standard output when they occur.	page 93

Working with Fscript

The following table contains commands you can use to work with Fscript itself.

	Fscript commands	Description	See...
Getting information about Fscript commands	Help	List information about using Fscript commands.	page 54
Leaving Fscript	Exit	Leave Fscript.	page 49
	Quit	Leave Fscript.	page 74
Setting a return value	ExitStatus	Set a value to be returned to the routine that started Fscript.	page 49
Changing Fscript settings	ModLogger	Add or remove trace flags.	page 68
	SetOutFile	Set the file where output is printed.	page 84
	UseLocal	Set Fscript to recognize file name input in local name format.	page 102
	UsePortable	Set Fscript to recognize file name input in portable format.	page 103
Working with aliases	AddAlias	Define an alias for an Fscript command and its arguments.	page 34
	ShowAlias	Display one or all defined aliases with their expansions.	page 92
	RemoveAlias	Remove an alias.	page 74

Working with Files and the Operating System

The following table contains commands you can use to work with the operating system and files stored in the operating system.

	Fscript commands	Description	See...
Specifying the file name format to use	UseLocal	Set Fscript to recognize file name input in local name format.	page 102
	UsePortable	Set Fscript to recognize file name input in portable format.	page 103
Listing the files in a directory	Directory	List files in a directory.	page 46
	ListFiles	List files in a directory.	page 62
	Ls	List files in a directory.	page 65
Creating a directory	MkDir	Create a directory in the operating system.	page 68
Working with the contents of a file	SearchFile	Locate text in a file.	page 81
	ListFile	List the contents of a file in output.	page 62
	Vi	Start an editor so that you can edit the specified file.	page 104
Writing to a file	ReadIntoFile	Read lines of data following this command and write the data to the specified file.	page 74
Changing the permissions of a file	Chmod	Change the permissions of the specified file.	page 41
Copying a file	CopyFile	Copy a file.	page 44
	Cp	Copy a file.	page 44
	Duplicate	Copy a file.	page 47
Renaming a file	Mv	Rename a file.	page 69
Deleting a file	RemoveFile	Remove the specified file.	page 75
	Rm	Remove the specified file.	page 78
Setting the current working directory	SetDefault	Change the current working directory to the specified directory.	page 83
	Cd	Change the current working directory to the specified directory.	page 39
Getting the current directory	Pwd	Print the current working directory.	page 74
Defining the Fscript command search path	SetPath	Set the directory path for Fscript commands to seek files.	page 87
	SetSearchPath	Set the directory path for Fscript commands to seek files.	page 90
	AddPath	Add a new directory path to the directory path Fscript commands use to seek files. (See also **SetPath**).	page 34
Displaying the current search path	ShowPath	Shows the directory paths Fscript commands use to seek files. (See also **SetPath**).	page 95
Locating a file in the search path	WhichFile	Find a file in the current directory search path.	page 105
Invoking OS commands	ExecCmd	Execute an operating system command.	page 48
	Shell	Starts a session in which operating system commands can be invoked.	page 92
Working with environment variables	SetEnv	Set an environment variable to the specified value.	page 83
	PrintEnv	Print the current value of an environment variable.	page 73

Managing the Development Environment

The following table contains commands you can use to manage your development environment.

	Fscript commands	Description	See...
Running the garbage collector	CollectMem	Run the memory garbage collector.	page 41
Getting memory statistics	MemStats	Print memory statistics to standard output.	page 66

Chapter 2

Fscript Commands

This chapter describes, in alphabetical order, the commands that let you start and quit Fscript, as well as the commands that let you work with:

- workspaces
- plans
- project components
- configurations, partitions, and libraries

This chapter also describes commands that:

- write scripts to automate tasks
- work with files in your operating system
- test distributed applications
- perform repository maintenance

This chapter assumes that you are familiar with the standard Forte functions provided by the Forte Workshops, as described in *A Guide to the Forte 4GL Workshops*.

AddAlias

Add a command alias

The **AddAlias** command adds an alias expansion mapping *alias_name* to *command_string*. An *alias* is a synonym you can define for an Fscript command and its arguments.

Syntax **AddAlias** *alias_name command_string*

Argument	Description
alias_name	A string without blanks that represents the name of the alias.
command_string	A string containing a command and its arguments, if arguments are required for the command. If this string contains blanks, surround the string with double quotation marks.

Aliases exists only for the duration of the current Fscript session.

The following example shows how you can create an alias using the AddAlias command:

```
AddAlias FProjs “ListPublicPlans f*”
```

This example creates an alias called FProjs that invokes the command “ListPublicPlans f*”.

AddPath

Add directories to directory search path

The **AddPath** command adds the specified directories to the current search path used by any of the commands that take a file name as an argument.

Syntax **AddPath** *directory-name*[;*directory-name...*]

Argument	Description
directory-name	The name of a directory in which to look for files that are specified without a path.

Most of the commands that have input file arguments allow you to specify the name of the input file, without a full directory specification. In this case, they use the directory search path, as defined by the **SetPath** and **AddPath** commands, to find the file. The directory search path provides a set of directories that are checked in turn, until a file matching the unexpanded name is found.

The **AddPath** command is used to add one or more directories to the end of the current list of directories in the search path. You can use the **SetPath** command to reset the entire directory search path list.

By default, the directory search path only includes the current working directory. The current working directory is always considered the last directory in the directory search path, even after you give the **SetPath** command. If you want the current working directory to be searched first, you can specify the current working directory first in the list of directories for the **SetPath** command.

Specify each *directory_name* as a full directory path name. By default, directories should be specified in the local operating system directory format. However, if you have previously invoked the **UsePortable** command, the *directory_name* should be specified in Forte portable format, a UNIX-style directory format. To specify more than one directory, separate the directory names with semicolons.

You can embed environment variable names within the directory names, by using the following syntax:

${*environment_variable_name*}

The dollar sign and braces indicate that the name inside the braces is an environment variable, and the entire specification is replaced with the current setting of the environment variable.

You can also use the following syntax to expand the environment variable name and convert it to a portable file format as well:

%{*environment_variable_name*}

The percent sign and braces indicate that the name inside the braces is an environment variable, and the entire specification is replaced with the current setting of the environment variable. This is useful if you have invoked the **UsePortable** command, but have directories specified in environment variables in local format, which you need to convert to portable format.

Examples of the **AddPath** command are:

```
Fscript> AddPath /mydisk/mydir;${ENV_VAR}/subdir
Fscript> AddPath c:\mydir;${ENV_VAR}\subdir
Fscript> AddPath “Mac HD:Apps:TempFolder”;${ENV_VAR}:Sub
Fscript> AddPath $dka0:[path];${ENV_VAR}:[otherdisk.otherdir]
Fscript> UsePortable
Fscript> AddPath %{FORTE_ROOT}/install/examples
```

AddProjToLib

Add a project to a library configuration

The **AddProjToLib** command adds the specified project to the current library configuration.

Syntax **AddProjToLib** *project_name*

Argument	Description
project_name	The name of the project to include in the library configuration. This name must be the name of a project included in your workspace.

When you use the **AddProjToLib** command, the current configuration must be a library configuration; otherwise, this command will fail.

After you add projects to the library configuration, you need to repartition this library configuration to assign the project to nodes in the library configuration.

To assign an added project to nodes in the library configuration:

1. Repartition the library configuration using the **Partition** command. The added project is assigned to all the nodes in the library configuration where the other projects are assigned.
2. Use **UnassignAppComp** to remove all library projects from any node where you do *not* want the project installed as a library. When you remove a project from a node, all the projects in this library configuration are removed from the node.

However, when you add a restricted project to a library configuration using the **AddProjToLib** command, the project is assigned only to nodes where the required resources are installed. When you remove the project from the assigned node using the **UnassignAppComp** command, you remove just the restricted project.

Before you can add projects to a library configuration, you must have the library configuration as the current configuration. You can either use an existing library configuration, or you can create a new library configuration.

▶ **To use an existing library configuration:**

1. Use the **FindEnv** or **FindActEnv** command to set the current environment.
2. Use the **FindPlan** command to set the main project for the library configuration as the current project.

   The current configuration is now the library configuration for the current project and the current environment.

▶ **To create a new library configuration:**

1. Use the **FindPlan** command to set the main project for your library configuration as the current project.
2. Use the **SetProjType** command with the *type_flag* value of **3** to specify that the current project will be installed as a library.
3. Use the **FindEnv** or **FindActEnv** command to set the current environment.
4. Use the **Partition** command to generate the configuration for the new library configuration.

   You can now add or remove projects from the current library configuration using the **AssignAppComp** and **UnassignAppComp** commands.

AddSupplierPlan

Include supplier plans

The **AddSupplierPlan** command adds a supplier plan to the list of supplier plans for the current plan. A supplier plan can be a project, an application model, or a business model.

Syntax **AddSupplierPlan** *plan_name*

Argument	Description
plan_name	The name of the plan to include as a supplier plan. This name must be the name of a plan included in your workspace.

After you designate a supplier plan to a plan, you can reference any of the supplier plan components in your plan.

The order in which supplier plans are added to a TOOL plan does not matter. If there are name conflicts between components of several supplier plans (that is, two components have the same name), you will get compilation errors when you reference the conflicting component. You can use fully qualified TOOL names to specify exactly which plan component you want. See the *TOOL Reference Manual* for more information.

AssignAppComp

Assign an application component to a node

The **AssignAppComp** command assigns the specified application component—a partition or a library project—in the current configuration for eventual installation on the specified node.

Syntax **AssignAppComp** *node_name* [*component_name*]

Argument	Description
node_name	The name of the node where the component is to be assigned for installation.
component_name	The optional name of the application component which is to be assigned to the node.

The *node_name* must be a valid node defined in the environment. This node must have all of the external resource managers, communications protocols, and restricted projects needed to support the partition or library project.

The optional *component_name* is the name of the application component to be assigned to the node. An application component can be a partition or a library project. You can specify the unique trailer portion of the name only, such as “client”, to identify the application component. If no *component_name* is given, then the current application component is assigned. You can use the **FindAppComp** command to designate a current application component.

Use **AssignAppComp** to modify an application configuration

When an application is partitioned, each partition of the application is designated for future installation and execution on one or more nodes in the environment, based on matching the needed properties of the partition and the actual properties of each node. You can use the **AssignAppComp** command to designate additional nodes for future installation of a partition. When you use the **MakeAppDistrib** command for this configuration for the project, the distribution provides support for installing the partition on the newly designated node.

If a partition is assigned to more than one node and it is not a replicated partition, the partition will automatically start on only one of the assigned nodes. The **EnableAppComp** command designates which node is to used for automatically starting the partition, once installed. However, by assigning the partition for installation on more than one node, you can provide manual failover for a partition by starting other instances of the partition using Environment Console or Escript. For information about starting partitions manually, see *Forte 4GL System Management Guide*.

AttachToCentral

Reattach a shadow to the central repository

The **AttachToCentral** command attaches the currently-opened, detached workspace to the central repository.

Syntax AttachToCentral

Use the **AttachToCentral** command only if you are working in a shadow repository that has been detached from the central repository with the **DetachFromCentral** command. You must be running in distributed mode (no **-fs** flag) when you invoke the **AttachToCentral** command to reestablish communication with the central repository. You also must commit any outstanding changes to the detached shadow repository before giving the **AttachToCentral** command.

All changes made in the workspace since the workspace was detached are transferred to the central repository and committed. If these changes are significant, the **AttachToCentral** command can take some time to complete execution.

When you are reattached to the central repository, you can invoke any of the commands that are not allowed while the shadow repository is detached, such as **CheckOutComp** and **UpdateWorkspace**.

Because all changes in a workspace are saved in the central repository after an **AttachToCentral** command, you should invoke this command from time to time as a safe backup procedure, and then immediately reinvoke the **DetachFromCentral** command to run as a detached shadow repository.

BackupRepos

Back up a local or detached shadow repository

The **BackupRepos** command backs up the current local or detached shadow repository to another directory.

Syntax **BackupRepos** *directory_name*

Argument	Description
directory_name	The name of a directory where the backed-up copy of the repository is placed.

To use the **BackupRepos** command, the current repository must be a local repository or a detached shadow repository. Repositories are backed up as B-tree repositories.

By default, the *directory_name* should be specified in local operating system format. However, if you have previously invoked the **UsePortable** command, the *directory_name* should be specified in portable file name format. You can embed special syntax in the *directory_name* to have environment variable expansion performed on the specified name. See "SetPath" on page 87 for more details.

BranchAllComps

Branch all components in a project

The **BranchAllComps** command branches all read-only components in the current project.

Syntax BranchAllComps

The **BranchAllComps** command branches every component in the current project that is not already checked out or branched. Components that are checked out remain checked out. This command is very useful if you need to perform a command that might change any of the components in the project, such as **ImportPlan**.

For more information about branching, see "BranchComp" on page 38.

BranchComp

Branch a component in a project

The **BranchComp** command branches a read-only or checked-out component in the current project in the current workspace.

Syntax **BranchComp** *component_name*

Argument	Description
component_name	The name of a component in the current project to branch.

Branching a component gives you write access to a copy of the component, and you can modify this copy in the workspace. However, the changes to branched components cannot be integrated back into the system baseline in the repository. The **BranchComp** command is primarily used to test out changes while someone else has the component checked out.

To obtain the exclusive write lock on the component in the repository, you must use the **CheckOutComp** command.

You can use the **BranchComp** command on a component that you have already checked out to your workspace. However, when you do, your workspace no longer has the checkout lock, and the component is available for other workspaces to check out. Forte does not integrate the changes you made to the component when you had it checked out. These changes are kept in the branched copy of this component in your workspace.

You can invoke the **BranchComp** command from a shadow repository even if you are detached from the central repository.

BranchPlan

Branch a business model or application model

The **BranchPlan** command branches the current business model or application model in the current workspace. This command cannot branch projects; to branch a component of a project, use the **BranchComp** command.

Syntax BranchPlan

Branching a plan gives you write access to a copy of the plan, and you can modify this copy in the workspace. However, the changes to branched plans cannot be integrated back into the system baseline in the repository. The **BranchPlan** command is primarily used to test out changes while someone else has the plan checked out.

To obtain the exclusive write lock on the plan in the repository, you must use the **CheckOutPlan** command.

You can use the **BranchPlan** command on a plan that you have already checked out to your workspace. However, when you do, your workspace no longer has the checkout lock, and the plan is available for other workspaces to check out. Forte does not integrate the changes you made to the plan when you had it checked out. These changes are kept in the branched copy of this plan in your workspace.

You can invoke the **BranchPlan** command from a shadow repository even if you are detached from the central repository.

Cd

Change current directory

The **Cd** command calls the operating system to change the current working directory.

Syntax **Cd** *directory_name*

Argument	Description
directory_name	The name of a directory to make the new working directory.

By default, the *directory_name* should be specified in local operating system format. However, if you have previously invoked the **UsePortable** command, the *directory_name* should be specified in portable file name format. You can embed special syntax in the *directory_name* to have environment variable expansion performed on the specified name. See "SetPath" on page 87 for more details.

The **Cd** command and the **SetDefault** command are synonyms.

CheckoutAllComps

Check out all components in a project

The **CheckoutAllComps** command checks out all components in the current project; if any components are branched, the command converts branches to checkouts.

Syntax **CheckoutAllComps** [*available_flag*]

Argument	Description
available_flag	Specifies whether to require that all components in the project be available for check out before proceeding.

The *available_flag* argument can have one of two values:

Value	Description
0	The default. Check out any available components and ignores any components that cannot be checked out.
1	A check is made to see that all components in the project can be checked out before proceeding, and an error will be invoked if some components are not available

This command is useful if you want to ensure that your workspace has exclusive write locks on an entire project. For example, if you are importing project definition information using the **ImportPlan** command or the Project Workshop, you want to make sure that you have checked out all parts of the project that might be updated.

CheckoutComp

Check out one component

The **CheckoutComp** command checks out the specified component in the current project.

Syntax

CheckoutComp *component_name*

Argument	Description
component_name	The name of a component in the current project to check out.

Checkout lock

Checking out a component gives the current workspace an exclusive write lock, or *checkout lock*, for the component in the repository. After you check out a component, you can modify a copy of the component in your workspace, then integrate your changes.

You can invoke the **CheckoutComp** command from a shadow repository only if you are attached to the central repository. If you want to change a component in a detached shadow repository, you can use the **BranchComp** command.

If the specified component is already branched, the branch is converted to a checkout on the component.

Release a checkout lock

Because checkout locks are exclusive, no other workspace can check out the same component until the lock is removed. You can remove a checkout lock by integrating your changes, or by releasing the checked-out component.

To integrate changes to a checked-out component into the system baseline in the repository, use the **IntegrateWorkspace** command.

To release a checked-out component, you can use the **UndoCheckout**, **RevertProj** or **BranchComp** commands.

If you want to get temporary write access to a component without maintaining an exclusive lock on it, you can use the **BranchComp** command.

CheckoutPlan

Check out a business model or application model

The **CheckoutPlan** command checks out the current business model or application model. This command cannot check out projects; to check out a component of a project, use the **CheckoutComp** command.

Syntax

CheckoutPlan

Checking out a plan gives the current workspace an exclusive write lock, or *checkout lock*, for the plan in the repository. After you check out a plan, you can modify a copy of the plan in your workspace, then integrate your changes.

You can invoke the **CheckoutPlan** command from a shadow repository only if you are attached to the central repository. If you want to change a plan in a detached shadow repository, you can use the **BranchPlan** command.

If the current plan is already branched, the branch is converted to a checkout on the plan.

Release a checkout lock

Because checkout locks are exclusive, no other workspace can check out the same plan until the lock is removed. You can remove a checkout lock by integrating your changes, or by releasing the checked-out plan.

To integrate changes to a checked-out plan into the system baseline in the repository, use the **IntegrateWorkspace** command.

To release a checked-out plan, you can use the **UndoCheckoutPlan** or **BranchPlan** commands.

If you want to get temporary write access to a plan without maintaining an exclusive lock on it, you can use the **BranchPlan** command.

Chmod

Change the permissions of a file

The **Chmod** command changes the permissions of the specified file in the local file system for all user groups.

Syntax **Chmod** *mode file_name*

Argument	Description
mode	The permissions that you want to set for the file. *mode* can be **none**, **read**, **readwrite**, **execute**, **readexecute**, or **all**.
file_name	The name of the file on which you are changing the permissions.

mode defines the permissions you want to set for the file. The following table describes the possible values for *mode*:

***mode* value**	Description
none	Users cannot access this file.
read	Users can only read this file.
readwrite	Users can read and write into this file.
execute	Users can only execute this file.
readexecute	Users can read and execute this file.
all	Users can read, write, and execute this file.

Specify the *file_name* argument in local operating system format, by default. If you invoke the **UsePortable** command, however, the argument must be specified in portable file format.

Close

Close your workspace

The **Close** command closes the current open workspace.

If you have outstanding changes that have not been committed, the **Close** command prompts you to save the changes.

Syntax Close

You can only use the **Close** command on an open workspace. For information about opening a workspace, see "Open" on page 72.

To leave **Fscript**, you should invoke the **Exit** or **Quit** command.

CollectMem

Run memory reclamation

The **CollectMem** command runs the memory reclamation (garbage collection) utility on Fscript**.**

Syntax CollectMem

Normally, Forte automatically performs memory reclamation whenever memory is running low. You can use the **CollectMem** command to explicitly invoke the memory reclamation utility. For example, you might want to run the memory reclamation consistently at a certain time.

CommentOff

Stop including comments in output

The **CommentOff** command tells Forte to stop writing the comments in Fscript script files to standard output when they are processed by the **Include** command.

Syntax CommentOff

Use this command in scripts to stop Fscript from writing out the comments from script files to standard output. By default, these comments are not written to standard output. You can use the **CommentOn** and **CommentOff** commands to toggle this setting.

CommentOn

Include comments in output

The **CommentOn** command tells Forte to start writing the comments in Fscript script files to standard output when they are processed by the **Include** command.

Syntax CommentOn

Use this command in scripts to make Fscript write out the comments from script files to standard output. By default, comments are not written to standard output. You can use the **CommentOn** and **CommentOff** commands to toggle this setting.

Commit

Committing changes to your workspace

The **Commit** command writes to the repository all changes made in the current workspace since the last **Commit** command.

Syntax Commit

You should frequently invoke the **Commit** command to ensure that you do not lose work. If you invoke the **Quit** or **Close** command after you have made changes to the repository but not invoked the **Commit** command, Forte prompts you to save your work. If you choose not to save your work, Forte discards all changes since the last **Commit** command.

If you are working with an attached shadow repository, the **Commit** command saves all current changes to both the attached shadow repository and the central repository. If you want to save the current changes only to the attached shadow repository, use the **Save** command (see "Save" on page 80).

Compile

Compile TOOL statements

The **Compile** command reads a file containing definitions of project components and inserts the components into the current project.

Syntax **Compile** *file_name*

Argument	Description
file_name	The file containing commands that define components of a project.
	Note: The **Compile** command assumes a default file name extension of ".4gl".

The **Compile** command reads a file that contains TOOL statements that define one or more components or sub-components of a project and merges them into the current project. The **ImportClass** command and the **Compile** command are equivalent.

For information about importing and compiling an entire project, see "ImportPlan" on page 56.

Before you can invoke the **Compile** command, you must check out or branch any component that will be changed by the contents of the file. New components are automatically added and checked out to the workspace.

A definition file processed by the **Compile** command can include statements for creating classes, service objects, constants, interfaces, and cursors. For information about the syntax of the TOOL statements used in a project definition file, see the *TOOL Reference Manual*. The **Compile** command checks for compilation errors as it reads the file.

You can use the **Compile** command to add an exported class or to add classes that contain references to window definitions contained in **.fsw** files created by the Window Workshop. For information about adding classes that contain references to window definitions, see the description of the **has file** clause of the **class** statement in the *TOOL Reference Manual*.

Caution Any project component in a file processed by the **Compile** command completely replaces the current component definition. When you compile a new class or interface definition, the entire previous set of attributes, events, methods and method definitions are replaced for that class.

The following example illustrates a file that can be used with the **Compile** command. Both the class and a method are defined in the same file. You can add any number of interfaces, classes, methods, and other components to the same file.

```
class weather inherits from object
has public
  method setctemp(temp : integer):void;
  lowf : integer;
  virtual lowc : integer = (get = (5.0/9.0)*(lowf- 32),
              set = SetCTemp(lowc));
end class;

method weather.setctemp(temp : integer) : void
begin
  self.lowf = (9.0/5.0)*temp + 32;
end method;
```

CompilePlan

Compile out-of-date components

The **CompilePlan** command compiles the current plan. The current plan can be a project, an application model, or a business model.

Syntax **CompilePlan** [*force_flag*]

Argument	Description
force_flag	Flag indicating that Forte should compile all parts of the current plan, even if some or all of the components are up-to-date. **1** specifies that Forte should compile all of the components. **0** specifies that Forte should compile only the out-of-date components. The default value is **0**.

If the current plan is a project, the **CompilePlan** command compiles components of the project.

If the current plan is an application model or business model, the **CompilePlan** command generates TOOL code based on the model. For information about application models, business models, and the code generated using them, see *A Guide to Forte Express*.

The **CompilePlan** command automatically commits the changes it makes in the workspace to the repository.

CompileWorkspace

Compile all plans

The **CompileWorkspace** command compiles all plans in the current workspace.

Syntax **CompileWorkspace** [*projects_only*] [**force**]

Argument	Description
projects_only	A value of **0** (the default) compiles all plans in the current workspace. A value of **1** compiles only the projects in the workspace, not the other kinds of plans.
force	A value of **0** (the default) compiles only out-of-date objects in the current workspace. A value of **1** compiles all objects in the current workspace.

The **CompileWorkspace** command automatically commits the changes to the repository.

CopyFile

Copy a file

The **CopyFile** command copies a file or directory in the local file system.

Syntax **CopyFile** *file1_name file2_name* [**r**]

Argument	Description
file1_name	The name of the file to copy.
file2_name	The new file to create or overwrite.
r	Specifies to copy the directory and the entire subtree connected at that point, if *file1_name* is a directory.

Specify the *file1_name* and *file2_name* arguments relative to the current working directory. This is either the directory in which Fscript was started, or the last directory specified in the **Cd** command.

Specify the file name arguments in local operating system format, by default. If you have invoked the **UsePortable** command, specify the arguments in portable file format.

If an existing file has the same name as *file2_name*, it is overwritten.

The **CopyFile** command, **Cp** command, and **Duplicate** command are synonyms.

Cp

Copy a file

The **Cp** command copies a file or directory in the local file system.

Syntax **Cp** *file1_name file2_name* [**r**]

Argument	Description
file1_name	The name of the file to copy.
file2_name	The new file to create or overwrite.
r	Specifies to copy the directory and the entire subtree connected at that point, if *file1_name* is a directory.

Specify the *file1_name* and *file2_name* arguments relative to the current working directory. This is either the directory in which Fscript was started, or the last directory specified in the **Cd** command.

Specify the file name arguments in local operating system format, by default. If you have invoked the **UsePortable** command, specify the arguments in portable file format.

If an existing file has the same name as *file2_name*, it is overwritten.

The **CopyFile** command, **Cp** command, and **Duplicate** command are synonyms.

Delay

Delay the current task

The **Delay** command delays the current task for the specified number of milliseconds.

Syntax

Delay *milliseconds*

Argument	Description
milliseconds	The number of milliseconds that you want the current task to wait.

Use the **Delay** command in scripts to make the current task wait for the specified amount of time. For example, if your script invokes an operating system command using the **ExecCmd** command that starts a server, you might want to delay executing other commands in your script until after the server has probably started.

DetachFromCentral

Detach a shadow repository

The **DetachFromCentral** command detaches a shadow repository from the central repository.

Syntax

DetachFromCentral

You can only use the **DetachFromCentral** command if you are using a shadow repository that is currently attached to a central repository.

When you detach from the central repository, Forte ensures that the shadow repository contains all components in the current workspace, so the command might take a long time to complete.

The **DetachFromCentral** command commits changes in the current workspace to the central repository before detaching the shadow repository.

After Forte detaches the shadow repository, all changes to components in the current workspace are written to the shadow repository. To re-establish a connection to a central repository, invoke the **AttachToCentral** command.

Once detached, you cannot open any other workspaces in the detached shadow repository until you reattach to the central repository.

After you detach a shadow repository, you can open the repository in standalone mode. However, if you need distributed access for other operations, such as accessing a database, you still need to run in distributed mode.

Note

You cannot check out components while detached. However, any components that you check out before you invoke the **DetachFromCentral** command remain checked out. You can, however, branch additional components and create new projects and models.

After you invoke the **DetachFromCentral** command, other users of the central repository can access your detached workspace in read mode only. Only the detached shadow repository can modify the contents of the workspace. If the detached shadow repository becomes corrupted, use the **ForceWorkspaceUnreserved** command to release the lock on the workspace.

For information about creating a shadow repository, see *A Guide to the Forte 4GL Workshops* or the *Forte 4GL System Management Guide*.

Directory

Show files in a directory

The **Directory** command lists the files in the specified directory.

Syntax

Directory [*directory_name*]

Argument	Description
directory_name	The name of a directory. The default is the current working directory for Fscript.

This command lists the full names and permissions for all the files in the specified *directory_name*. If no *directory_name* is specified, the current working directory for Fscript is used. The current working directory is either the directory in which Fscript was started, or the last directory set by the **Cd** command.

Specify the *directory_name* in local operating system format unless you have invoked a **UsePortable** command for using portable file formats.

The **ListFiles** command, the **Ls** command, and the **Directory** command are synonyms.

DisableAppComp

Disable automatic startup for a partition of an application

The **DisableAppComp** command defines a partition on a node in the current application configuration as disabled.

Syntax

DisableAppComp *node_name* [*partition_name*]

Argument	Description
node_name	The node on which the partition is to be disabled.
partition_name	The optional name of the partition to be disabled.

The **DisableAppComp** command defines the specified partition on the specified node as disabled, which means that the partition does not automatically start when the application needs to use this partition. For example, this might be a replicated partition and another copy of this partition is enabled. Only the enabled copy of the partition starts automatically when the application needs this partition.

The *node_name* argument is the name of a node in the environment to which the partition has already been assigned, either as part of the default partitioning, or using the **AssignAppComp** command.

The optional *partition_name* argument is the name of the partition to be disabled on the node. You can specify the unique trailer portion of the name only, such as “client”, to identify the partition. If no *partition_name* is given, then the current partition is disabled. Use the **FindAppComp** command on the current configuration to designate a current partition.

A partition within a project can be assigned to one or more nodes within the environment. For non-replicated partitions that are to be placed on servers, only one of the assigned nodes can be designated the node on which the partition is to be automatically started and managed. This is done by the default partitioning, or by invoking the **EnableAppComp** command for the partition on that node. For replicated partitions, any number of assigned nodes can be enabled using the **EnableAppComp** command, and servers will be automatically started on all of the enabled nodes.

Use the **DisableAppComp** command to leave the node as an assigned node for the partition, but not to automatically start the server on that node when the system runs the application.

Duplicate

Copy a file

The **Duplicate** command copies a file in the local file system.

Syntax **Duplicate** *file1_name file2_name*

Argument	Description
file1_name	The name of the file to copy.
file2_name	The new file to create or overwrite.

Specify the *file1_name* and *file2_name* arguments relative to the current working directory. This is either the directory in which Fscript was started, or the last directory specified in the **Cd** command.

Specify the file name arguments in local operating system format, by default. If you have invoked the **UsePortable** command, specify the arguments in portable file format.

If an existing file has the same name as *file2_name*, it is overwritten.

The **CopyFile** command, **Cp** command, and **Duplicate** command are synonyms.

EnableAppComp

Enabling automatic startup for a server partition

The **EnableAppComp** command enables the specified partition to automatically start on the specified node, in the application started by the current project.

Syntax **EnableAppComp** *node_name* [*partition_name*]

Argument	Description
node_name	The node on which the component is be enabled.
partition_name	The optional name of a partition in the current configuration for which a node is to be enabled.

The **EnableAppComp** command specifies that a component, which has already been assigned to a node in the environment, be enabled to automatically start on that node. The partition then automatically starts when the application needs to use this partition. For example, this partition might be replicated, and another copy of this partition is disabled. Only the enabled copy of the partition starts automatically when the application needs this partition.

The *node_name* argument is the name of a node in the environment to which the component has already been assigned, either as part of the default partitioning, or through the **AssignAppComp** command.

The optional *partition_name* is the name of the partition to be enabled on the node. You can specify the unique trailer portion of the name only, such as “client”, to identify the partition. If no *partition_name* is given, then the current partition is enabled. Use the **FindAppComp** command on the current configuration to designate a current partition.

A partition within an application can be assigned to one or more nodes within the environment. For non-replicated partitions that are to be placed on servers, only one of the assigned nodes can be designated the node on which the partition is to be automatically started and managed. This is done by the default partitioning, or by invoking the **EnableAppComp** command for the partition on that node. For replicated partitions, any number of assigned nodes can be enabled using the **EnableAppComp** command, and servers will be automatically started on all of the enabled nodes.

Use the **DisableAppComp** command to leave the node as an assigned node for the partition, but not to automatically start the server on that node when the system runs the application.

ExcludePlan

Removing a plan from the workspace

The **ExcludePlan** command removes the specified plan from the current workspace. The plan can be a project, an application model, or a business model.

Syntax **ExcludePlan** *plan_name*

Argument	Description
plan_name	The name of the plan to remove. This name must be the name of a project, business model, or application model included in your workspace.

If the specified plan has been created in this workspace and never integrated into the system baseline, then an **ExcludePlan** command completely removes the plan from the repository. For information about removing an integrated plan from the repository, see "RemovePublicPlan" on page 76.

You *cannot* exclude a plan from your workspace if the plan is:

- included as a supplier plan in any project or model in your workspace
- a project that has components checked out
- a checked-out application model or business model

If you are using an attached shadow repository, all current changes are automatically written to the central repository.

ExecCmd

Run an OS command

The **ExecCmd** command invokes the specified operating system command.

Syntax **ExecCmd** *opsys_command* [*bg_flag*] [*in_file*] [*out_file*] [*err_file*]

Argument	Description
opsys_command	A valid operating system command appropriate to the system on which you are running Fscript.
bg_flag	Flag to specify whether the command runs synchronously or asynchronously. The default is synchronously (**0**). **1** specifies asynchronously.
in_file	An alternate input file for the operating system command.
out_file	An alternate output file for the operating system command.
err_file	An alternate error file for the operating system command.

This runs the command specified in the *opsys_command* argument. You can specify command line arguments to the command by enclosing the command in double quotes.

Each operating system command executes in its own command shell process. Therefore, to execute a series of related commands, you might want to write a script and invoke that script using the **ExecCmd** command.

The bg_flag can be set to 0 to indicate that the command is to be run synchronously until it completes, or 1 to indicate that the command is to be started up in the background (asynchronously). By default, commands are run synchronously.

You can use the *in_file*, *out_file* and *err_file* arguments to redirect the input, output or errors for the command.

Special syntax for OpenVMS

On OpenVMS, if you want OpenVMS to execute the command, you need to specify the characters "$ " (dollar-sign and a space) before the command name so that OpenVMS knows to look for an executable (.com or .exe) file or DCL symbol. If you explicitly specify a path and file extension, OpenVMS tries to execute that particular file in the specified path. You cannot specify both "$ " and a path.

The following example shows how you would use the **ExecCmd** command with the "$ " syntax. In this example, the **ExecCmd** command invokes the OpenVMS **SHOW DEFAULT** command, which prints the current directory to the A.OUT file. This example then invokes the **ListFile** command to display the contents of the A.OUT file:

```
ExecCmd "$ SHOW DEFAULT" "" A.OUT A.OUT
fscript > ListFile A.OUT
>>> BEGIN LISTING <<<
 1>   USER:[TOM]
>>> END LISTING <<<
```

The following example shows how you could use the **ExecCmd** command with a full path name and filename to invoke the Forte Corbagen executable:

On one line

```
ExecCmd "FORTE_ROOT:[INSTALL.BIN.ALPHA]CORBAGEN /CORBA_TYPE=OBB
/IDL_FILE=NEW.IDL"
```

Exit

Leave Fscript

The **Exit** command exits **Fscript.**

Syntax Exit

This command will prompt you to save changes before allowing you to leave without saving changes. To commit any outstanding changes, you must use the **Commit** command before using the **Exit** command.

The **Exit** command and the **Quit** command are synonyms.

ExitIfNoEnvMgr

Exit when no environment manager is running

The **ExitIfNoEnvMgr** command causes Fscript to exit as soon as there is no active environment manager.

Syntax ExitIfNoEnvMgr

The **ExitIfNoEnvMgr** has Fscript exit if no environment manager is currently running and Fscript is running in distributed mode.

You can use this command in batch scripts to detect error conditions.

ExitStatus

Set return value

The **ExitStatus** command sets a return value for this session of Fscript. This value is returned to the routine that started this session of Fscript when Fscript exits.

Syntax **ExitStatus** *integer*

Argument	Description
integer	The value returned to the routine that started Fscript. By default, the return value is 0 for if Fscript completed without errors or 1 if Fscript exited abnormally. You can define other numeric values that are meaningful to you.

The routine that started Fscript can check this return value to determine whether Fscript completed without errors or exited abnormally.

ExportClass

Export a class or interface definition

The **ExportClass** command exports the definition of a single class or interface in the current project into a text file for backup or for transfer to another project or repository.

Syntax **ExportClass** *class_name file_name* [**noids** | **ids**]

Argument	Description
class_name	The name of a class or interface in the current project that is to be exported to a file.
file_name	The name of a file into which to write the class definition. If the file already exists, it is replaced.
noids | **ids**	**noids** specifies that unique identifiers not be included in the class or interface definition. **noids** is the default. **ids** specifies that unique identifiers be included in the or interface definition.

The exported file can subsequently be read back in by using the **ImportClass** or **Compile** command.

You must specify the *file_name* argument in local operating system format, unless you have previously invoked the **UsePortable** command.

The **noids** | **ids** argument specifies whether universal unique identifiers (UUIDs) are included in the information exported to the file. **noids** (the default) specifies that UUIDs are *not* included in the export file. **ids** specifies that the UUIDs are included in the export file.

Note For subclasses of the UserWindow class, the output file contains a text encoding of the class's window and menu definitions. These parts of the file cannot be edited. You can use the **ExportWindowClass** command to write out a UserWindow subclass to a file and the encoded window file to a separate file.

ExportPlan

Export a plan definition

The **ExportPlan** command writes the definition of the current plan to a text file for backup or for transfer to another repository. The current plan can be a project, an application model, or a business model.

There are two different versions of the syntax of the **ExportPlan** command: one for projects, the other for application models and business models. Before invoking the **ExportPlan** command, you must use the **FindPlan** command to set the current plan.

Export a project

For a project, the ExportPlan command exports the definitions of all classes, service objects, interfaces, cursors, and constants in the project, as well as the list of the supplier projects and properties of the current project. The file is readable and contains the project definition statements described in the *TOOL Reference Manual*.

Syntax **ExportPlan** *file_name* [**noids** | **ids**]

Argument	Description
file_name	The name of a file to write the plan definition into, given in local operating system format. If the file already exists, it is replaced.
noids | **ids**	**noids** specifies that unique identifiers not be included in the class or interface definition. **noids** is the default. **ids** specifies that unique identifiers be included in the class or interface definition.

You must specify the *file_name* argument in local operating system format, unless you have previously invoked the **UsePortable** command.

The **noids | ids** argument specifies whether universal unique identifiers (UUIDs) and runtime IDs are included in the information exported to the file. **noids** (the default) specifies that these IDs are *not* included in the export file. **ids** specifies that these IDs are included in the export file.

Export an application model or business model

For an application model or business model, the ExportPlan command exports the definition of the model as text to a file. For more information about the export format of models, see *A Guide to Forte Express*.

Syntax **ExportPlan** *file_name*

Argument	Description
file_name	The name of a file to write the plan definition into, given in local operating system format. If the file already exists, it is replaced.

Import the plan

The file that you export can be re-imported into the same or a different repository by using the **ImportPlan** command.

You can edit the exported plan file, but you should be careful if you change the output of the **ExportPlan** command, as there are often two definitions of a class and its attributes and methods in the file. Project components are often preceded by the qualified project name, so if you try to change the project name, you must be sure to change the name in all references before importing the project.

ExportTemplate

Export Express template

The **ExportTemplate** command exports the current project as a Forte Express template file.

Syntax **ExportTemplate** *file_name*

Argument	Description
file_name	The file to contain the exported template.

This template file is exported to the current working directory as *file_name*.tpl. This template can then be used by advanced Forte Express developers to customize generated code.

Caution These templates are intended for use by consultants and specialized developers only. For information about using these templates, contact Forte Technical Support.

ExportWindowClass

Export a UserWindow subclass

The **ExportWindowClass** command exports the specified UserWindow subclass in the current project to two files:

- the specified *class_file* for the class definition
- the specified *window_file* for the window definition

Syntax **ExportWindowClass** *class_name class_file window_file* [**noids | ids**]

Argument	Description
class_name	The name of a class in the current project that is to export to a file.
class_file	The name of a file into which to write the class definition. If the file already exists, it is replaced.
window_file	The name of the file into which to write the window definition. If the file already exists, it is replaced.
**noids	ids**	**noids** specifies that class identifiers not be included in the class definition. **noids** is the default. **ids** specifies that class identifiers be included in the class definition.

Specify the *class_file* and the *window_file* arguments in local operating system format, unless you have previously invoked the **UsePortable** command.

The **noids | ids** argument specifies whether universal unique identifiers (UUIDs) are included in the information exported to the file. **noids** (the default) specifies that UUIDs are *not* included in the export file. **ids** specifies that the UUIDs are included in the export file.

You can use the **ExportWindowClass** command to create the separate window file for import into the Window Workshop. You could also export UserWindow subclasses by using the **ExportClass** command, in which case the window portion is exported in an encoded form in the exported class file.

To import the exported files, use the **ImportClass** or **Compile** command.

ExportWorkspace

Export a workspace

The **ExportWorkspace** command writes the definition of all plans in the current workspace to a readable text file for backup or for transfer to another repository.

Syntax **ExportWorkspace** *file_name* [**noids | ids**]

Argument	Description
file_name	The name of a file to write the plan definitions into, given in local operating system format. If the file already exists, it is replaced.
**noids	ids**	**noids** specifies that unique identifiers not be included in the class or interface definition. **noids** is the default. **ids** specifies that unique identifiers be included in the class or interface definition.

You must specify the *file_name* argument in local operating system format, unless you have previously invoked the **UsePortable** command.

The **noids | ids** argument specifies whether universal unique identifiers (UUIDs) and runtime IDs are included in the information exported to the file. **noids** (the default) specifies that these IDs are *not* included in the export file. **ids** specifies that these IDs are included in the export file.

If the workspace contains projects, the projects are exported as project definition statements as described in the *TOOL Reference Manual*. If you choose, you can modify the project definitions before you re-import the file.

If the workspace contains application models or business models, the file contains information that should *not* be modified.

To import one or more plan definitions, use the **ImportWorkspace** or **ImportPlan** command, as described in "ImportPlan" on page 56.

FindActEnv

Set the active environment to be current

The **FindActEnv** command designates the active environment as the current environment for use in partitioning the current application or partitioning a new library configuration.

Syntax FindActEnv

You must designate a current environment before partitioning any application or library project.

You can also use the **FindEnv** command to designate one of the simulated environments for partitioning.

FindAppComp

Designate the current component

The **FindAppComp** command designates a specified partition or library project within the current configuration that you want to set as the current application component.

Syntax **FindAppComp** *component_name*

Argument	Description
component_name	The name of an application component within the current application.

A number of **Fscript** commands operate on the current component within a configuration. The **FindAppComp** command designates the component to be used for those commands. Before you can work with partitions or library projects, you must partition the application or library project using the **Partition** command to generate the default configuration.

You can specify the trailing part of the component name (such as "client") to identify the component uniquely. Use the **ShowApp** command to get a list of components in the current configuration of the application.

FindEnv

Set the current environment

The **FindEnv** command designates the specified environment in the environment repository or environment manager as the current environment.

Syntax **FindEnv** *environment_name*

Argument	Description
environment_name	The name of a simulated environment.

The **FindEnv** command designates the specified simulated environment as the current environment for use in partitioning the current application or partitioning a new library configuration. You must designate a current environment before partitioning an application or library.

The *environment_name* must be the name of a simulated environment that is contained in the active environment's repository.

You can also use the **FindActEnv** command to designate the current active environment for partitioning.

FindPlan

Designate the current plan

The **FindPlan** command designates the specified plan in the current workspace as the current plan.

Syntax **FindPlan** *plan_name*

Argument	Description
plan_name	The name of a plan within the current workspace.

The plan specified by *plan_name* can be a project, an application model, or a business model in your workspace.

You must specify the name of a plan that has been included or created in your workspace. You might need to invoke an **IncludePublicPlan** command to add a public plan to your workspace before you can start to work on it.

The current plan can be a project, an application model, or a business model. When the current plan is an application model or a business model, there is no current project.

ForceWorkspaceUnreserved

Make a workspace unreserved

The **ForceWorkspaceUnreserved** command forces the specified workspace to become unreserved.

Syntax **ForceWorkspaceUnreserved** *workspace* [*workspace_password*]

Argument	Description
workspace	The name of the workspace to be unreserved.
workspace_password	The optional password for the workspace to be unreserved. If a password is set for this workspace, then the password is required.

The **ForceWorkspaceUnreserved** command should be used only when the shadow repository that has the reservation on the workspace has been corrupted or to release a lock held by a detached shadow repository. To recover changes made in a detached shadow repository whose lock is released with this command, you can export the workspace and import the changes into the central repository. If the shadow repository is attached to the central repository when you use the **ForceWorkspaceUnreserved** command, then the changes in this repository are lost.

When you detach a shadow repository from the central repository, the workspace is considered to be reserved to the shadow repository, and is locked in the central repository, so that other workspaces cannot open the workspace in read-write mode, or detach from the central repository using that workspace. The workspace can also be reserved to an attached shadow repository if the shadow repository contains changes that are not yet committed to the central repository.

Because shadow repositories do not use files that are as protected from corruption as the central repository, and because of the very nature of distributed work, shadow repositories can become corrupted or lost, and must be discarded. The central repository will, however, be completely up-to-date with the changes to the shadow repository up until the time of the last time the shadow repository was detached from the central repository or the last time an attached shadow repository committed its changes. Therefore, the central repository provides significant backup storage for the shadow repository. To remove the reservation on the workspace in the central repository, you should use the **ForceWorkspaceUnreserved** command.

To recover from a lost or corrupted shadow repository:

1. Use the Fscript **ForceWorkspaceUnreserved** command while connected to the central repository.
2. Create a new shadow repository, overwriting the existing corrupted shadow repository if necessary.

Help

Get help for Fscript

The **Help** command provides help for **Fscript** commands.

Syntax **Help** [*command_name* | *match_string*]

Argument	Description
command_name	The name of an Fscript command.
match_string	The partial name of a command followed by an asterisk.

With no arguments given, the **Help** command lists all Fscript commands. If you specify the *command_name* argument, this command lists the arguments and a short description for the specified Fscript command.

If a match string is given (an asterisk at the end of the string), this command lists all matching commands, their arguments and a short description. For example, the following command lists all of the Fscript **Set...** commands:

```
Fscript> Help Set*
```

ImportClass

Compile TOOL statements

The **ImportClass** command reads a file containing definitions of project components and inserts the components into the current project.

Syntax

ImportClass *file_name*

Argument	Description
file_name	The file containing commands that define components of a project.
	Note: The **ImportClass** command assumes a default file name extension of ".4gl".

The **ImportClass** command reads a file that contains TOOL statements that define one or more components or sub-components of a project and merges them into the current project. The **ImportClass** command and the **Compile** command are equivalent.

For information about importing and compiling an entire project, see "ImportPlan" on page 56.

Before you can invoke the **ImportClass** command, you must check out or branch any component that will be changed by the contents of the file. New components are automatically added and checked out to the workspace.

A definition file processed by the **ImportClass** command can include statements for creating classes, interfaces, service objects, constants, and cursors. For information about the syntax of the TOOL statements used in a project definition file, see *TOOL Reference Manual*. The **ImportClass** command checks for compilation errors as it reads the file.

You can use the **ImportClass** command to add an exported class or interface to add classes that contain references to window definitions contained in **.fsw** files created by the Window Workshop. For information about adding classes that contain references to window definitions, see the description of the **has file** clause of the **class** statement in the *TOOL Reference Manual*.

Caution

Any project component in a file processed by the **ImportClass** command completely replaces the current component definition. When you compile a new class or interface definition, the entire previous set of attributes, events, methods and method definitions are replaced for that class.

The following example illustrates a file that can be used with the **ImportClass** command. Both the class and a method are defined in the same file. You can add any number of classes, methods, and other components to the same file.

```
class weather inherits from object
has public
  method setctemp(temp : integer):void;
  lowf : integer;
  virtual lowc : integer = (get = (5.0/9.0)*(lowf- 32),
              set = SetCTemp(lowc));
end class;

method weather.setctemp(temp : integer) : void
begin
```

```
  self.lowf = (9.0/5.0)*temp + 32;
end method;
```

ImportPlan

Import a plan into the repository

The **ImportPlan** command reads a file containing a definition of one or more plans into your repository.

Syntax

ImportPlan *file_name* [**merge**]

Argument	Description
file_name	The name of a file containing plan definition information.
merge	(For projects only) Specifies whether to keep or delete components of the project that are in the repository but not in the import file. The default value is **merge**, which keeps components that are not in the import file. The value **nomerge** deletes components not present in the imported file. The value **prompt** prompts the user whether to keep or delete these components when importing a project. Application and business models are always replaced by the imported plan.

The plans can be projects, application models, or business models.

The **ImportPlan** command imports the plans specified in the file. The plans named there may or may not already exist in your workspace.

Import a project

If the plans are projects, then the definition information in the file could be created by either exporting projects using the **ExportPlan** or **ExportWorkspace** commands or by writing the definition information yourself using TOOL statements described in *TOOL Reference Manual*. If a project does not exist in your workspace, a new project is created. If a project does exist, components of the project that are redefined in the imported file are overwritten.

By default, if you import into a project that already contains components, the components in the export file will be merged with the existing components. Components in the export file using the same names as existing components will replace the existing components. If there are existing components in the project that have no new definitions in the export file, the components remain unchanged in the project. However, you can specify **nomerge** for the *merge* argument to have the imported projects completely replace the existing projects in the repository, which includes deleting components that are not included in the export file. You could alternatively specify **prompt**, to be prompted whether to merge or replace the projects.

Before you can import an existing project, all components that will be overwritten by newer versions from the imported file must be checked out or branched in your workspace. New components in the export file will be added and automatically checked out to the workspace. See “CheckoutAllComps” on page 39 or “BranchAllComps” on page 38 for information about branching or checking out all project components.

Import an application model or business model

If the plans are application models or business models, the definition information in the file must have been previously exported using the **ExportPlan** command. If the model does not exist in your workspace, a new model is created. If the model does exist, the imported model completely replaces the existing model.

Before you can import an existing model, the model must be checked out or branched in your workspace. See “CheckoutPlan” on page 40 or “BranchPlan” on page 39 for information about branching or checking out models.

Changes are automatically committed

ImportPlan automatically commits changes to the repository as it imports a project—if the command gets errors, successfully imported classes are committed to the repository.

ImportWorkspace

Import one or more plans into a repository

The **ImportWorkspace** command reads a file containing a definition of one or more plans into your repository.

Syntax

ImportWorkspace *file_name* [**merge**]

Argument	Description
file_name	The name of a file containing definition information for one or more plans.
merge	(For projects only) Specifies whether to keep or delete components of the projects that are in the repository but not in the import file. The default value is **merge**, which keeps components that are not in the import file. The value **nomerge** deletes components not present in the imported file. The value **prompt** prompts the user whether to keep or delete these components when importing a project. Application and business models are always replaced by imported plans.

The plans can be projects, application models, or business models.

The **ImportWorkspace** command imports the plans specified in the file. The plans named there may or may not already exist in your workspace.

Import projects

If the plans are projects, then the definition information in the file could be created by either exporting projects using the **ExportPlan** or **ExportWorkspace** commands or by writing the definition information yourself using TOOL statements described in *TOOL Reference Manual*. If a project does not exist in your workspace, a new project is created. If a project does exist, components of the project that are redefined in the imported file are overwritten.

By default, if you import into a project that already contains components, the components in the export file will be merged with the existing components. Components in the export file using the same names as existing components will replace the existing components. If there are existing components in the project that have no new definitions in the export file, the components remain unchanged in the project. However, you can specify **nomerge** for the *merge* argument to have the imported projects completely replace the existing projects in the repository, which includes deleting components that are not included in the export file. You could alternatively specify **prompt**, to be prompted whether to merge or replace the projects.

Before you can import an existing project, all components that will be overwritten by newer versions from the imported file must be checked out or branched in your workspace. New components in the export file will be added and automatically checked out to the workspace. See “CheckoutAllComps” on page 39 or “BranchAllComps” on page 38 for information about branching or checking out all project components.

Import an application model or business model

If the plans are application models or business models, the definition information in the file must have been previously exported using the **ExportPlan** or **ExportWorkspace** command. If the model does not exist in your workspace, a new model is created. If the model does exist, the imported model completely replaces the existing model.

Before you can import an existing model, the model must be checked out or branched in your workspace. See “CheckoutPlan” on page 40 or “BranchPlan” on page 39 for information about branching or checking out models.

Changes are automatically committed

ImportWorkspace automatically commits changes to the repository as it imports a project—if the command gets errors, successfully imported classes are committed to the repository.

Include

Run a script

The **Include** command executes the **Fscript** commands in the specified script file.

Syntax **Include** *file_name*

Argument	Description
file_name	The name of a file containing Fscript commands to execute.

You can store a commonly executed set of Fscript commands in a script file, then run the commands using the **Include** command.

The **Include** command uses the current directory search path in determining the location of the script file. See "SetPath" on page 87 for more information.

A script file can contain comments, which are any lines beginning with the # sign.

IncludePublicPlan

Include a plan in your workspace

The **IncludePublicPlan** command includes a plan in the current workspace. A plan can be a project, an application model, or a business model.

Syntax **IncludePublicPlan** *plan_name*

Argument	Description
plan_name	The name of the plan to include. This must be the name of a public plan that is not yet included in your workspace.

If you are using a shadow repository, you must be attached to the central repository before you can invoke the **IncludePublicPlan** command.

The plan you include must be in the repository, but must not yet be included in the current workspace. If this public plan has supplier plans that are not yet included in the current workspace, these plans are automatically included as well.

You must commit any outstanding changes to the repository before invoking the **IncludePublicPlan** command. The **IncludePublicPlan** command automatically commits the changes it makes to the workspace.

UpdateWorkspace command synchronizes plans

After invoking an **IncludePublicPlan** command in a workspace, you should immediately invoke the **UpdateWorkspace** command to synchronize with the other plans in the workspace.

The **IncludePublicPlan** command adds the plan to a workspace as it exists in the system baseline. You must invoke the **UpdateWorkspace** command to make sure that your workspace is synchronized with the system baseline for any plans that act as supplier plans to both the plan to be included, and any plans already included in your workspace. If you do not update the workspace, serious problems can occur because the newly included plan might have dependencies on plans that are newer than the plans and components used in your workspace.

The following example illustrates use of the **IncludePublicPlan** command:

```
Fscript> Commit
Fscript> IncludePublicPlan otherplan
Fscript> UpdateWorkspace
Fscript> FindPlan otherplan
```

IncreaseCompatLevel

Raise project compatibility level

The **IncreaseCompatLevel** command increases the compatibility level of the current project by 1.

Syntax IncreaseCompatLevel

Projects within an environment are identified by name and compatibility level. You can deploy an application where the compatibility level is set to one value, and then invoke the **IncreaseCompatLevel** command to increment the compatibility level for the project in order to continue development on the project. When you go to deploy the new version of the application, there will be no conflict with the previous version, and they can be executed at the same time.

IntegrateWorkspace

Integrate your workspace into the repository

The **IntegrateWorkspace** command checks in all the changes you made in the current workspace since your last **IntegrateWorkspace** command to the system baseline in the repository.

Syntax **IntegrateWorkspace** *comment* [*logfile_name*] [*baseline_password*]

Argument	Description
comment	A comment to enter into the repository. A comment must take one of three forms: a string enclosed in double quotes, a question mark (**?**), in which case Fscript will prompt for the comment, or a colon (**:**) followed by a file name.
logfile_name	The name of the log file in which to list the status of the **IntegrateWorkspace** operation. If not specified, the status will be put in the Fscript standard output.
baseline_password	If the repository has a baseline password, you must specify the password in the **IntegrateWorkspace** command.

After the **IntegrateWorkspace** command has completed, any other workspaces can invoke the **UpdateWorkspace** command to see the changes made by this workspace.

The *comment* argument is required and can take one of three forms. If a string enclosed in double quotes is specified, that is used as the comment. If a single question mark is specified, Fscript prompts for a comment to be entered. If a colon followed by a valid file name is specified, the comment is read from that file. Specify the comment file name in local operating system format, unless the **UsePortable** command has been invoked previously in the session.

The optional *logfile_name* argument specifies the name of a file where the integration status messages will be written, in addition to the standard output. This allows you to keep a record of the integration status, and any problems that might occur in integrating. Specify *logfile_name* in local operating system format, unless the **UsePortable** command has been invoked previously in the session.

The following examples show possible **IntegrateWorkspace** commands:

```
Fscript> IntegrateWorkspace “Changes to fix bug 1234”
Fscript> IntegrateWorkspace ? /mylogs/changelog.txt
Fscript> IntegrateWorkspace :comment.txt
```

You must commit any outstanding changes to the repository before invoking the **IntegrateWorkspace** command. Once completed, the changes made by the **IntegrateWorkspace** command are automatically committed to the repository.

Use **UpdateWorkspace** before using **IntegrateWorkspace**

To invoke the **IntegrateWorkspace** command, you must first use the **UpdateWorkspace** command to synchronize your workspace with the system baseline. After testing to ensure that your projects are compatible with any changes made to other projects and components, you can then invoke the **IntegrateWorkspace** command to make your changes public.

If, between the time you invoke the **UpdateWorkspace** command and the **IntegrateWorkspace** command, some other workspace has invoked the **IntegrateWorkspace** command, Forte rejects your **IntegrateWorkspace** command. If this occurs, reinvoke the **UpdateWorkspace** command, test to ensure that the recent changes have not caused problems, and then reinvoke the **IntegrateWorkspace** command.

The **IntegrateWorkspace** command checks in all components that you have checked out. To continue changing any plans in your workspace, you must invoke appropriate **CheckoutPlan**, **CheckoutComp**, or **CheckOutAllComps** commands.

A workspace cannot have any branched components or plans when you invoke the **IntegrateWorkspace** command. Before you can use the **IntegrateWorkspace** command, you must either convert them to checked out components by invoking the **CheckoutPlan** or **CheckoutComp** command on them, or revert to the system baseline version by invoking the **UndoBranchComp** or **UndoBranchPlan** command.

If you are using a shadow repository, you must be attached to the central repository before you can invoke the **IntegrateWorkspace** command.

Recovery from integration failures

If the integration fails, any changes that have been made to the repository are backed out, and the repository is in the same state as before the **IntegrateWorkspace** command was invoked. The global repository lock, which prevents other developers from updating or integrating their workspaces, is automatically removed as part of the recovery process.

If the **IntegrateWorkspace** command fails because the repository client session that invoked the command fails, the repository server might not release the global repository lock held by the client session. You might need to use the Fscript **UnlockWorkspace** command to notify the repository server that the client session is no longer active. The **UnlockWorkspace** command, described in "UnlockWorkspace" on page 101, unlocks both the workspace and any global repository locks held by that workspace. System managers can also unlock workspaces using Escript and Environment Console commands, as described in *Escript and System Agent Reference Manual*.

ListChangesInWorkspace

Show components in a workspace

The **ListChangesInWorkspace** command lists all components and plans in the workspace that have a specific status with respect to check out or branching.

Syntax **ListChangesInWorkspace** [*filter_type*]

Argument	Description
filter_type	The filter to use for limiting the listing to objects that only have a certain status (see below for details). The default is "bnc" for all components and plans.

You can filter the list by specifying a *filter_type* of any combination of:

Filter Type	Description
b	Branched
c	Checked out and/or removed
n	New
r	Read only

ListComps

Show components in a workspace

The **ListComps** command lists a subset of the project components in the workspace, with information about their branch or checkout status.

Syntax **ListComps** [*comp_match_string*] [*filter_type*] [*proj_match_string*]

Argument	Description
comp_match_string	Specifies the name of a component, or a match string for components (a prefix followed by an asterisk). The default is '*' for all components.
filter_type	The filter to use for limiting the listing to components that only have a certain status (see below for details). The default is "bncr."
proj_match_string	Specifies the name of a project in the current workspace, or a match string for projects (a prefix followed by an asterisk). The default is the current project only.

Use the *comp_match_string* argument to select a subset of component names (default is all components). You can give a prefix to the name, followed by an asterisk to list only some of the components (for example, "a*' will list only components starting with the letter a).

You can filter the list by specifying a *filter_type* of any combination of:

Filter Type	Description
b	Branched
c	Checked out and/or removed
n	New
r	Read only

The *proj_match_string* argument can be used to list components in the matching included projects in the workspace. This can be specified as a single project name, or a prefix followed by an asterisk to list all matching projects (for example, "pr*" will list components in all projects starting will the letters 'pr'). The default is to list components in the current project only.

ListEnvs

Show environment names

The **ListEnvs** command lists the names of the environments defined in the environment repository.

Syntax ListEnvs

This command will list the active environment name and any simulated environments that are in the active environment's repository. You can then invoke the **FindEnv** command or **FindActEnv** commands to designate one of the environments as the current environment for partitioning.

If you have started Fscript in standalone mode (with the **-fs** flag), ListEnvs cannot list any environments, because no environment repository is available.

ListFile

Show a file's contents

The **ListFile** command prints the contents of the specified file to standard output.

Syntax **ListFile** *file_name*

Argument	Description
file_name	The name of the file to print.

You should only print files containing text using the **ListFile** command.

If you do not specify a directory name, the command searches the current directory search path, as specified in the "SetPath" on page 87.

ListFiles

Show files in a directory

The **ListFiles** command lists the files in the specified directory.

Syntax **ListFiles** [*directory_name*]

Argument	Description
directory_name	The name of a directory. The default is the current working directory for Fscript.

This command lists the full names and permissions for all the files in the specified *directory_name*. If no *directory_name* is specified, the current working directory for Fscript is used. The current working directory is either the directory in which Fscript was started, or the last directory set by the **Cd** command.

Specify the *directory_name* in local operating system format unless you have invoked a **UsePortable** command for using portable file formats.

The **ListFiles** command, the **Ls** command, and the **Directory** command are synonyms.

ListPlans

Show the plans in your workspace

The **ListPlans** command lists all plans in the current workspace that match the specified match string. If you do not specify a match string, the default is "*," meaning all plans.

Syntax **ListPlans** [*match_string* | *]

Argument	Description
match_string	A string of characters that matches the name of the plans you want to list. You can include '*' at the end of the string to get a list of plans whose names start with the string.
*	An asterisk requesting a list of all plans available in this workspace.

A plan can be a project, an application model, or a business model.

If you specify a *match_string* argument (with an asterisk at the end of the name), matching plan names are shown. If you specify an asterisk or no arguments, all plans are listed.

The plans are listed under the following headings:

- Projects
- Business Models
- Application Models

ListPublicPlans

Show the plans in your repository

The **ListPublicPlans** command lists all the public plans in the repository. Public plans can be projects, application models, or business models.

Syntax **ListPublicPlans** [*match_string** | *] [*show_unintegrated*] [*show_internal*]

Argument	Description
match_string	A set of characters followed by an asterisk to match plan names to list. The default is all plans.
show_unintegrated	Set to **1** to show a listing of the plans that have been created in workspaces but not yet integrated, or set to **0** to show only integrated plans. Default (**0**) is integrated only.
show_internal	Set to **1** to show the Forte internal plans. Default (**0**) is not to show Forte internal plans.

If you specify a *match_string* argument (an asterisk at the end of the name), matching plan names are shown. If you specify an asterisk or no arguments, all plans are listed.

The *show_unintegrated* argument can be **0** (the default) for showing only plans that have been integrated into the system baseline, or **1** to also show plans that have been created in workspaces but not yet integrated.

The *show_internal* argument can be **0** (the default) to exclude internal Forte plans or **1** to list internal Forte plans.

If there are plans in the public plan list that you would like to include in your workspace, you can invoke the **IncludePublicPlan** command to include them.

The plans are listed under the following headings:

- Projects
- Business Models
- Application Models

ListServiceApps

Show applications containing service object

The **ListServiceApps** command lists the installed applications that contain the specified service object.

Syntax **ListServiceApps** *service_object_name*

Argument	Description
service_object_name	The name of a service object in a project in the current workspace. This must be given as a fully qualified name in the following format: *project_name.service_object_name*.

The **ListServiceApps** command lists the instances of a service object in the current environment. You can use these installed service objects as reference partitions in your current application. For more information about making reference partitions, see *A Guide to the Forte 4GL Workshops*.

Before you can invoke the **ListServiceApps** command, you must generate a default configuration for the current application by invoking the **Partition** command.

The **ListServiceApps** command lists any other instances of the specified service object that have been distributed (and presumably installed) in the current environment. You can invoke the **UseServiceFromApp** command to define one of the installed service objects as a reference partition for the specified service object. When you later make a distribution for

the current application and install the distribution in the environment, the current application will use the installed service object instead of creating and using a new service object.

For example, in the sample applications, there is a project called ImageProject, which contains a service object called ImageService. This project is defined as a supplier project to the ImageTester project, which has a simple window that displays images. You might invoke the following commands on the ImageTester project to configure and make a distribution for it:

```
Fscript> FindPlan ImageTester
Fscript> FindActEnv
Fscript> Partition
Fscript> Commit
Fscript> MakeAppDistrib
```

The resulting distribution can be installed in the environment. See the *Forte 4GL System Management Guide* document for information on installing applications.

Another project in the sample applications, called Auction, also includes the ImageProject as a supplier project. When configured, we can direct Auction to share the ImageService from the ImageTester by invoking the following commands:

During testing, a private version of the service object and partition are used in Auction, but once it is installed, the service from ImageTester is used in its place.

```
Fscript> FindPlan Auction
Fscript> FindActEnv
Fscript> Partition
Fscript> ListServiceApps ImageTester.ImageService
Fscript> UseServiceFromApp ImageTester.ImageService ImageTester_cl0
Fscript> Commit
Fscript> MakeAppDistrib
```

ListTestApps

Show projects names in environment

The **ListTestApps** command lists all the main projects that are currently being partitioned in another session of Fscript or a session of the Partition Workshop.

Syntax **ListTestApps** [*use_cache_flag*]

Argument	Description
use_cache_flag	Specifies whether to use a stored cache for the list of the main projects being partitioned (**1**), or whether the environment is to be queried (**0**). Default is **0**, meaning to query.

The *use_cache_flag* argument determines if the environment is to be queried again for the current list. **0** means do not use the current cached list and re-query the environment for the current list (the default), and **1** means use the current cached list.

The **ListTestApps** command, along with the **TestApp** command, let you set up tests of your distributed application with multiple client partitions running. You might want to do this to stress test your application, for example.

For information about setting up a test using the **ListTestApps** and **TestApp** commands, see "TestApp" on page 98.

You can also test client partitions using the TestClient utility. This utility is explained in *A Guide to the Forte 4GL Workshops*.

ListWorkspaces

Show workspaces

The **ListWorkspaces** command lists the workspaces in the current repository.

Syntax **ListWorkspaces** *[verbose_flag]*

Argument	Description
verbose_flag	Specifies whether to display only the names of the workspaces (**0**) or to list the names and some information (**1**). Default is names only.

If the *verbose_flag* is set to **1**, more information about each workspace is provided, such as whether the workspace is reserved.

Ls

Show files in a directory

The **Ls** command lists the full names and permissions for all the files in a specified directory.

Syntax **Ls** [*directory_name*]

Argument	Description
directory_name	The name of a directory. The default is the current working directory for Fscript.

If no *directory_name* is specified, the current working directory for Fscript is used. The current working directory is either the directory in which Fscript was started, or the last directory set by the **Cd** command.

Specify the *directory_name* in local operating system format unless you have invoked a **UsePortable** command for using portable file formats.

The **ListFiles** command, the **Ls** command, and the **Directory** command are synonyms.

MakeAppDistrib

Making application distribution directory

The **MakeAppDistrib** command creates an application distribution for the current configuration.

Syntax **MakeAppDistrib** [*remake_flag*] [*node_name*] [*auto_compile*] [*install*]

Argument	Description
remake_flag	Flag that specifies whether to make a completely new distribution or to make new distribution parts for components that have changed. **1** indicates a completely new distribution and **0** indicates new distribution parts for changed components. The default value is **0**.
node_name	The node where the distribution is to be written. Default or empty string ("") is the local node running Fscript.
auto_compile	Flag indicating that Forte should automatically try to compile the generated files. **1** specifies that Forte should automatically try to compile. **0** specifies that Forte should *not* automatically try to compile. The default value is **0**.
install	Flag indicating that Forte should try to automatically install the application. **1** specifies that Forte should try to automatically install the application. **0** specifies that Forte should not try to automatically install the application. The default value is **0**.

The **MakeAppDistrib** command makes the distribution files for Forte applications and libraries. The distribution files form the basis of the files needed for installation in a runtime environment.

The remake_flag argument specifies whether to make a completely new distribution or to make new distribution parts for components that have changed. **1** indicates a completely new distribution and **0** indicates new distribution parts for changed components. The default value is **0**.

The *node_name* argument specifies the node on which the distribution files are to be written. If it is omitted, or specified as an empty string (""), the distribution will be written on the node running Fscript. If you specify another *node_name*, the node manager must be currently running on that node in order to create the distribution on your behalf. In order to make a distribution on an alternate node, you must have the root Forte directory for the alternate node mounted on the local node.

The *auto_compile* argument indicates whether Forte should try to automatically compile the generated files. **1** specifies that Forte should automatically try to compile. **0** specifies that Forte should not automatically try to compile. The default value is **0**.

If the application configuration contains components that are to be compiled, and you do not specify **1** for the *auto_compile* flag, then you must use some operating system commands to actually code generate and link together the compiled partition before the distribution is complete and ready to be loaded into a deployed environment. For more information about compiling partitions and libraries, see *A Guide to the Forte 4GL Workshops*.

The *install* argument indicates whether Forte should try to automatically install the application or the libraries. **1** specifies that Forte should try to automatically install the application or the libraries. **0** specifies that Forte should not try to automatically install the application or the libraries. The default value is **0**.

An application distribution directory contains the files needed to install an application in a deployment environment. You should create one application distribution for each environment.

For information about distributions created for libraries and applications, see *A Guide to the Forte 4GL Workshops*.

C, DCE, ObjectBroker project distributions

The behavior of this command is different depending on whether you are making a distribution for a C, DCE, or ObjectBroker project or for a TOOL application. For C, DCE, and ObjectBroker projects, the **MakeAppDistrib** command writes out the C++ wrapper code needed to construct the library that provides access to the C functions or C client stubs.

For details on completing the distribution and installation of C, DCE, and ObjectBroker projects, see *Integrating with External Systems*.

TOOL application distributions

Be sure that the application configuration or library configuration is completely partitioned before running **MakeAppDistrib**. For information about partitioning, see *A Guide to the Forte 4GL Workshops* or "Partition" on page 73.

For information on installing application distributions in deployment environments, see the *Forte 4GL System Management Guide*.

MemStats

Get memory statistics

The **MemStats** command prints memory statistics to standard output.

Syntax

MemStats

You can use the **MemStats** command to get information about how memory is being used. You can use this information to diagnose memory problems, for example.

The following example shows sample output by the **MemStats** command:

```
fscript > memstats
! Memory Statistics
!
!   Minimum:2048  Incremental:1024  Maximum:10240  Utilization:85%
!   ExpandAt:80%  ExpandBy:10%  ContractAt:20%  ContractBy:5%
```

```
!   Mapped:4096  Active:2048  Allocated:1062  Available:368
!   LargePages:516  HugePages:0  Reserved:64  Immobile:50
!   Largest(Free:716  Possible:1028) Allocated(Peak:0  Total:1427)
!   Expansions:0  CopyCollects:1  TraceCollects:0
!
!-- Objects retained
!   <Small-Non-Objects>                  2096   158664       75
!   <Large-Non-Objects>                   142   491520     3461
!
!   TOTALS                               2238   650184
```

The following tables describes the contents of the output for this command:

Label	Description
Active	Current number of active pages in the memory manager.
Allocated	Current number of allocated pages in the memory heap.
Allocated(Peak: Total:)	Number of pages that have been allocated. Peak indicates the highest number of pages that have been allocated during this session and Total indicates the total number of pages that have been allocated during this session.
Available	Current number of available pages in the memory heap.
ContractAt	Threshold at which the memory pool should be contracted.
ContractBy	Percent by which the memory pool should be contracted.
CopyCollects	Number of incremental memory reclamations (garbage collections) that have occurred.
ExpandAt	Threshold at which the memory pool should be expanded.
ExpandBy	Percent by which the memory pool should be expanded.
Expansions	Number of times the memory has expanded.
Immobile	Objects in memory that cannot be moved by the memory manager.
Incremental	Incremental unit in pages by which memory is expanded or contracted.
LargePages	Number of pages contained in objects that are greater than 1K. These pages are not moved by the memory manager.
Largest(Free Possible)	Largest contiguous free block and the largest possible block.
Mapped	Size of memory allocated by Forte and by non-Forte components that are also running in the Forte process.
Maximum	Maximum number of pages that can be allocated.
Minimum	Minimum number of pages that can be allocated.
Requested	Number of pages requested in the most recent allocation, which failed. This item indicates that an out of memory exception occurred.
Reserved	Number of pages that are reserved in case an out of memory exception occurs, so that Forte can perform an orderly shutdown.
TraceCollects	Number of comprehensive memory reclamations (garbage collections) that have occurred.
Utilization	Target percentage of the active memory heap that should be allocated to live pages.

MkDir

Creating a directory

The **Mkdir** command creates a directory, in the operating system, with the name you specify.

Syntax **MkDir** *directory_name*

Argument	Description
directory_name	The name of the directory to create. Either use portable name syntax or the naming conventions appropriate for your particular operating system.

ModLogger

Change logger flags

The **ModLogger** command modifies the current logger flag settings for Fscript.

Syntax **ModLogger** +(*logger_flags*) | -(*logger_flags*)

Argument	Description
+(*logger_flags*)	Turn on the logger flag settings given in the parentheses.
-(*logger_flags*)	Turn off the logger flag settings given in the parentheses.

To specify additional logging flags, use "+" followed by a set of logger settings in parentheses. To remove specific logging flags, use "-" followed by a set of logger settings in parentheses. The new logging flags apply to Fscript itself and any applications run from Fscript using the **Run** or **RunDistrib** commands.

The **ModLogger** command modifies the logger flags that were specified when Fscript started execution. The logger flags are first set either with the **-fl** flag on the **fscript** command or from the value of the FORTE_LOGGER_SETUP environment variable.

The modified logger settings are only applied to the first file specified in the original logger settings for Fscript.

For a detailed description of the logger flag syntax, see "-Fl Flag (Forte Logger)" on page 116.

The following example shows possible uses of the **ModLogger** command:

```
Fscript> ModLogger +(trc:os:1:1 cfg:c4:2-3:1)
Fscript> ModLogger -(cfg:c4)
```

MoveServiceToPart

Move service object to another partition

The **MoveServiceToPart** command moves the specified service object to the specified server partition.

Syntax **MoveServiceToPart** *service_object_name* [*partition_name*]

Argument	Description
service_object_name	The name of a service object in the current project to move to the partition. This is specified as a fully qualified name, that is, *project_name.service_object_name*.
partition_name	The name of a partition where the service object is to be moved.

You should specify the *service_object_name* as a fully-qualified name in the following format:

project_name.service_object_name

If you do not specify the *partition_name*, Forte moves the service object to the current partition, as set by the **FindAppComp** command. You can identify the partition by using only the unique trailing portion of the partition name.

Before you can use the **MoveServiceToPart** command, you must generate the default configuration for the current application in the current environment using the **Partition** command.

You can invoke the **ShowApp** command to show the current partitioning for the current application in the current environment.

For information about the rules for moving service object between partitions, see *A Guide to the Forte 4GL Workshops*.

If you want to move a service object out of a partition to a new partition, use the **NewPart** command, explained in "NewPart" on page 69.

Mv

Rename file

The **Mv** command renames a file in the local file system.

Syntax **Mv** *old_file_name new_file_name*

Argument	Description
old_file_name	The name of the file to rename.
new_file_name	The new name for the file.

Specify the two file names relative to the current working directory. The current working directory is either the directory in which Fscript was started or the last directory specified using the **Cd** command.

Specify the *old_file_name* and *new_file_name* arguments in local operating system format unless you gave the **UsePortable** command to specify files using the portable file format.

A file with the name *new_file_name* must not exist before you invoke the **Mv** command.

NewPart

Create a new partition

The **NewPart** command creates a new partition in the current configuration, and moves a service object to the partition. The new partition becomes the current partition.

Syntax **NewPart** *service_object_name*

Argument	Description
service_object_name	The name of a service object in the current project to move to the newly created partition.

Before you can use the **NewPart** command, you need to generate the default configuration using the **Partition** command.

You can invoke the **ShowApp** command to show the current partitioning for the current application in the current environment.

To move a service object from one existing partition to another, use the **MoveServiceToPart** command (see "MoveServiceToPart" on page 68).

NewPlan

Create a new plan

The **NewPlan** command creates a new plan in the workspace. A plan can be a project, application model, or business model. The new plan becomes the current plan.

Syntax **NewPlan** *plan_type plan_name*

Argument	Description
plan_type	The type of plan to create. Legal values are: **project** to create a new project **ApplicationModel** to create a new application model **BusinessModel** to create a new business model
plan_name	The name of the new plan to create. This name must be unique for all plans in the repository.

Plan names must be unique

Plan names must be unique for all plans in the repository. When you add a new plan to the repository, other workspaces cannot see the contents of the plan until you integrate your workspace. In Fscript, you can use the **IntegrateWorkspace** command for your workspace to make the plan contents publicly available.

Rename a plan in a detached shadow repository

You can invoke a **NewPlan** command while running in a shadow repository that is detached from the central repository. When you attach to the central repository, however, you could get an error that indicates that the new plan name conflicts with a plan that was added while the shadow repository was detached, and the attachment is rejected. In that case, you must rename the new plan.

▶ **To rename a plan in the detached shadow repository:**

1. Export the new plan to a file, by invoking the **ExportPlan** command.
2. Edit the export file to replace any references to the plan name with another name. The plan name may be in a number of places in the file, so be sure to check carefully that you have changed all references.
3. Remove the plan as a supplier from any other plans using the **RemoveSupplierPlan** command.
4. Remove the new plan from your shadow repository using the **ExcludePlan** command.
5. Import the edited plan definition by invoking the **ImportPlan** command.
6. Add the renamed plan back into the supplier plan lists of any affected plans, using the **FindPlan** and **AddSupplierPlan** commands.

invoke the **AttachToCentral** command to attach the shadow repository to the central repository and check for further name conflicts.

NewProj

Create a new project

The **NewProj** command creates a new project in the workspace. The new project becomes the current project.

Syntax **NewProj** *project_name*

Argument	Description
project_name	The name of the new project to create. This name must be unique for all plans in the repository.

Project names must be unique

Project names must be unique for all projects in the repository. When you add a new project to the repository, other workspaces cannot see the contents of the project until you integrate your workspace. In Fscript, you can use the **IntegrateWorkspace** command for your workspace to make the project contents publicly available.

Rename a project in a detached shadow repository

You can invoke a **NewProj** command while running in a shadow repository that is detached from the central repository. When you attach to the central repository, however, you could get an error that indicates that the new project name conflicts with a project that was added while the shadow repository was detached, and the attachment is rejected. In that case, you must rename the new project.

To rename a project in the detached shadow repository:

1. Export the new project to a file, by invoking the **ExportPlan** command.
2. Edit the export file to replace any references to the project name with another name. The project name may be in a number of places in the file, so be sure to check carefully that you have changed all references.
3. Remove the project as a supplier from any other projects using the **RemoveSupplierPlan** command.
4. Remove the new project from your shadow repository using the **ExcludePlan** command.
5. Import the edited project definition by invoking the **ImportPlan** command.
6. Add the renamed project back into the supplier project lists of any affected projects, using the **FindPlan** and **AddSupplierPlan** commands.

invoke the **AttachToCentral** command to attach the shadow repository to the central repository and check for further name conflicts.

NewWorkspace

Create a new workspace

The **NewWorkspace** command creates a new workspace in the current repository.

Syntax

NewWorkspace *workspace_name* [*initial_password* [*admin_password*]]

Argument	Description
workspace_name	The name of the new workspace to create. This name must not be the name of an existing workspace in the current repository.
initial_password	The initial password for the workspace. If the current repository is a secure repository, you must specify an initial password. If the current repository is a standard repository, a workspace password is not required.
admin_password	The administrator password for the current repository. In a secure repository, you must specify the administrator password. In a standard repository, do not specify this password. This password is ignored if you specify it.

For example, to create a new workspace using the Fscript **NewWorkspace** command on a secure repository, specify the following command:

```
NewWorkspace PrivateWorkspace mysecret123 pwadmin8
```

Before you invoke the **NewWorkspace** command, you should close the current repository session using the **Close** command. Forte automatically adds the new workspace to the repository and opens the new workspace in read-write mode.

The new workspace contains only the Forte system libraries. You can invoke any number of **IncludePublicPlan** commands to add projects to your workspace.

Open

Open current workspace

The **Open** command opens the current workspace on the current repository.

Syntax

Open [**readonly|readwrite|exclusivewriter|exclusive**] [*workspace_password*]

Argument	Description
readonly	Open the current workspace in read-only mode. No changes to any aspects of the workspace will be allowed. If the workspace is being used, any number of people can open a workspace in read-only mode, but only one can open in read-write mode.
readwrite	The default, opens the current workspace in read-write mode, exclusively. If the workspace is being used concurrently in either read-write or read-only mode, the **Open** will fail. If you open in read-write mode, you can make changes to any parts of the workspace that are checked out.
exclusivewriter	Locks the entire repository against read-write access by any other users. If any users are currently using the repository in read-write mode, the **Open** command will fail.
exclusive	Locks the entire repository against all access by other users. If any other users are currently using the repository, the **Open** will fail.
workspace_password	If the workspace has a password, you must specify the correct password in the **Open** command. The only exception is that you do not have to use the workspace password to open a detached shadow repository in read-only mode.

To open a workspace:

1. Specify the current repository using the **SetRepos** command.
2. Specify the current workspace using the **SetWorkspace** command.
3. Enter the **Open** command.

The **Open** command opens the repository and workspace in the mode as specified with the command. By default, the **Open** command opens the workspace in read-write mode.

If you have already been working in a repository and workspace in this **Fscript** session, you must invoke the **Close** command before invoking a new **Open** command.

Set the current repository

The current repository is set by one of the following, in order of precedence:

1. **SetRepos** command (see "SetRepos" on page 90)
2. **-fr** flag on the **fscript** command that started this Fscript session (see "fscript Command" on page 17)
3. environment variable FORTE_REPOSNAME (see *A Guide to the Forte 4GL Workshops*)
4. default central repository called Central Repository

Set the current workspace

The current workspace is set by one of the following, in order of precedence:

1. **SetWorkspace** command (see "SetWorkspace" on page 91)
2. **-fw** flag on the **fscript** command that started this Fscript session (see "fscript Command" on page 17)
3. environment variable FORTE_WORKSPACE (see *A Guide to the Forte 4GL Workshops*)
4. default workspace called FirstWorkspace

Rules for locks and modes set by **Open**

When you open a workspace, a lock is placed in the repository for the workspace, indicating that someone is using the workspace in either read-write or read-only mode. If a user has opened the workspace and established a read-write lock on it, then no other users can open the workspace in any mode. You cannot open a workspace in read-write mode if other users currently have the workspace open in read mode.

Partition

Partition an application or library project

The **Partition** command produces a default configuration for the current project in the current environment. The current project must be the main project for an application or a library project.

Syntax **Partition** [**1** | **3**]

Argument	Description
1	The default. Partitions an application or library project incrementally, changing only those things that invalidate an existing configuration—usually changes in source code. Adding a node to an environment does not necessarily invalidate the configuration for a library or application. To incorporate changes in an environment, you should use the more comprehensive partitioning function provided by option **3**.
3	Comprehensively repartitions the current application or library project in the current environment, replacing any existing configuration.

Before you can invoke the **Partition** command, you must invoke a **FindPlan** command to set the current project, and then invoke a **FindEnv** command or **FindActEnv** command to set the current environment.

The current project must be either:

- the main project for an application, with the project type set as a user application (**1**) or a server-only application (**2**) using the **SetProjType** command

  If the current project is the main project for an application, you can specify the starting class and method for the project using the **SetProjStart** command.

- a library project, with the project type set as library (**3**) using the **SetProjType** command

All components in the current project must compile without errors (warnings are allowed) before you can successfully partition the application or library project.

The **Partition** command uses information about the service objects defined for each project in the application and the constraints and resources within the environment to generate the default configuration for the application within the environment.

After you have partitioned the application, you can change the default configuration by invoking the following Fscript commands:

- Use the **NewPart** and **MoveServiceToPart** commands to move service objects among partitions.
- Use the **AssignAppComp**, **UnassignAppComp** commands to assign partitions or library projects for installation on specific nodes.
- Use the **EnableAppComp** and **DisableAppComp** commands to specify which nodes automatically start their partitions.
- Use the **SetPartRepCount** command to change properties of the partitions.

PrintEnv

Print environment variable

The **PrintEnv** command prints the current value of the specified environment variable.

Syntax **PrintEnv** *variable_name*

Argument	Description
variable_name	The name of an environment variable.

The *variable_name* argument specifies the environment variable name.

Pwd

Show working directory

The **Pwd** command prints the name of the current working directory.

Syntax Pwd

The current working directory is either the directory in which Fscript was started or the last directory specified using the **Cd** command.

Quit

Leave Fscript

To leave **Fscript**, use the **Quit** command.

Syntax Quit

This command will prompt you before allowing you to leave without saving changes. To commit any outstanding changes, you must use the **Commit** command before using the **Quit** command.

The **Exit** command and the **Quit** command are synonyms.

ReadIntoFile

Writing to a file

The **ReadIntoFile** command reads subsequent lines and writes them to the specified file until Fscript encounters the terminating string.

Syntax **ReadIntoFile** *file_name* [*term_str*]

Argument	Description
file_name	The file into which to write the text. This file name is in local format. Fscript creates a new file by this name in the current working directory and writes the data into the file as lines.
term_str	The string that terminates the data. The default value is "!!". The terminating string must begin on a new line.

The data being read must begin on a new line following the line containing the **ReadIntoFile** command. The data does not need any additional punctuation to be read correctly. Fscript does not display the "fscript>" prompt until you enter the terminating characters.

For example, could contain the following lines:

```
The main project has been updated.
No errors occurred.
!!
```

RemoveAlias

Remove an alias

The **RemoveAlias** command removes the specified alias expansion mapping. An *alias* is a synonym you can make for an Fscript command and its arguments using the **AddAlias** command.

Syntax **RemoveAlias** [*alias_name*]

Argument	Description
alias_name	The name of the alias expansion mapping to be removed.

If *alias_name* is not specified, then **RemoveAlias** removes all alias mappings.

You can see a list of all available aliases using the **ShowAlias** command.

RemoveConf

Remove a configuration

The **RemoveConf** command removes the specified configuration for the current project.

Syntax **RemoveConf** [*configuration_name*]

Argument	Description
configuration_name	The name of the configuration to remove.

The *configuration_name* is the name of the environment for which this configuration was defined. You can create a configuration by invoking the **Partition** command, which uses the current project and the current environment. You can create a different configuration for the current project for each possible environment.

If you do not specify a *configuration_name*, Forte removes the current configuration.

To set the current configuration:

1. Use the **FindPlan** command to set the current project, which should be the main project for the application or the library project associated with the configuration you want to remove.
2. Use the **FindEnv** or **FindActEnv** commands to set the current environment, which should be the environment associated with the configuration you want to remove.

   You can use the **ShowPlan** command to list the names of the configurations available for the current project. The configurations have the same names as their environments.

RemoveComp

Remove a project component

The **RemoveComp** command removes the specified project component from the current project.

Syntax **RemoveComp** *component_name*

Argument	Description
component_name	The name of a component in the current project to be removed.

The specified *component_name* is a class, constant, service object, interface, or cursor defined in the current project, as specified by the **FindPlan** command.

If the component has been integrated, you must check out the component to your workspace, using the **CheckoutComp** or **CheckoutAllComps** commands before you can invoke the **RemoveComp** command. If the component is newly created in your workspace, you do not need to check out the component before using the **RemoveComp** command.

Until you invoke an **IntegrateWorkspace** command, you can undo the removal of a checked out component by invoking the **UndoRemoveComp** command.

RemoveFile

Delete file

The **RemoveFile** command deletes the specified file in the local file system.

Syntax **RemoveFile** *file_name*

Argument	Description
file_name	The name of the file to remove.

The **RemoveFile** command removes a file in the local file system. The *file_name* is specified relative to the current working directory. The current working directory is either the directory in which Fscript was started or the last directory specified using the **Cd** command.

The **RemoveFile** command and the **Rm** command are synonyms.

RemoveProjFromLib

Remove a project from a library configuration

The **RemoveProjFromLib** command removes the specified project from the current library configuration.

Syntax **RemoveProjFromLib** *project_name*

Argument	Description
project_name	The name of the project to remove from the current library configuration.

You can only use the **RemoveProjFromLib** command when the current configuration is a library configuration.

For information about setting the current library configuration or creating a new library configuration, see "AddProjToLib" on page 35.

RemovePublicPlan

Remove a public plan from the repository

The **RemovePublicPlan** command removes the specified public plan from the repository. A public plan can be a project, an application model, or a business model.

Syntax **RemovePublicPlan** *plan_name*

Argument	Description
plan_name	The name of the public plan to remove from the repository.

Before you can remove a public plan from a repository, you need to make sure that the plan is not included in any workspace and is not a supplier plan to any other plan.

▶ **To remove an integrated plan from the repository:**

1. Exclude the plan from all workspaces that include it, using the **ExcludePlan** command (see "ExcludePlan" on page 48).
2. Use the **RemovePublicPlan** command to remove the plan from the current repository.

RemovePublicPlan automatically commits changes to the repository after it removes the plan.

RemoveSupplierPlan

Remove a supplier plan

The **RemoveSupplierPlan** command removes a plan from the list of supplier plans for the current plan. A supplier plan can be a project, an application model, or a business model.

Syntax **RemoveSupplierPlan** *plan_name*

Argument	Description
plan_name	The name of the plan to remove from the supplier list. This must be the name of a plan included as a supplier to the current plan.

Once you have invoked the **RemoveSupplierPlan** command, any references to components of that plan will cause compilation errors.

RemoveWorkspace

Delete a workspace

The **RemoveWorkspace** command removes a workspace from the current repository.

Syntax **RemoveWorkspace** *workspace_name* [*workspace_password*]

Argument	Description
workspace_name	The name of the workspace to delete. This name must be the name of an existing workspace in the current repository.
workspace_password	If the workspace has a password, you must specify it in the **RemoveWorkspace** command.

Before you invoke the **RemoveWorkspace** command, make sure that *no* components are checked out to this workspace, and that no other developers have the workspace open (not even in read-only mode).

The **RemoveWorkspace** command always prompts for verification, even if you are running from a command script. You can include a single "y" in your command script to answer the prompt without stopping the running script, as shown:

```
...
RemoveWorkspace oldWorkspace myPassword
y
...
```

The **RemoveWorkspace** command deletes any projects that have been created in that workspace but never integrated.

The **RemoveWorkspace** command commits the changes it makes in the workspace to the central repository. This command discards any changes in branched components or project definitions that have not yet been integrated from this workspace.

RenameComp

Rename a project component

The **RenameComp** command renames the specified project component in the current project.

Syntax **RenameComp** *old_name new_name*

Argument	Description
old_name	The name of an existing component in the current project to rename.
new_name	The new name to be given the component.

The specified *old_name* is the name of a class, constant, service object, interface, or cursor defined in the current project, as specified by the **FindPlan** command.

Because component names must be unique within a project, no components in the current project can have the name you specify as *new_name*.

After you rename the component, you must find and rename all references to this component in the method source code, cursor text, or virtual attribute expressions in your source code. References in other contexts to the component automatically reflect the new name.

Repeat

Repeat a command

The **Repeat** command repeats the next command the specified number of times.

Syntax **Repeat** *rep_count*

Argument	Description
rep_count	The number of times you want the next command to be repeated.

You can use the **Repeat** command in a script to repeat an Fscript command. For example, you might want to repeat an **Include** command that runs a script a specified number of times.

RevertProj

Undo all changes to a project

The **RevertProj** command removes all changes made to the current project in the current workspace since the last time this workspace was integrated.

Syntax RevertProj

The **RevertProj** command reverses changes you made to the project since the last time you integrated the workspace using the **IntegrateWorkspace** command. The project is now the same as it was in the system baseline the last time you updated the workspace using the **UpdateWorkspace** command.

The **RevertProj** command releases any checkout locks you have on project components in this workspace.

Rm

Remove an operating system file

The **Rm** command deletes the specified file in the local file system. The *file_name* is specified relative to the current working directory.

Syntax **Rm** *file_name*

Argument	Description
file_name	The name of the file to remove.

The **RemoveFile** command and the **Rm** command are synonyms.

Run

Test the project

The **Run** command runs the current application in test mode using the starting class and method for the current project.

Syntax **Run** [*arguments*]

Argument	Description
arguments	Command-line arguments to be passed to the client partition.

Before you can run the application using the **Run** command, you need to set the current application using the **FindPlan** command to set the main project of the application as the current plan. You also need to set the starting class and method for the main project of the application using the **SetProjStart** command.

You can invoke the **Run** command to test the current application before partitioning the application. The **Run** command tries to run as much of the current application as possible in the client partition for the application. Even references to service objects that are declared with visibility of environment are run in the client partition, even though these service objects can only run on server partitions when deployed.

If the application needs to use projects or service objects that are not available in the client partition, such as for database access, the **Run** command partitions only those portions of the application into servers on other systems. If your application contains such service objects, you cannot run **Fscript** as a stand-alone utility (with the **-fs** flag specified). You must be connected to the name server and the network.

To run an application with the real partitioning behavior, you must partition the application, perhaps using the **Partition** command, then run the application using the **RunDistrib** command.

If you want to test part of an application that does not have a starting class and method, for example, a project containing only services, you can use the **RunFile** command to specify startup code for the project.

Note You cannot use the **Run** command when an application model or business model is the current plan.

▶ **To test an application generated from an application model:**

1. Make an application model the current plan using **FindPlan**.
2. Generate the application code for the application model using **CompilePlan**.
3. Make the main project of the generated application the current plan using **FindPlan**. The main project of the generated application is named *application_model_name***Client**.
4. Use the **Run** command to test run the generated application.

   To test the generated application in distributed mode, partition the application with the **Partition** command, then use the **RunDistrib** command.

RunDistrib

Run project using distributed configuration

The **RunDistrib** command runs the current project as a distributed application within the current configuration.

Syntax **RunDistrib** [*arguments*]

Argument	Description
arguments	Command-line arguments to be passed to the client partition.

The **RunDistrib** command automatically starts servers, sends partitions across the network, and does everything required to test the distributed execution of an application.

▶ **To use the RunDistrib command for the first time for an application:**

1. Set the main project for the application as the current project by using the **FindPlan** or **NewProj** command.
2. Make sure that you have set a starting class and method (with no parameters) as a starting point for the current project. Use the **SetProjStart** command if necessary.
3. Partition the current application using the **Partition** command.
4. Use the **RunDistrib** command to run a test for the configuration of the application.

Before you use the **RunDistrib** command, you should use the **Run** or **RunFile** commands to test for basic errors. The **Run** and **RunFile** commands let you test the project without running in a distributed environment. Running your application entirely as a client partition is generally a much faster and more flexible way to debug the basic logic of your application.

If other Fscript sessions have invoked the **TestApp** command so that they can participate in testing the client partitions of the application, the **RunDistrib** command sends the client partition to those Fscript sessions. For information about setting up client partitions for testing, see "TestApp" on page 98.

RunDistrib command starts remote servers

After the **RunDistrib** command completes execution of the client portion of an application and returns to the Fscript command prompt, the remote partitions for the application are still running. The remote partitions do not automatically shut down when the client shuts down because the remote partitions are often set up to be shared among client partitions.

Leaving the remote servers running is useful, because you can execute the **RunDistrib** command several times to test the logic of a client attaching to the running server.

To stop these remote server partitions, use the **StopRemoteParts** command described in "StopRemoteParts" on page 97.

RunFile

Test TOOL code fragments

The **RunFile** command executes the TOOL code fragment contained in the specified file.

Syntax

RunFile *file_name*

Argument	Description
file_name	The name of a file containing a block of TOOL commands.

The **RunFile** command can run parts of an application in test mode, and cannot test application partitioning. To run a project with partitioning, use the **RunDistrib** command.

The *file_name* argument you specify for the **RunFile** command should contain a block of TOOL code. Forte compiles and executes this block of TOOL code directly. Forte runs this block of code within the context of the current project, so you can reference components of the current project and its supplier projects within the TOOL code fragment.

The *file_name* should be specified in local operating system format, unless you have previously invoked the **UsePortable** command. The **RunFile** command uses the current directory search path, as specified with the **SetPath** and **AddPath** commands.

The following example shows the content of a file that the **RunFile** command could use:

```
io : BasicIO = new;
a : artist = new;
a.PromptForInput(io);
a.WriteToLog();
```

Save

Save changes to the attached shadow repository

The **Save** command saves the current changes in your workspace to the attached shadow repository. This command lets you save the changes in your workspace without committing the changes to the central repository.

If you are not using any shadow repository, the **Save** command commits changes directly to the central repository.

Syntax

Save

The **Save** command lets you save your workspace changes without interacting with the central repository. To commit your changes to the central repository when using an attached shadow repository, you must use the **Commit** command (described in "Commit" on page 42).

Using the **Save** command for incremental changes can improve repository performance, because the central repository can handle a larger number of users and the performance for each user is improved. The disadvantage to using the **Save** command is that you can lose the changes you make in your workspace if your shadow repository is lost before you commit your changes to the repository.

Script

Record Fscript commands

The **Script** command captures Fscript commands entered by the interactive user and writes them into the specified file.

Syntax **Script** *file_name*

Argument	Description
file_name	The name of an operating system file.

The **Script** command tells Fscript to start writing all Fscript commands entered by the user into a separate file. You can use the **Script** command to capture a session of Fscript commands that you can later edit or replay using the **Include** command.

The *file_name* argument is specified in local operating system format, by default. If you invoke the **UsePortable** command, however, the *file_name* must be specified in portable file format.

SearchFile

Search for text in a file

The **SearchFile** command prints lines of the specified file that contain the *searchText* text string into standard output.

Syntax **SearchFile** *file_name searchText* [*replaceText*] [*showLineNumbers*]

Argument	Description
file_name	The name of an operating system file.
searchText	The text string that is being searched for, enclosed in single quotes.
replaceText	The text string that is supposed to replace the *searchText*, enclosed in single quotes.
showLineNumbers	Flag that specifies **TRUE** to show the number for each printed line within the file or **FALSE** to show no numbers.

The *file_name* is specified relative to the current working directory. The current working directory is either the directory in which Fscript was started or the last directory specified using the **Cd** command.

Specify *searchText* in single quotation marks.

Specify *replaceText* in single quotation marks. When *replaceText* is specified, **SearchFile** replaces the occurrences of *searchText* with *replaceText* before printing the lines from the file.

showLineNumbers can be **1** or **0**. **1** means that line numbers will be printed with the lines containing *searchText*. The default is **0**.

SetAppCompCompiled

Specify that a library or partition be compiled

The **SetAppCompCompiled** command indicates whether or not the specified component from the current application or library configuration should be used in compiled form on the specified node.

Syntax **SetAppCompCompiled** *node_name compiled_flag* [*component_name*] [*C++_library*]

Argument	Description
node_name	The name of the node where the component will be installed.
compiled_flag	Flag that specifies whether or not the component will be compiled.
component_name	The name of the component. By default, the component is the current component.
C++_library	Flag that specifies whether or not a C++ library will be generated for this component. By default, the value is 0.

The *node_name* argument is the name of the node where the component will be installed.

The *compiled_flag* argument specifies whether or not the component will be compiled. This value can be:

Value	Description
0	(the default) installs the interpreted version of the component
1	installs the compiled version of the component

If you do not specify that this component be installed as compiled, then the component is installed as interpreted.

The *component_name* is the name of the component for which you are specifying whether the component will be compiled. By default, the *component_name* is the name of the current component.

The *C++_library* flag specifies whether or not to generate a C++ library for the specified component. This value can be:

Value	Description
0	(the default) does not generate a C++ library
1	generates a C++ library

For information about writing C++ applications that use the generated C++ library to access this Forte component, see *Integrating with External Systems*.

If you define a library or partition, you need to compile this application component as part of deploying it. You can compile the library or partition using the auto-compile argument of the Fscript **MakeAppDistrib** command or using the Forte **fcompile** command. The **MakeAppDistrib** command is described in "MakeAppDistrib" on page 65. Using the **fcompile** command for compiling partitions and libraries is described in *A Guide to the Forte 4GL Workshops*.

SetAppletFlag

Define a client partition as an applet

The **SetAppletFlag** command defines the client application associated with a logical client partition as an applet.

Syntax **SetAppletFlag 0** | **1**

Argument	Description
0 | ***1***	Specifies whether a client partition is an applet or not. By default, the client partition is not an applet (**0**). To define a client partition as an applet, specify **1**.

An *applet* is an application that is intended to be started only by another application. An application can start an application or applet using the LaunchMgr.RunApplet method.

Before you can invoke the **SetAppletFlag** command, you need to identify an application and select the client partition as the current component, as shown in the following example:

```
fscript> FindPlan Banking
fscript> FindActEnv
fscript> FindAppComp Client
fscript> SetAppletFlag 1
```

For more information about writing applications that use applets, see *Forte 4GL Programming Guide*. For information about the LaunchMgr.RunApplet method, see the AppletSupport library in the Forte online Help.

SetDefault

Set the working directory

The **SetDefault** command calls the operating system to change the current working directory.

Syntax

SetDefault *directory_name*

Argument	Description
directory_name	The name of the directory to which you want to change the current working directory.

By default, the *directory_name* should be specified in local operating system format. However, if you have previously invoked the **UsePortable** command, the *directory_name* should be specified in portable file name format. You can embed special syntax in the *directory_name* to have environment variable expansion performed on the specified name. See "SetPath" on page 87 for more details.

The **Cd** command and the **SetDefault** command are synonyms.

SetEnv

Set the environment variable

The **SetEnv** command sets an environment variable to a specified value.

Syntax

SetEnv *variable_name* [*value*]

Argument	Description
variable_name	The name of the environment variable whose value you want to set
value	The value that you want to use to set the environment variable.

The *variable_name* is the name of the environment variable you want to set. The variable setting is available in this session of Fscript, as well as to any applications or scripts you might start in Fscript.

On UNIX, VMS, and Windows NT, the variable disappears from the environment when Fscript finishes executing.

The *value* sets the value you want to assign to the environment variable. If you do not specify *value*, the environment variable is set to NIL.

SetOutFile

Set the output file

The **SetOutFile** command sets the specified file as the output file.

Syntax **SetOutFile** [*file_name*]

Argument	Description
file_name	The simple name of a file to be specified as the output file.

The *file_name* is specified relative to the current working directory. The current working directory is either the directory in which Fscript was started or the last directory specified using the **Cd** command.

If you do not specify *file_name*, then the output file is reset to "%stdout."

SetPartArgs

Set argument string for a partition

The **SetPartArgs** command sets the argument string that will be passed to the specified assigned partition on a particular node when it is started.

Syntax **SetPartArgs** *node_name arguments* [*partition_name*]

Argument	Description
node_name	The name of the node on which a partition is assigned.
arguments	The command line arguments to send to the partition when started.
partition_name	The name of a partition in the current configuration.

You can use the **SetPartArgs** command to specify startup command line flags for a partition on a specific node. The command operates on the current configuration of the application.

▶ **To specify the current configuration:**

1. Set the main project for the application as the current project using the **FindPlan** command.
2. Set the current environment using the **FindEnv** or **FindActEnv** commands.

If you have not yet partitioned the application in the current environment, you must invoke the **Partition** command before using the **SetPartArgs** command.

The *node_name* argument specifies one of the nodes to which you have assigned the partition, either by default with the **Partition** command, or deliberately with the **AssignAppComp** command.

The *arguments* argument specifies a set of command line arguments to send to the partition when it first starts up. These are given as UNIX-style command line arguments on all systems, using the "-" to designate the argument, and then the value. You should enclose the set of arguments in double quotation marks.

On all platforms, the **-fm** flag arguments need to be enclosed in quotation marks when they include embedded spaces, as shown:

```
Fscript> SetPartArgs myserver ”-fm (n:4000, x:8000)”
```

If the **-fm** flag arguments do not contain spaces in that flag, as shown in the following example, you can simply specify double quotes around the entire flag:

```
Fscript> FindPlan Acctg
Fscript> FindActEnv
Fscript> Partition
Fscript> FindAppComp AcctMgr
```

```
Fscript> SetPartArgs myserver “-fm(n:4000,x:8000)”
Fscript> Commit
```

On UNIX platforms, you need to enclose all arguments that contain parentheses or spaces in quotation marks so that the operating system can parse the flags correctly, as shown in the following example:

```
SetPartArgs Server1 ‘-fl “%stdout(trc:user) trc1_3.log(trc:user1)”’
```

For explanations of the **-fl** and **-fm** flags, see Appendix B, “Memory and Logger Flags.”

The optional argument *partition_name* specifies a partition other than the current partition. If this argument is not specified, the **SetPartArgs** command operates on the current partition, which you set using the **FindAppComp** command

SetPartRepCount

Set replication count for a partition

The **SetPartRepCount** command sets the replication count for the specified partition on a specified node.

Syntax **SetPartRepCount** *node_name replication_count* [*partition_name*]

Argument	Description
node_name	The name of the node on which a partition is assigned.
replication_count	The new replication count for automatically starting up servers for the partition.
partition_name	The name of a replicated partition within the current application.

The replication count indicates the number of instances of the partition will automatically be started on a particular node.

To specify the current configuration:

1. Set the main project for the application as the current project using the **FindPlan** command.
2. Set the current environment using the **FindEnv** or **FindActEnv** commands.

   If you have not yet partitioned the project in the current environment, you must invoke the **Partition** command before using the **SetPartArgs** command.

The *node_name* argument specifies one of the nodes to which you have assigned the partition, either by default with the **Partition** command, or deliberately with the **AssignAppComp** command.

The *replication_count* specifies the number of replicates of the partition that you want Forte to automatically start on the specified node.

The optional argument *partition_name* specifies a partition other than the current partition. If this argument is not specified, the **SetPartRepCount** command operates on the current partition, which you set using the **FindAppComp** command

When you create a partition that contains a replicated service object, either for failover or for load balancing, you can specify a replication count for the partition. Forte uses the replication count to determine how many replicates of the partition to automatically start when your application needs the services provided by the partition. You can only set the replication count for partitions that contain replicated service objects.

SetPassword

Set a password for a repository or workspace

The **SetPassword** command sets the master, administrator, baseline, or workspace passwords for the current repository or current workspace.

Syntax **SetPassword** *password_type new_password* [*current_password*]

Argument	Description
password_type	The type of password you are setting.
new_password	The new password you are setting for the repository or workspace.
current_password	The current password for the repository or workspace.

The *password_type* argument specifies whether the kind of password you are setting for your current repository or for your workspace. You can set the current repository using the **SetRepos** command. You can set the current workspace using the **SetWorkspace** command.

password_type can have one of the following values:

***password_type* value**	**Description**
admin	For a secure repository, sets a password for creating workspaces and copying the repository. You cannot set this password for a standard repository.
baseline	Sets a password for integrating a workspace into the system baseline in the current repository.
master	Sets a master password for the repository, which allows complete access to the repository.
workspace	Sets a password for accessing the current workspace.

The *new_password* argument specifies the new password for the repository or workspace. A legal password is a string of 7-bit ASCII characters, of any length with no spaces. In a secure repository, a non-null password is required. In a standard repository, you can remove a password by specifying a null string, as shown in the following example:

```
SetPassword workspace '' secretpassword
```

The *current_password* argument specifies the current password that is being replaced by the new password. If a password is being set for the first time, then this value is not required.

If you want to set the administrator password for a repository that is a standard repository, you need to convert the repository to a secure repository using the **rpcopy** command, as described in *Forte 4GL System Management Guide*.

SetPath

Set directory search path

The **SetPath** command sets the directory search path used by any of the commands that take a file name as an argument.

Syntax **SetPath** *directory_name* [**;** *directory_name...*]

Argument	Description
directory_name	The name of a directory (or set of directories) in which to look for files which are specified without a path.

The **SetPath** command resets the entire directory search path. To add more directories to a directory search path, you can use the **AddPath** command.

Most of the commands that have input file arguments, such as the **Include** command, allow you to specify the name of the input file without a full directory specification. In that case, the commands use the current directory search path, as defined by the **SetPath** and **AddPath** commands, to find the file. The directory search path provides a set of directories that are checked in turn until a file matching the name is found.

By default, the directory search path only includes the current working directory. The current working directory is always considered the last directory in the directory search path, even after you give the **SetPath** command. If you want the current working directory to be searched first, you can specify the current working directory first in the list of directories for the **SetPath** command.

Each *directory_name* is a full directory path name. By default, directories should be specified in the local operating system directory format. If you have previously invoked the **UsePortable** command, then you should specify the *directory_name* in Forte portable format, which is a UNIX-style directory format. To specify more than one directory, separate the directory names with semicolons.

You can embed environment variable names within the directory names, by using the syntax:

Syntax **$**{*environment_variable_name*}

The dollar sign and braces indicate that the name inside the braces is an environment variable, and the entire specification is replaced with the current setting of the environment variable.

You can also use the following specification to expand the environment variable name and convert it to a portable file format as well:

Syntax **%**{*environment_variable_name*}

The percent sign and braces indicate that the name inside the braces is an environment variable, and the entire specification is replaced with the current setting of the environment variable. This syntax is useful when you have directories specified in environment variables in local format and need to convert them to portable format.

The **SetPath** command and **SetSearchPath** command are synonyms.

Sample uses of the **SetPath** command are:

```
Fscript> SetPath /mydisk/mydir;${ENV_VAR}/subdir
Fscript> SetPath c:\mydir;${ENV_VAR}\subdir
Fscript> SetPath "Mac HD:Apps:TempFolder";${ENV_VAR}:Sub
Fscript> SetPath $dka0:[path];${ENV_VAR}:[otherdisk.otherdir]
Fscript> UsePortable
Fscript> SetPath %{FORTE_ROOT}/install/examples
```

SetPrefNode

Set preferred server node for partitioning

The **SetPrefNode** command designates the preferred server node to be used in a subsequent **Partition** command.

Syntax **SetPrefNode** *node_name*

Argument	Description
node_name	The name of a server node in the environment to use as the preferred server when partitioning.

The **SetPrefNode** command tells the partitioning system where unrestricted application partitions or library projects are to be placed when you invoke a **Partition** command. If you do not use the **SetPrefNode** command, the **Partition** command assigns partitions and library projects to default nodes.

The **SetPrefNode** command sets the preferred node only for the partitioning of a single library project or application in a single environment. This preferred node information is not valid for other library projects, applications, or environments.

SetProjRestricted

Setting Restricted property

The **SetProjRestricted** command defines the current project as a restricted project.

Syntax **SetProjRestricted** *restricted_flag*

Argument	Description
restricted_flag	Set to a value of **0** for unrestricted projects or a value of **1** for restricted projects.

Before you can use this command, you must set the current project, using either the **FindPlan** or **NewProj** command.

Restricted projects are projects that can be run only on certain nodes in the environment. These restricted projects include projects like C, DCE, or ObjectBroker projects that can only be installed on certain nodes in the environment.

If you are using the **ImportPlan** command to import a project definition statement, you can also set the restricted attribute on the **has properties** clause for the **begin TOOL** statement. The **begin TOOL** statement is described in *TOOL Reference Manual*.

Note If methods within the project perform an operation on classes that come from restricted supplier projects, including instantiating the classes, but you do not set the *restricted_flag* argument for a project to **1**, you will get an error when you try to compile the TOOL code.

To create a non-restricted project that references a class in a restricted project, you can create service objects that instantiate the classes and handle all interactions with classes of the restricted project. For more information about creating non-restricted projects that reference C, DCE, or ObjectBroker restricted projects, see *Integrating with External Systems*.

SetProjStart

Specify startup class and method

The **SetProjStart** command specifies the starting class and method for the current project that are used by subsequent **Run** or **RunDistrib** commands.

Syntax **SetProjStart** *class_name method_name*

Argument	Description
class_name	The name of a class in the current project. An object of this class is allocated when you invoke the **Run** or **RunDistrib** command, and the *method_name* is invoked on this object.
method_name	The name of a method defined for *class_name*. This method must already be defined for the class, and cannot have any parameters defined for it.

Before you can use the **SetProjStart** command, you must have a current project, set by invoking either the **FindPlan** or **NewProj** command.

When you use a **Run** or **RunDistrib** command, Forte looks up the class and method defined for the current project, instantiates an object of the starting class, and invokes the starting method on that object.

The method specified as the starting method cannot have parameters defined for it. If you try to specify a starting method that has parameters, you get an error. You should not later change the method to add parameters.

If you are using the **ImportPlan** command to import a project definition, you can also set the starting class and method for the project on the **has properties** clause for the **begin TOOL** statement. The **begin TOOL** statement is described in *TOOL Reference Manual*.

SetProjType

Specify project type

The **SetProjType** command defines the purpose of the current project as one of the following: the main project for a user application, the main project for a server application, or a library project.

Syntax **SetProjType** *type_flag*

Argument	Description
type_flag	The purpose of the current project, specified as **1** for user applications, **2** for server-only applications, and **3** for libraries.

This setting is used when you later partition an application or library project using the **Partition** command.

Before you can use the **SetProjType** command, you must have a current project, set by invoking either the **FindPlan** or **NewProj** command.

The *type_flag* argument values are described in the following table:

***type_flag* value**	**Description**
1	The default. The project is defined as the main project for a user application, which will have a client partition.
2	The project is defined as the main project for a server-only application, which will *not* have a client partition.
3	The project is defined as a library project.

You might want to designate a server-only project when your application contains only service objects that will be used by other applications. However, because you often need a client partition for testing the server, you can use a user application type project for creating and installing a service partition that is to be shared between applications. See the "ListServiceApps" on page 63 for a discussion of sharing service partitions between applications.

SetRepos

Designate current repository

The **SetRepos** command specifies the name of the current repository.

Syntax

SetRepos *repository_name*

Argument	Description
repository_name	The name of the repository, specified as a full repository specification. This repository can be a central repository, a private repository, or a shadow repository.

The **SetRepos** command overrides any setting for the repository from the **-fr** flag on the **fscript** command line. You can invoke the **SetRepos** command before the first **Open** command in Fscript. For more information about specifying a repository, see "Specifying a Repository" on page 18.

To open a session on another repository and workspace:

1. Close the current repository session with the **Close** command.
2. Change the current repository with the **SetRepos** command.
3. Change the current workspace with the **SetWorkspace** command.
4. Open the workspace with the **Open** command.

As an example, you could invoke the following set of commands to set and open a B-tree repository.

```
Fscript> Close
Fscript> SetRepos bt:myrepos
Fscript> SetWorkspace jimmy
Fscript> Open
```

SetSearchPath

Set directory search path

The **SetSearchPath** command sets the directory search path used by any of the commands that take a file name as an argument.

Syntax

SetSearchPath *directory_name* [**;** *directory_name...*]

Argument	Description
directory_name	The name of a directory (or set of directories) in which to look for files which are specified without a path.

The **SetSearchPath** command resets the entire directory search path.

By default, the directory search path only includes the current working directory. The current working directory is always considered the last directory in the directory search path, even after you give the **SetSearchPath** command. If you want the current working directory to be searched first, you can specify the current working directory first in the list of directories for the **SetSearchPath** command.

Each *directory_name* is a full directory path name. By default, directories should be specified in the local operating system directory format. If you have previously invoked the **UsePortable** command, then you should specify the *directory_name* in Forte portable format, which is a UNIX-style directory format. To specify more than one directory, separate the directory names with semicolons.

You can include environment variable names in the directory names. This task is described in "SetPath" on page 87.

The **SetPath** command and **SetSearchPath** command are synonyms.

SetServiceEOSInfo

Set service object to support external system

The **SetServiceEOSInfo** command tells Forte to generate the files that make the specified service object available to a specified external object service (EOS). Forte generates these files when you make the distribution, using either the **MakeAppDistrib** command, or by using the **Make Distribution** command in the Partition Workshop.

Syntax **SetServiceEOSInfo** *service_object_name EOS_Type* [*export_name*]

Argument	Description
service_object_name	The name of the service object that you want to make available to the external object service.
EOS_Type	The identifier for the type of external object service.
export_name	The name used by the external object service to identify this service object.

The *service_object_name* argument specifies the name of the service object that you want to make available to the external object service. If the current project contains the service object, you can specify just the name of the service object; otherwise, *service_object_name* should specify the project name and the service object name, as shown in the following example:

```
Fscript> SetServiceEOSInfo App1_Main.External_SO OBB App1Serv
```

In this example, App1_Main is the name of the project, and External_SO is the name of the service object.

The *EOS_Type* argument specifies the identifier for the type of external object service that you want to make the service object available to. You can specify the following values:

***EOS_Type* Value**	**Description**
DCE	OSF's Distributed Computing Environment
Enc	Transarc's Encina
OBB	ObjectBroker, produced by Digital Equipment Corporation
OLE	Microsoft's OLE
none	Cancels any previously specified external object service.

The *export_name* argument specifies the name to be used by the external object service to identify this service object. If no export name is specified, the default server name is the project name for the server, an underscore character (_), and the server object name.

For more information about making service objects available to DCE, Encina, ObjectBroker, or OLE, see *Integrating with External Systems*.

SetWorkspace

Designate current workspace

The **SetWorkspace** command sets the current workspace in the current repository that will be opened by the next **Open** command.

Syntax **SetWorkspace** *workspace_name*

Argument	Description
workspace_name	The workspace name to be used on the next **Open** command.

The **SetWorkspace** command overrides any setting for the workspace from the **-fw** flag on the **fscript** command line or from the FORTE_WORKSPACE environment variable. You can invoke the **SetRepos** command before the first **Open** command in Fscript.

To open another repository and workspace:

1. Invoke a **Close** command to close the current repository session.
2. Invoke a **SetRepos** command to change the current repository.
3. Invoke a **SetWorkspace** command to change the current workspace.
4. Invoke the **Open** command.

As an example, you could invoke the following set of commands to set and open a B-tree repository.

```
Fscript> Close
Fscript> SetRepos bt:myrepos
Fscript> SetWorkspace jimmy
Fscript> Open
```

Shell

Invoke OS commands

The **Shell** command starts a session where operating system commands can be invoked.

Syntax Shell

For example, in UNIX, you can use the **Shell** command to start a Bourne shell session.

If you are running Fscript on the OpenVMS platforms, you cannot use the **Shell** command.

You can control the type of session that is started by setting the UNIX SHELL environment variable.

ShowAlias

Show a list of available aliases

The **ShowAlias** command shows the *alias expansion*, or command string, that the specified alias directly maps to. An *alias* is a synonym you can make for an Fscript command and its arguments using the **AddAlias** command.

Syntax **ShowAlias** [*alias_name*]

Argument	Description
alias_name	The alias name whose expansion you want to see.

The *alias_name* argument specifies the alias whose expansion you want to see. If you do not specify *alias_name*, then **ShowAlias** lists all alias mappings defined for this session of Fscript.

ShowApp

Show information about the application configuration

The **ShowApp** command shows the current configuration information for the current application in the current environment.

Syntax **ShowApp**

The command gives the name of the main project and environment, and lists the set of partitions and node assignments for these partitions.

You create a current configuration for the current application by partitioning it in an environment using the **Partition** command. You can set the current application by setting the main project for the application as the current project using the **FindPlan** command.

ShowCompHistory

Show integration history of a component

The **ShowCompHistory** command prints information about past integrations of a component or set of components.

Syntax **ShowCompHistory** *comp_match_string* [*verbose_flag*] [*num_to_show*]

Argument	Description
comp_match_string	The name of a single component, or a prefix followed by an asterisk (for all components whose names start with a set of characters).
verbose_flag	Specifies whether to show limited information (set to **0**) or to show more detailed information (set to **1**).
num_to_show	The number of integrations to list, starting with the most recent. Set to **"*"** to show all integrations. The default is **1**.

Use the **ShowCompHistory** command to list the following information for the specified component or set of components:

- The workspace that has the component checked out.
- The version of the component in the current workspace.
- The times and workspace names that integrated the component into the system baseline. The argument *num_to_show* will control how many of these to list, starting with the most recent.
- If the *verbose_flag* is set to **1**, the comment from the integration will be listed as well.

Specify the *comp_match_string* argument as a prefix of characters followed by an asterisk to get a list of matching components. For example, the string "Test*" will list all components starting with the characters "Test" in either upper or lower case.

ShowEnv

Show environment information

The **ShowEnv** command provides information about the current environment, as set by the **FindEnv** or **FindActEnv** command.

Syntax ShowEnv

The **ShowEnv** command lists:

- basic environment properties
- loaded and installed applications in the environment
- nodes in the environment
- available external resource managers in the environment

ShowExpansions

Show alias expansions

The **ShowExpansions** command enables or disables the printing of *alias expansions*, the command strings that aliases map to, to standard output when they occur. An *alias* is a synonym you define for an Fscript command and its arguments using the **AddAlias** command.

Syntax **ShowExpansions** *show_flag*

Argument	Description
show_flag	The flag that determines the level of expansion printing

The *show_flag* argument specifies whether and how alias expansions are printed when they occur. You can specify the following values:

***show_flag* value**	**Description**
0	No alias expansions are printed.
1	The final alias expansions are printed.
2	Each step of the alias expansion is printed.

ShowIntegrations

Show workspace integration history

The **ShowIntegrations** command prints information about the integrations of a workspace or set of workspaces into the system baseline for the repository.

Syntax **ShowIntegrations** [*workspace_match_string*] [*num_to_show*]

Argument	**Description**
workspace_match_string	The name of a single workspace, or a prefix followed by an asterisk (for all workspaces whose names start with a set of characters). The default is "*****", which will show the integrations of all workspaces in the repository.
num_to_show	The number of integrations from the most recent to list. This can be set to "*****" to show all integrations. The default is **1**.

The **ShowIntegrations** command lists, for each workspace integration:

- the name of the workspace
- the time and date the workspace was integrated.
- the contents of the integration log, including a list of all components and plans that were integrated
- comments from the integration.

The *workspace_match_string* argument defines the list of workspaces to one or more. You can specify *workspace_match_string* as:

- An "*" to list the integrations of all workspaces in the environment. The default is to list integrations for all workspaces.
- A workspace name for listing the integrations of one workspace only.
- A prefix followed by an asterisk for listing the integrations of any workspaces with names starting with the prefix, as shown in the following example:

```
Fscript> ShowIntegrations MyWork*
```

The *num_to_show* argument limits the number of integrations in the list to the number specified, starting at the most recent integration. The default is to list the most recent integration only.

ShowLockedWorkspaces

Show locked workspaces in the repository

The **ShowLockedWorkspaces** command prints a list of the workspaces that are locked in the current repository.

Syntax ShowLockedWorkspaces

The **ShowLockedWorkspaces** command prints information about any workspaces, including:

- Workspaces that are currently open.

  This command also prints the name of the node on which a client application has the workspace open.

- Workspaces that have global versioning locks, which are taken when some large repository commands, such as **UpdateWorkspace** and **IntegrateWorkspace**, are invoked.

If locks are left on workspaces and must be removed, you can use the **UnlockWorkspace** or **ForceWorkspaceUnreserved** commands.

ShowPath

Show search path

The **ShowPath** command shows the current search path.

Syntax ShowPath

The **ShowPath** command displays the current directory search path for resolving file names in Fscript commands that take file name arguments.

You use the **SetPath**, **SetSearchPath** and **AddPath** commands to set up the directory search path.

ShowPlan

Show plan information

The **ShowPlan** command prints information about the current plan. A current plan can be a project, an application model, or a business model.

Syntax ShowPlan

Use the **FindPlan** command to set the current plan.

The **ShowPlan** command prints information about the current plan, as designated by the **FindPlan** command.

For projects, this information can include:

- plan name
- properties of the plan
- list of supplier plans
- set of components of the project
- set of configurations defined for the project

For application models and business models, this information can include:

- plan name
- properties of the plan
- list of supplier plans
- list of generated projects
- contents of the plan (classes, windows, services, and so on)

ShowPlanHistory

Show integration history of a plan

The **ShowPlanHistory** command prints information about past integrations of the current plan. The current plan must be an application model or business model. If the current plan is a project, you should use the **ShowCompHistory** command.

Syntax **ShowPlanHistory** [*verbose_flag*] [*num_to_show*]

Argument	Description
verbose_flag	Specifies whether to show limited information (set to **0**) or to show more detailed information (set to **1**).
num_to_show	The number of integrations to list, starting from the most recent integration. This can be set to **"*"** to show all integrations. The default is **1**.

Use the **ShowPlanHistory** command to list the following information for the current plan:

- The workspace that has the plan checked out.
- The version of the plan in the current workspace.
- The times and workspace names that integrated the plan into the system baseline. The argument *num_to_show* will control how many of these to list, counting back from the most recent.
- If the *verbose_flag* is set to **1**, the comment from the integration will be listed as well.

ShowReposInfo

Show repository information

The **ShowReposInfo** command prints information about the repository and current workspace.

Syntax ShowReposInfo

The **ShowReposInfo** command prints information about the current repository, including the name and some physical properties of the repository. These properties include:

- whether you are in a shadow repository or not, and the name of the central repository for a shadow repository
- for a shadow repository, a list of workspaces cached the list of workspaces in the shadow repository
- creation date of the repository
- creation date of the workspace
- date that the workspace was last updated

ShowWorkspace

Show current workspace name

The **ShowWorkspace** command prints the names of the current workspace and current repository.

Syntax ShowWorkspace

You can set the current repository using the **SetRepos** command. You can set the current workspace using the **SetWorkspace** command.

SilentOff

Turn off printing of exceptions

The **SilentOff** command disables the printing of exceptions to standard output on subsequent Fscript commands.

Syntax SilentOff

You can use this command to control exception printing in running scripts.

SilentOn

Turn on printing of exceptions

The **SilentOn** command enables the printing of exceptions to standard output on subsequent Fscript commands.

Syntax SilentOn

You can use this command to control exception printing in running scripts.

Step

Step through commands interactively

The **Step** command allows you to step through the commands interactively before they are executed while a script started by an **Include** command is running.

Syntax Step

The **Step** command is useful in troubleshooting files started by the **Include** command in Fscript. Once you invoke the **Step** command, Fscript will prompt you before invoking each command read from the script file.

StopRemoteParts

Stop remote partitions

The **StopRemoteParts** command stops any of the remote partitions that have been automatically started through the **RunDistrib** command.

Syntax StopRemoteParts

When you partition an application and execute the **RunDistrib** command to test the application configuration, Forte automatically starts up server partitions on remote server machines if they are needed by the application. After the **RunDistrib** command completes execution of the client portion of an application and returns to the Fscript command prompt, the remote partitions for the application are still running. The remote partitions do not automatically shut down when the client shuts down because the remote partitions are often set up to be shared among client partitions.

The remote partitions are automatically shut down only when one of the following happens:

- You close the repository session by invoking the **Close** command or by leaving Fscript using the **Exit**, **ExitIfNoEnvMgr**, or **Quit** commands.
- You switch to another project or configuration, by invoking the **FindPlan** or **FindEnv** commands.
- You invalidate the current configuration by changing the partitioning or application logic so that a new partition must replace the partition currently executing. The old remote partitions are shut down when you reinvoke the **RunDistrib** command.

You can use the **StopRemoteParts** command to explicitly stop the partitions, even if one of the situations described above does not occur. For example, you might want to recheck logic of the remote partition startup by reinvoking the **RunDistrib** command.

You can also use the **StopRemoteParts** command to release the shared configuration lock on the environment that is set by any of the partitioning manipulation commands, such as **Partition**, **ShowApp** or **RunDistrib**. If you keep this configuration lock, users of the Environment Console and **escript** cannot manage the environment. You can use the **StopRemoteParts** command to explicitly release the configuration lock.

TestApp

Test an application being executed remotely

The **TestApp** command requests that **Fscript** wait for a remote execution of an application, and then run the client portion of that application on this node.

Syntax

TestApp *project_name* [*environment_name*]

Argument	Description
project_name	The name of a main project of an application that is currently being partitioned in another session.
environment_name	The name of the environment in which the application is being partitioned.

The **TestApp** command requests that this Fscript session wait to be sent a client partition so that this Fscript session can participate in a shared test of the project.

The *project_name* and *environment_name* arguments are taken from the output of the **ListTestApps** and are used to identify which remote partitioning session you want to test.

Test client partitions of applications

The **TestApp** command, along with the **RunDistrib** and **ListTestApps** commands, let you set up tests of your distributed application with multiple client partitions running. For example, you might want to do this to stress test your application.

When you invoke the **TestApp** command, this Fscript session stops prompting for commands. This Fscript session notifies the session that is partitioning the application that this session is waiting to participate in the distributed test.

When the remote partitioning session starts the application using the Partition Workshop **Run** command or the Fscript **RunDistrib** command, the client portion of the application starts running in your Fscript session. You can interact with the client interface as though you started the application yourself. After you exit the client portion of the application, the Fscript session displays the Fscript prompt, and you can invoke further Fscript commands.

To test the client partition of an application using Fscript:

1. Start an Fscript session or a session of the Partition Workshop.
2. Partition the application you want to test.

   In the Fscript session, you need to open your workspace, make the main project for the application the current project using the **FindPlan** command, set the current environment using the **FindEnv** command, invoke the **Partition** command to generate the default configuration, and modify your configuration, as appropriate.

   For information about partitioning in the Partition Workshop to set up tests, see *A Guide to the Forte 4GL Workshops*.
3. Start an Fscript session on the client node.
4. Invoke the **ListTestApps** command on the client node to get a list of applications available for testing.
5. Invoke the **TestApp** command on the client node to request that Fscript wait to be sent a client partition to run. See "TestApp" on page 98 for details.

You can also test client partitions using the TestClient utility. This utility is explained in *A Guide to the Forte 4GL Workshops*.

UnassignAppComp

Unassign an application component

The **UnassignAppComp** command unassigns the specified application component—a partition or a library project—from the specified node in the current configuration.

Syntax **UnassignAppComp** *node_name* [*component_name*]

Argument	Description
node_name	The name of the node for which the component is to be unassigned.
component_name	The name of a component in the current application.

Use the **UnassignAppComp** command to remove a component from future installation on a specific node in the environment.

The *node_name* must be a valid node defined in the environment, and the component must be assigned to it, either through the default partitioning, or by an explicit **AssignAppComp** command.

The optional *component_name* is the name of the component in the current configuration of the current project, which is to be unassigned to the node. You can specify the unique trailer portion of the name only, such as “client”, to identify the component. If no *component_name* is given, then the current component is assigned. Use the **FindAppComp** command to designate a current component.

See “AssignAppComp” on page 36 for more information on assigning components to nodes.

UndoBranchComp

Undo changes made in branch

The **UndoBranchComp** command removes any changes made in a branched component, and reverts the component to its state as of the last **UpdateWorkspace** command.

Syntax **UndoBranchComp** *component_name*

Argument	Description
component_name	The name of a component in the current project which has been branched.

If you branched a component in the current project using the **BranchComp** command, you can discard the branch by invoking the **UndoBranchComp** command. Because you cannot integrate a workspace that has branches in it, you must either invoke the **UndoBranchComp** command to delete any branches, or convert the branches to checkouts using the **CheckoutComp** command before integrating.

You can invoke the **UndoBranchComp** command on a shadow repository only while the shadow repository is attached to the central repository.

You can undo all branches and checkouts for a project with the **RevertProj** command, described in “RevertProj” on page 78.

UndoBranchPlan

Undo changes made in branch

The **UndoBranchPlan** command removes any changes made in the current plan if it is branched and a business model or an application model. This command reverts the plan to its state as of the last **UpdateWorkspace** command. You cannot use this command if the current plan is a project.

Syntax UndoBranchPlan

If you branched the current plan using the **BranchPlan** command, you can discard the branch by invoking the **UndoBranchPlan** command. Because you cannot integrate a workspace that has branches in it, you must either invoke the **UndoBranchPlan** command to delete any branches, or convert the branches to checkouts using the **CheckoutPlan** command before integrating.

You can invoke the **UndoBranchPlan** command on a shadow repository only while the shadow repository is attached to the central repository.

UndoCheckoutComp

Undo changes made in check out

The **UndoCheckoutComp** command removes any changes made to a checked-out component, and reverts the component to its state as of the last **UpdateWorkspace** command. It also releases the exclusive checkout lock on the component.

Syntax **UndoCheckoutComp** *component_name*

Argument	Description
component_name	The name of a component in the current project which has been checked out.

The **UndoCheckoutComp** command removes the exclusive checkout lock on the component, which lets other workspaces check out the component. The **UndoCheckoutComp** command also discards any unintegrated changes you made to the checked-out version of the component. The component is left in your workspace as a read-only component.

After you invoke the **UndoCheckoutComp** command, you should invoke a **Commit** command to make the component available to other workspaces for check out.

You can invoke the **UndoCheckoutComp** command on a shadow repository only while the shadow repository is attached to the central repository.

You can undo all branches and checkouts for a project with the **RevertProj** command, described in “RevertProj” on page 78.

UndoCheckoutPlan

Undo changes made in check out

The **UndoCheckoutPlan** command removes any changes made to the current plan if it is checked-out and a business model or application model. This command reverts the plan to its state as of the last **UpdateWorkspace** command and releases the exclusive checkout lock on the component. If the current plan is a project, use the **UndoCheckoutComp** command.

Syntax UndoCheckoutPlan

The **UndoCheckoutPlan** command removes the exclusive checkout lock on the plan, which lets other workspaces check out the plan. The **UndoCheckoutPlan** command also discards any unintegrated changes you made to the checked-out version of the plan. The plan is left in your workspace as a read-only component.

After you invoke the **UndoCheckoutPlan** command, you should invoke a **Commit** command to make the plan available to other workspaces for check out.

You can invoke the **UndoCheckoutPlan** command on a shadow repository only while the shadow repository is attached to the central repository.

UndoRemoveComp

Retrieve component that were removed

The **UndoRemoveComp** command retrieves a project component that was removed from the workspace using the **RemoveComp** command.

Syntax **UndoRemoveComp** *component_name*

Argument	Description
component_name	The name of a component in the current project which has been removed using the **RemoveComp** command.

If you have removed a checked-out component in the current project using the **RemoveComp** command, you can undo the effects of the **RemoveComp** command by invoking the **UndoRemoveComp** command.

You can use the **UndoRemoveComp** command only until you integrate the workspace using the **IntegrateWorkspace** command or the Repository Workshop. After you integrate your workspace, the component is permanently removed from the workspace and the system baseline, and you can no longer use the **UndoRemoveComp** command to retrieve the component.

If a component was newly created in your workspace but has not been integrated, then you cannot recover the deleted component using the **UndoRemoveComp** command. The **UndoRemoveComp** command does not recover any unintegrated changes you made to the component before you removed it.

The **UndoRemoveComp** command removes the exclusive checkout lock on the component, which lets other workspaces check out the component. The component is left in your workspace as a read-only component.

You can invoke the **UndoRemoveComp** command on a shadow repository only while the shadow repository is attached to the central repository and while this shadow has no uncommitted changes.

UnlockWorkspace

Remove repository lock on workspace

The **UnlockWorkspace** command unlocks a workspace that has been left locked by unusual error conditions.

Syntax **UnlockWorkspace** *workspace_name* [*workspace_password*]

Argument	Description
workspace_name	The name of the workspace to unlock or unreserve.
workspace_password	If the workspace has a password, you must specify the password in the **UnlockWorkspace** command.

The repository software normally removes locks, so you rarely need this command. However, in certain error conditions, it is necessary to explicitly unlock workspaces.

Caution You must be very careful when using the **UnlockWorkspace** command to ensure that you are correcting one of the situations described. You can lose work in your repository if you use the **UnlockWorkspace** command incorrectly.

To see a list of workspaces that are currently locked, invoke the **ShowLockedWorkspaces** command.

Normally, workspace locks are released whenever the sessions close the repository, even if they abort abnormally. However, because the central repository servers are not always notified by network software when client nodes are unexpectedly rebooted, the locks on workspaces can be left in the repository for periods of time, even though the session that presumably has the lock has gone away. You can use the **UnlockWorkspace** command to

unlock the lock on a workspace that has been “orphaned” in this way. System managers can also unlock workspaces using Escript and Environment Console commands, as described in *Escript and System Agent Reference Manual*.

If you need to release a reserved workspace in the central repository because a shadow repository was corrupted, use the **ForceWorkspaceUnreserved** command. This command is explained in “ForceWorkspaceUnreserved” on page 54.

UpdateWorkspace

Update workspace with integrated changes

The **UpdateWorkspace** command updates the contents of the current workspace with any changes integrated into the system baseline since the last time you used the **UpdateWorkspace** command for this workspace.

Syntax **UpdateWorkspace** [*logfile_name*]

Argument	Description
logfile_name	The name of a file to use for logging the results of the **UpdateWorkspace** operation. Using the **UpdateWorkspace** command can be time consuming, so you can specify the name of the file to later examine the resulting status messages. If this argument is not specified, the results are listed in the **Fscript** trace output.

You should use the **UpdateWorkspace** command regularly to add changes that have been made to the system baseline since you first created your workspace. The **UpdateWorkspace** command only copies changes made to plans that are included in your workspace.

Use **Commit** before **UpdateWorkspace**

Before you can use the **UpdateWorkspace** command, you must commit all outstanding changes in the session using the **Commit** command**.**

If you have branches that might be overwritten by the **UpdateWorkspace** command, you need to remove the branches using the **UndoBranchComp** command. When you invoke the **UpdateWorkspace** command, you will get errors if you have branched components in your workspace that would be overwritten by the **UpdateWorkspace** command. The Repository Workshop provides additional capabilities for merging these branch conflicts. See *A Guide to the Forte 4GL Workshops* for details.

Recovery from updating failures

If the update to the workspace fails, any changes that have been made to the workspace are backed out, and the workspace is in the same state as before the **UpdateWorkspace** command was invoked. The global repository lock, which prevents other developers from updating or integrating their workspaces, is automatically removed as part of the recovery process.

If the **UpdateWorkspace** command fails because the repository client session that invoked the command fails, the repository server might not release the global repository lock held by the client session. You might need to use the Fscript **UnlockWorkspace** command to notify the repository server that the client session is no longer active. The **UnlockWorkspace** command, described in “UnlockWorkspace” on page 101, unlocks both the workspace and any global repository locks held by that workspace.

UseLocal

Use local file naming for Fscript commands

The **UseLocal** command specifies that path names in Fscript commands use local operating system file name format.

Syntax UseLocal

The **UseLocal** command tells Fscript to assume that the file names given for Fscript commands are specified in local operating system format instead of the portable file naming format. The local operating system is determined by the machine where you start the Fscript command.

When Fscript starts, the default is to use the local operating system file naming format for file name arguments. However, if you invoke a **UsePortable** command during the Fscript session, you must use the **UseLocal** command to revert to the default naming conventions.

The following table shows how you enter local file formats for each operating system:

File format	Examples

UNIX	`Fscript> UseLocal`
	`Fscript> Include /mydisk/mydir/myfile.inc`
	`Fscript> ListFile`
	`${FORTE_ROOT}/install/examples/tstapps.fsc`

Windows	`Fscript> UseLocal`
	`Fscript> Include c:\mydir\myfile.inc`
	`Fscript> ListFile`
	`${FORTE_ROOT}\install\examples\tstapps.fsc`

VMS	`Fscript> UseLocal`
	`Fscript> Include $dska:[mydir]myfile.inc`
	`Fscript> ListFile`
	`${FORTE_ROOT}:[install.examples]tstapps.fsc`

UsePortable

Use portable file naming for Fscript

The **UsePortable** command specifies that path names in Fscript commands use Forte portable file name format.

Syntax UsePortable

The **UsePortable** command tells Fscript to assume that the file names given for Fscript commands are specified in portable file naming format instead of the local operating system format. You can write portable scripts of Fscript commands when you use the portable file name format.

When Fscript starts, the default is to use the local operating system file naming format for file name arguments. Use the **UsePortable** command to tell Fscript to expect the portable file format.

Forte portable file naming uses the following conventions:

- Directory paths are specified in UNIX format, with slashes to represent the directory hierarchy.
- Directory names can be up to eight characters long.
- File names can be up to 8 characters long, and are followed by an extension up to 3 characters long. Certain extensions support conventions for file types, which are described in the Forte online Help.
- The special syntax "%{*environment_variable_name*}" can be embedded in a portable file name. This syntax looks for an environment variable with the given name, and assumes that the environment variable represents a directory name in local operating system format. However, it expands it within a portable file name as if it were specified in portable format. This provides a good way to provide local "roots" for directories of files, and then use portable format underneath that root to the tree.

The following example shows how you can use **UsePortable** and portable file format:

```
Fscript> UsePortable
Fscript> Include /mydisk/mydir/myfile.inc
Fscript> Include %{FORTE_ROOT}/install/examples/tstapps.fsc
```

UseServiceFromApp

Defining a reference partition

The **UseServiceFromApp** command defines a reference partition that references a service object that has been installed as a shared service in the current environment.

Syntax **UseServiceFromApp** *service_object_name application_name*

Argument	Description
service_object_name	The name of a service object in a project in the workspace.
application_name	The name of an application that has been installed (or at least has had an application distribution made for it), which contains the same service object.

The **UseServiceFromApp** command defines a reference partition so that a service object in an installed application in the environment can be used in the current application. This allows you to share partitions across installed applications.

The *service_object_name* specifies the name of the service object to be shared, and the *application_name* specifies the application that provides this service object.

You can use the **ListServiceApps** command to list instances of a service object in the current environment. You can then invoke the **UseServiceFromApp** command to define one of the installed service objects as a reference partition for the specified service object. When you later make a distribution for the current application and install the distribution in the environment, the current application will use the installed service object instead of creating and using a new service object.

For more information about making reference partitions, see *A Guide to the Forte 4GL Workshops*.

ValidatePlan

Check a business or application model for errors

The **ValidatePlan** command checks the current plan—a business model or application model—for errors without generating application code. This command does not work for projects.

Syntax ValidatePlan

If you want to check your model for errors without going through a lengthy code generation process, use **ValidatePlan** instead of **CompilePlan**.

Vi

Edit a file

The **Vi** command starts an editor so that you can edit the specified file.

Syntax **Vi** [*file_name*]

Argument	Description
file_name	The name of the file you want to edit.

The *file_name* argument, given without a directory path, specifies the file that you want to edit. Specify the file name relative to the current working directory. By default, specify the *file_name* argument in local operating system format. If you invoke the **UsePortable** command, however, the file name must be specified in portable file format.

If you do not specify the *file_name* argument, then the editor starts a new file.

You can define what editor is used with the **Vi** command by setting the FORTE_EDITOR environment variable.

WhichFile

Search for file in directory path

The **WhichFile** command searches through the directories in the current directory search path to locate the first directory in which the specified file exists.

Syntax **WhichFile** *file_name*

Argument	Description
file_name	The simple name of a file to locate.

You can invoke the **WhichFile** command to search each of the directories in the current directory search path to see where a specific file is located. The current directory search path is defined using the **SetPath** and **AddPath** commands. You can use the **ShowPath** command to display the current directory search path.

The *file_name* argument is the name of a file, given without a directory path. Each directory in the current directory search path is checked in turn to see if it contains the named file. When a match is found, the directory name that contained the file is displayed.

Appendix A

Fscript Command Summary

This appendix contains a summary list of all Fscript commands, with their arguments and a brief description of the purpose of each command.

Fscript Command Summary

Command	Description	See...
AddAlias *alias_name command_string*	Define an alias for an Fscript command and its arguments.	page 34
AddPath *directory_name* [**;** *directory_name*...]	Add a new directory path to the directory path Fscript commands use to seek files. (See also **SetPath**).	page 34
AddProjToLib *project_name*	Add a project to the current library configuration.	page 35
AddSupplierPlan *plan_name*	Include a plan as a supplier plan to the current plan.	page 36
AssignAppComp *node_name* [*partition_name*]	Assign a partition or a library to a specific node.	page 36
AttachToCentral	Attach a shadow repository to the central repository.	page 37
BackupRepos *directory_name*	Backup the current repository to another directory.	page 38
BranchAllComps	Branch all components in the current project to the workspace.	page 38
BranchComp *component_name*	Branch a read-only or checked out component in the workspace.	page 38
BranchPlan	Branch the current business model or application model.	page 39
Cd *directory_name*	Change the current working directory to the specified directory.	page 39
CheckoutAllComps [*available_flag*]	Check out all components in the current project to the workspace.	page 39
CheckoutComp *component_name*	Check out a read-only or branched component in the workspace.	page 40
CheckoutPlan	Check out the current business model or application model.	page 40
Chmod *mode file_name*	Change the permissions of the specified file.	page 41
Close	Close the current workspace.	page 41
CollectMem	Run the memory garbage collector.	page 41
CommentOff	Turn off recording of comments to standard output.	page 42
CommentOn	Turn on recording of comments to standard output.	page 42
Commit	Commit changes to current workspace.	page 42
Compile *file_name*	Compile a file containing project component definitions.	page 42
CompilePlan [*force_flag*]	Compile all out-of-date components in a plan.	page 43
CompileWorkspace [*projects-only*] [*force*]	Compiles all plans in the current workspace.	page 44
CopyFile *file1_name file2_name* [**r**]	Copy a file.	page 44

Command	Description	See...
Cp *file1_name file2_name* [**r**]	Copy a file.	page 44
Delay *milliseconds*	Delay the current task for the specified number of milliseconds.	page 45
DetachFromCentral	Detach a shadow repository from the central repository, reserving the current workspace.	page 45
Directory [*directory_name*]	List files in a directory.	page 46
DisableAppComp *node_name* [*partition_name*]	Disable autostart for a partition on a node.	page 46
Duplicate *file1_name file2_name* **[r]**	Copy a file.	page 47
EnableAppComp *node_name* [*partition_name*]	Enable autostart for a partition on a node.	page 47
ExcludePlan *plan_name*	Exclude a plan from the current workspace, which removes it completely if it has never been integrated.	page 48
ExecCmd *opsys_cmd* [*bg_flag*] [*in_file*] [*out_file*] [*err_file*]	Execute an operating system command.	page 48
Exit	Leave Fscript.	page 49
ExitIfNoEnvMgr	Leave Fscript once no environment manager is running.	page 49
ExitStatus *integer*	Set a value to be returned to the routine that started Fscript.	page 49
ExportClass *class_name file_name* [**noids | ids**]	Export the named class to a text file.	page 50
ExportPlan *file_name* [**noids | ids**]	Export all components in a plan to a file.	page 50
ExportTemplate *file_name*	Export the current project as a Forte Express template file.	page 51
ExportWindowClass *class_name class_file window_file* [**noids | ids**]	Export a UserWindow subclass to two files, one for the class and one for the window definition.	page 51
ExportWorkspace *file_name* [**noids | ids**]	Writes the definition of all plans in the current workspace to a text file.	page 52
FindActEnv	Set the current environment to the active environment.	page 52
FindAppComp *component_name*	Set the current component for the current configuration.	page 53
FindEnv *environment_name*	Set the current environment.	page 53
FindPlan *plan_name*	Set the current plan.	page 53
ForceWorkspaceUnreserved *workspace* [*workspace_password*]	Forces a workspace to become unreserved.	page 54
Help [*command_name* | *match_string**]	List help on Fscript commands.	page 54

Command	Description	See...
ImportClass *file_name*	Compile a file containing project component definitions.	page 55
ImportPlan *file_name* [*merge*]	Import the plan from a file.	page 56
ImportWorkspace *file_name* [*merge*]	Import one or more plans belonging to a workspace from a file.	page 57
Include *file_name*	Execute a set of Fscript commands from another file.	page 58
IncludePublicPlan *plan_name*	Include a publicly-available plan in the current workspace.	page 58
IncreaseCompatLevel	Increase the compatibility level for a project.	page 59
IntegrateWorkspace *comment* [*logfile_name*] *[baseline_password]*	Integrate the changes in this workspace into the shared repository.	page 59
ListChangesInWorkspace [*filter_type*]	List all project components and plans in the workspace whose status matches the specified filter.	page 60
ListComps *comp_match_string* [*filter_type*] [*proj_match_string*]	List project components in the workspace whose version state matches the specified filter.	page 61
ListEnvs	List the environments in the workspace.	page 61
ListFile *file_name*	List contents of a file in output.	page 62
ListFiles [*directory_name*]	List files in a directory.	page 62
ListPlans [*match_string*] | *]	List the plans in the current workspace.	page 62
ListPublicPlans *match_string* [*show_unintegrated*] [*show_internal*]	List the publicly available plans in the repository.	page 63
ListServiceApps *service_object_name*	List installed applications containing the same referenced service object.	page 63
ListTestApps [*use_cache_flag*]	List applications which are currently being partitioned for use in shared client testing.	page 64
ListWorkspaces [*verbose_flag*]	List the workspaces in the repository.	page 65
Ls [*directory_name*]	List files in a directory.	page 65
MakeAppDistrib [*remake_flag*] [*node_name*] [*auto_compile*] [*install*]	Create a distribution for the current configuration.	page 65
MemStats	Print memory statistics to standard output.	page 66
MkDir *directory_name*	Create a directory.	page 68
ModLogger + (*logger_flags*) | - (*logger_flags)*	Add or remove logger flags.	page 68

Command	Description	See...
MoveServiceToPart *service_object_name* [*partition_name*]	Move a service object in current configuration to another partition.	page 68
Mv *old_file_name new_file_name*	Rename a file.	page 69
NewPart *service_object_name*	Create a new partition, with the specified service object.	page 69
NewPlan *plan_type plan_name*	Create a new plan in the current workspace.	page 70
NewProj *project_name*	Create a new project in the current workspace.	page 70
NewWorkspace *workspace_name* [*initial_password* [*admin_password*]]	Create a new workspace in repository.	page 71
Open [**readonly**|**readwrite**|**exclusive**|**exclusivewriter**] [*workspace_password*]	Open the workspace specified on the last **SetWorkspace** command.	page 72
Partition [1 | 3]	Partition using the current project and the current environment.	page 73
PrintEnv [*variable_name*]	Print the current value of one or all environment variables.	page 73
Pwd	Print the current working directory.	page 74
Quit	Exit Fscript.	page 74
ReadIntoFile *file_name* [*term_str*]	Read lines of data following this command and write the data to the specified file.	page 74
RemoveAlias [*alias_name*]	Remove an alias.	page 74
RemoveConf [*configuration_name*]	Remove a configuration for the current project.	page 75
RemoveComp *component_name*	Remove the named component from the current project.	page 75
RemoveFile *file_name*	Remove the specified file.	page 75
RemoveProjFromLib *project_name*	Remove a project from the current library configuration.	page 76
RemovePublicPlan *plan_name*	Remove the publicly-available plan completely from the repository.	page 76
RemoveSupplierPlan *plan_name*	Remove a supplier plan from the current plan.	page 76
RemoveWorkspace *workspace_name* [*workspace_password*]	Remove a workspace from the repository.	page 77
RenameComp *old_name new_name*	Rename a component in the current project.	page 77
Repeat *rep_count*	Repeats the next command the specified number of times.	page 78

Command	Description	See...
RevertProj	Undo all changes made to a project since last integration.	page 78
Rm *file_name*	Remove the specified file.	page 78
Run [*arguments*]	Run the project in test mode from the starting class and method.	page 78
RunDistrib [*arguments*]	Run the current project in distributed mode for the current configuration.	page 79
RunFile *file_name*	Run the fragment of TOOL code that is in the file.	page 80
Save	Save changes in the current workspace to the shadow repository.	page 80
Script *file_name*	Capture Fscript commands into a file.	page 81
SearchFile *file_name searchText* [*replaceText*] [*showLineNumbers*]	Locate text in a file.	page 81
SetAppCompCompiled *node_name compiled_flag* [*component_name*]	Define a partition or library as compiled on a specific node.	page 82
SetAppletFlag **0** | **1**	Defines the client application associated with a logical client partition as an applet.	page 82
SetDefault *directory_name*	Change the current working directory to the specified directory.	page 83
SetEnv *variable_name* [*value*]	Set an environment variable to the specified value.	page 83
SetOutFile [*file_name*]	Set the file where standard output is printed.	page 84
SetPartArgs *node_name arguments* [*partition_name*]	Set the startup arguments for a partition on a node.	page 84
SetPartRepCount *node_name replication_count* [*partition_name*]	Set the autostart replication count for a load balanced or failover server on a node.	page 85
SetPassword *password_type new_password* [*current_password*]	Set the password for the current repository.	page 86
SetPath *directory_name* [**;** *directory_name*...]	Set the directory path for Fscript commands to seek files.	page 87
SetPrefNode *node_name*	Set the preferred server node for subsequent partitioning of a project in a configuration.	page 88
SetProjRestricted *restricted_flag*	Set the restricted property for a project.	page 88
SetProjStart *class_name method_name*	Set the starting class and method for the current project.	page 88
SetProjType *type_flag*	Set project as user application (1), server only (2), or library (3).	page 89

Command	Description	See...
SetRepos *repository_name*	Set the name of repository for next **Open.**	page 90
SetSearchPath *directory_name* [; *directory_name*...]	Set the directory path for Fscript commands to seek files.	page 90
SetServiceEOSInfo *service_object_name EOS_Type* [*export_name*]	Set the type of external object service a service object is available to and its export name.	page 91
SetWorkspace *workspace_name*	Set the name of the workspace for next **Open.**	page 91
Shell	Starts a session in which operating system commands can be invoked.	page 92
ShowAlias [*alias_name*]	Display one or all defined aliases with their expansions.	page 92
ShowApp	Show information about the current application configuration.	page 92
ShowCompHistory *comp_match_string* [*verbose_flag*] [*num_to_show*]	Show the integration history of a component, or set of matching components.	page 93
ShowEnv	Show information about the current environment.	page 93
ShowExpansions *show_flag*	Enable or disable the printing of alias expansions to standard output when they occur.	page 93
ShowIntegrations [*workspace_match_string*] [*num_to_show*]	Show information about integrations of workspaces.	page 94
ShowLockedWorkspaces	Show a list of workspaces that have locks on them, and show the type of lock.	page 95
ShowPath	Shows the directory paths Fscript commands use to seek files. (See also **SetPath**).	page 95
ShowPlan	Display information about the current plan.	page 95
ShowPlanHistory [*verbose_flag*] [*num_to_show*]	Print information about past integrations of the current plan.	page 96
ShowReposInfo	Show physical information about the repository.	page 96
ShowWorkspace	Show the name of the current workspace and repository.	page 96
SilentOff	Turn off the printing of exceptions to standard output.	page 97
SilentOn	Turn on the printing of exceptions to standard input.	page 97
Step	Step through commands in an include file.	page 97
StopRemoteParts	Stop remote partitions which have been launched by the **RunDistrib** command.	page 97
TestApp *project_name* [*environment_name*]	Start a wait for testing the client portion of a remotely started application.	page 98
UnassignAppComp *node_name* [*component_name*]	Remove the assignment of a partition or library on a node.	page 99
UndoBranchComp *component_name*	Undo the branch for a component, and revert to previous state.	page 99
UndoBranchPlan	Undo the branch for the current business model or application model, and revert to its previous state.	page 99

Command	Description	See...
UndoCheckoutComp *component_name*	Undo the checkout for a component, and revert to previous state.	page 100
UndoCheckoutPlan	Undo the checkout for the current business model or application model, and revert to its previous state.	page 100
UndoRemoveComp *component_name*	Undo the effects of a **RemoveComp** on a component and revert to previous state.	page 101
UnlockWorkspace *workspace_name* [*workspace_password*]	Force an unlock of a workspace in the repository.	page 101
UpdateWorkspace [*logfile_name*]	Update the current workspace with any changes in the system baseline since the last update.	page 102
UseLocal	Set Fscript to recognize file name input in local name format.	page 102
UsePortable	Set Fscript to recognize file name input in portable format.	page 103
UseServiceFromApp *service_object_name application_name*	Reference the named service object from the named application.	page 104
ValidatePlan	Check the business model or application model for errors without generating application code.	page 104
Vi [*file_name*]	Check the business model or application model for errors without generating application code.	page 104
WhichFile *file_name*	Find a file in the current directory search path.	page 105

Appendix B

Memory and Logger Flags

This appendix contains a detailed description of how to use the memory and logger flags.

You can specify these flags using the -fl and -fm flags with the **fscript** command. You can also specify these flags using the **SetPartArgs** command. You can also set the logger flags using the **ModLogger** command.

-Fl Flag (Forte Logger)

The **-fl** flog allows you to specify logger flags to be used for the command. The logger flags set the file or files used by the LogMgr object for logging messages, and specify the types of messages logged in each file. See LogMgr class in the Forte online Help for information on how to produce the actual messages.

The **-fl** flag overrides the setting of the FORTE_LOGGER_SETUP environment variable.

Syntax **-fl** *file_name*(*file_filter*)[*file_name*(*file_filter*)...]

For UNIX and VMS, any arguments that contain parentheses must be enclosed by double quotes.

The following sections provide information specifying the file name and file filters.

File Name

The log file name is any valid file name where you want to log messages. The special file names "%stdout" and "%stderr" log the messages to standard output or standard error, respectively. In interactive Windows 3.0 or Macintosh applications, standard output and standard error create a window for display.

You can specify several files for logging messages. Multiple logging files are useful, for example, in an application where you want to display general tracing on standard output (%stdout), but want detailed tracing logged to a file for later review.

On Windows only, you can use the name "%stdwin" to create a simple, scrollable output window for textual output. "%stdwin" is particularly useful to specify an alternative file for the output from Fscript or the Forte Workshops.

File Filter

Each file name is associated with a *file filter*.

Syntax *message_type*[:*service_type*[:*group_number*[:*level_number*]]]

A description of each file filter option follows.

Message Type Option

The most general filter is message type. The value of message type differentiates messages such as errors, debugging information, or performance data. The message types appear in the following table. Each type is paired with a runtime LogMgr constant that corresponds to the message type when used with the more complex versions of the Put and PutLine methods:

Type	Meaning	Put or PutLine Constant
err	Error Messages	SP_MT_ERROR
sec	Security messages	SP_MT_SECURITY
aud	Audit messages	SP_MT_AUDIT
prf	Performance information	SP_MT_PERFORMANCE
cfg	Configuration modification	SP_MT_CONFIGURATION
trc	Debugging Information	SP_MT_DEBUG
*	All of the above	Any of the above

By using the message type categories, you can print different types of messages to different files. For example, you may want to print trace messages on standard output, error messages on standard output and an error log file, and performance information in a performance log file. The specification for this setup might be the following:

```
%stdout(trc:user err:user) err.log(err:user) perf.log(prf:user)
```

Service Type Option

Within message types there are service types. Service types are the large subdivisions you make within your program and typically map to projects. The service type parameter is optional. If used, the service type value must be between “user1” and “user10”. Typically, a service is a large portion of your application, such as inventory control, accounts receivable, or employee administration.

The LogMgr constant that corresponds to the service types “user1” through “user10” is SP_ST_USER1 through SP_ST_USER10. You can use these constants with the advanced version of the Put or PutLine methods or with the Test method. For convenience, you can use the name “user” or the asterisk symbol (*) to specify all user service types. Previous examples used the specification “user” without a trailing digit to indicate all user services.

For example, if you want all tracing to go to standard output, but tracing from service types “user1” and “user3” to be logged in a special file as well, you would use the following specification:

```
%stdout(trc:user) trc1_3.log(trc:user1 trc:user3)
```

Group Number Option

Within a service type there are group numbers. Group numbers are smaller subdivisions you make within a particular service and typically map to a group of related facilities. The optional group number provides further filtering within the service. A group number is between 1 and 63 inclusive.

For example, within a particular service (say, “user3”) you may have subdivided the modules into groups (for example, “transactions in progress”, “queued work lists”, and “problem reports”). Each module is large enough to warrant a group number within the service. “Transactions in progress” may be group number 2, whereas “problem reports” may be the group number 4. The following specification puts performance information from group number 2 into one file and trace information from group number 4 into another file:

```
xactprog.prf(prf:user3:2) probrep.trc(trc:user3:4)
```

The group number you specify in a Put method may be a constant that you defined to be equivalent to the numeric literal that you specified in FORTE_LOGGER_SETUP. For example, even though the literal 2 indicates the “transaction in progress” group, your specification to print the related performance information may be the following:

```
task.Part.LogMgr.PutLine(SP_MT_PERFORMANCE,
       SP_ST_USER3,TRANSACT_IN_PROGRESS,1,perfTextData);
```

This code assumes the value of the TOOL constant TRANSACT_IN_PROGRESS is 2.

You can also specify a range of group numbers using the syntax *group#-group#*. In the previous example, if you want trace information from groups 2 through 4 to go to a specific file, you would use the following statement:

```
some_trc.log(trc:user3:2-4)
```

Level Number Option

Within a group there are level numbers that you use to specify particularly detailed levels of information. The greater the level number value, the more detailed the information. The optional level number indicates the detail level of the information printed. Level numbers must be from 1 to 255 inclusive.

As with group numbers, the level number is determined by the application. Typically, developers use level numbers to filter out trace messages. Using the current example, the specification%stdout(trc:user3:2:1) indicates that all level 1 trace data from the “transaction in progress” (group 2) module of the “user3” service should be printed to standard output. Levels greater than 1 do not print. Thus, the following fragment prints only one line:

```
log: LogMgr = task.Part.LogMgr;
-- Printed (level <= 1)
   log.Put(SP_MT_DEBUG, SP_MT_USER3, TRANSACT_IN_PROGRESS, 1,
          'Browsing account # ');
   log.PutLine(SP_MT_DEBUG, SP_MT_USER3, TRANSACT_IN_PROGRESS,
            1,acc.Number);
-- Not printed (level > 1)
   log.Put(SP_MT_DEBUG, SP_MT_USER3, TRANSACT_IN_PROGRESS,
            2,acc.Owner);
   log.Put(SP_MT_DEBUG, SP_MT_USER3, TRANSACT_IN_PROGRESS,
            2,acc.LastChangeDate);
```

-Fm Flag (Memory Manager)

The **-fm** flag allows you to control the space used by the Forte memory manager.

If you do not set the memory flags, Forte uses defaults appropriate for the operating system. On the Macintosh Forte uses the settings in the Memory Requirements dialog. To open this dialog, use the **Get Info** command on the File menu for the ftclntws executor in the Forte:install:bin folder.

Note that you can change the memory configurations for a running application using the Environment Console and instruments defined on the OperatingSystem agent. See *Forte 4GL System Management Guide* for information.

Syntax **-fm**(*memory_option* {: | =} *number* [, *memory_option* {: | =} *number*])

To make this flag portable across the platforms supported by Forte, do not include any spaces in this argument, and do not enclose any part of the argument in single quotes.

For UNIX and VMS, any arguments that contain parentheses must be enclosed by double quotes, as shown in the following example:

```
"-fm(n:4000,x:8000)"
```

In UNIX, if you include spaces in this argument, you need to enclose the values, including the parentheses, in single quotes. You do not need to use single quotes for any other platform. The following table describes the memory options. For options that refer to "pages," a page is 1024 bytes of memory.

Memory Option	Description
c	Specifies when the memory pool should be contracted. The value represents the percentage utilization of the active pages that will trigger a memory pool contraction. Range is 0 to 100. The default value is 80. This option is valid only for Windows 95.
d	Sets the level of debugging information that is provided. The value is interpreted as a bit-mask of enabled options. The default is 0. The options are: 1—Verify memory before every collect. This checks that all of the memory manager's data structures are correct, that all pages containing user objects are correct, and that all pointers point to something legal. 2—Verify memory after every collect. 4—Verify memory before every allocation. 8—Zero-Fill free memory. 16—Pattern-Fill free memory.
e	Specifies when the memory pool should be expanded. The value represents the percentage utilization of the active pages that will trigger a memory pool expansion. Range is 0 to 100. The default value is 80. On a Macintosh partition, application memory does not grow. It always starts with the maximum.
g	Sets the percentage by which the memory pool is expanded. The default is 10 percent.
i	Incremental unit in pages for memory expansion or contracting. Range is 64 to 1,048,576. Default is 256.
n	Minimum number of pages managed by the memory manager. The value specifies the absolute minimum number of pages that will be allocated to the memory heap. Range is 1024 to 4194304. Must be less than the x memory option. The default value is 1024. See "Setting Maximum and Minimum Size of the Memory Heap" on page 120 for information about how n and x interact.

Memory Option	Description
r	Sets the minimum number of free pages needed to perform a shutdown. Range is 64 to 1,024. The default is 64.
u	Target average memory use. The value specifies the target percentage utilization of the memory heap, calculated as the proportion of allocated pages that are active. Specify this as a percent of currently allocated memory. Legal range is 25 to 95. The default is 85.
x	Maximum number of pages managed by the memory manager. The value specifies the absolute maximum number of pages that can be allocated to the memory heap. Range is 1024 to 4194304. Must be greater than the n memory option.The default value is 8192. See "Setting Maximum and Minimum Size of the Memory Heap" on page 120 for information about how n and x interact.

Setting Maximum and Minimum Size of the Memory Heap

To specify the maximum and minimum sizes of the Forte memory heap, use the n and x memory options as described in the previous table.

Most operating systems

For most operating systems, Forte follows these rules to determine the actual maximum and minimum sizes, based on the values specified:

When you specify only the value of n:

- If n is less than 1024, n is set to 1024.
- If n is smaller than the default value of x (8192), then x is 8192.
- If n is larger than the default value of x (8192), then x is also set to n. The values of the maximum and minimum memory heap sizes in this case are equal.

When you specify only the value of x:

- If x is larger than the default value of n (1024), then n is 1024.
- If x is smaller than the default value of n (1024), then n is also set to x. The values of the maximum and minimum memory heap sizes in this case are equal.

When you specify both the n and x values:

- x is set to the larger value specified, whether by x or n. The value of n is always the value specified.

Appendix C

Undocumented Fscript Commands

This appendix contains a list of Fscript commands that were documented for Release 1, but replaced by other commands. These commands are still supported, but only to provide backward-compatibility.

We recommend that you use only documented commands when you write or update your Fscript scripts.

About Undocumented Fscript Commands

Certain Release 1 Fscript commands have been replaced and removed from the documentation. These Release 1 commands still work with Release 3, so scripts you wrote for Release 1 should still work as before. However, we recommend that you use the documented Fscript commands.

The following list maps the replaced Fscript commands to the currently-documented Fscript commands:

Release 1 Fscript command	Documented Fscript Command	See:
AssignPart	AssignAppComp	page 36
CompileProj	CompilePlan	page 43
EnablePart	EnableAppComp	page 47
ExcludeProj	ExcludePlan	page 48
ExportProj	ExportPlan	page 50
FindPart	FindAppComp	page 53
FindProj	FindPlan	page 56
ImportProj	ImportPlan	page 56
ListProjs	ListPlans	page 62
MakeDistrib	MakeAppDistrib	page 65
ShowProj	ShowPlan	page 95
UnassignPart	UnassignAppComp	page 99

Index

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W